# Agent Triage, Git Planning, and the PM Build-to-Learn Shift

PM Daily Digest

2026-04-15

## Agent Triage, Git Planning, and the PM Build-to-Learn Shift

*By PM Daily Digest • April 15, 2026*

This issue covers three shifts reshaping PM work: classifying AI agent ideas before prioritizing them, moving planning and AI context into the repo, and narrowing the gap between prototype and production. It also includes practical playbooks, case studies, career signals, and tools worth using now.

### Big Ideas

#### 1) Agent prioritization needs an architectural first pass

Teams are often comparing "agents" that are not the same class of product. The proposed hierarchy starts with **Category 1: deterministic automation** (you define the flow; good for predictable workflows and 60-70% of opportunities), moves to **Category 2: reasoning and acting** (you define tools; the model chooses the path for ambiguous or multimodal work and roughly 25-30% of opportunities), and only later reaches **Category 3: multi-agent networks** for cross-domain coordination [1].

**Why it matters:** The architecture changes the skills required, delivery timeline, operating cost, and success metrics. That is why a deterministic email support agent could reach **87% workflow completion** and **$18K/month savings**, while a voice-and-image shopping assistant needed a richer reasoning loop to reach **86% task completion**, **91% image accuracy**, **+22% conversion**, and **$0.08/session** [1].

**How to apply:** Categorize every AI backlog item before estimating effort. Start with Category 1 if the process is flowchartable; move to Category 2 when the same request can trigger different action sequences; reserve Category 3 for multi-domain delegation and shared ownership across teams [1].


*Not all AI agents are created equal (1:30)*

**2) PMs are getting closer to production, but not by taking over engineering**

Aakash Gupta's role map moves PMs closer to **copy, config, prompts, planning docs in git, small front-end changes, and production monitoring**, while design moves into **coded prototypes, design system changes, and visual QA** and engineering stays concentrated on **architecture, infrastructure, security, complex logic, performance, and code review** [2]. Marty Cagan draws a compatible line: PMs **build to learn** in discovery, engineers **build to earn** in delivery, and PMs still own the **value** and **viability** risks [3].

**Why it matters:** Shipping small, context-heavy changes can tighten feedback loops and sharpen specifications, but it does not replace strategy, user research, or stakeholder alignment [2].

**How to apply:** Let PMs and designers own the work where they already hold the most context, but keep engineering focused on the hard problems. Use prototypes and lightweight production changes to learn faster, then keep PM accountability on outcome quality—not just output volume [2, 3].

> "When PMs build their own ideas, their specifications get sharper, because they now understand what the agent needs to execute well. Sharper specs produce better agent output." [2]

### 3) The prototype-to-production gap is narrowing

AI prototyping tools can now work from an engineering team's real front-end components and design tokens, which makes prototypes more consistent and easier to port into production [4]. A parallel shift is happening in agent delivery: one case study describes shipping a working Next.js knowledge chatbot in an afternoon using the same `.claude/` setup already used in development, with no new framework or translation layer between dev and prod [5].

**Why it matters:** Discovery artifacts are becoming more reusable, which lowers handoff waste between prototyping and implementation [4, 5].

**How to apply:** Prototype against the real design system early, and treat agent behavior as a versioned product asset: `CLAUDE.md` for identity, skills for reusable behaviors, MCPs for tools, hooks for safety and observability, and sub-agents as discrete folders [5].

## Tactical Playbook

### 1) Run a 5-minute agent triage before roadmap discussion

1. If the whole process can be mapped as a clear flowchart, start in Category 1 [1].
2. Use Category 1 tools such as **n8n, Zapier, Make.com, or OpenAI AgentKit** and measure **workflow completion, automation rate, accuracy, latency, cost per workflow, and human review rate** [1].
3. Move to Category 2 when the same request can lead to different paths, the agent needs **5-15+** capabilities, or user intent must be clarified through interaction [1].
4. Use Category 2 tools such as **LangGraph, CrewAI, or AutoGen** and measure **task completion, reasoning accuracy, conversation length, tool efficiency, cost per session, CSAT, and business impact** [1].
5. Consider Category 3 only when one agent is spanning too many domains, tasks run for hours or days, or multiple teams need specialized agents delegating to one another [1].

**Why it matters:** This prevents roadmap debates from collapsing into false effort estimates across incompatible systems [1].

**How to apply:** Use this checklist before scoping, staffing, or comparing ROI across agent ideas [1].

### 2) Put the spec in the repo with a four-part planning file

> "No more planning decks, only markdown pushed to a git repo." [2]

1. Write `PLANNING.md` in the same repo as the code so builders and AI can reference the spec directly and git history captures why something shipped

[2].
2. Keep four sections: **Problem**, **Hypothesis**, **Success Metrics**, and **Rollout** [2].
3. Use specific thresholds, not vague goals. The sample notification-batching plan sets a primary goal of **15%+ mute-rate reduction**, guardrails on **CTR** and **DAU**, and a **10% rollout** with a kill rule if improvement stays below **5% after two weeks** [2].
4. Add `CLAUDE.md` at the project root to encode product context, coding standards, review expectations, and PM scope boundaries [2].
5. Start in GitHub's web UI if needed; this workflow does not require a terminal to begin [2].

**Why it matters:** The content may look like a good PRD, but the location makes it versioned, accountable, and AI-readable [2].

**How to apply:** Fork a template, then write the next feature plan in the same repo where the code lives [2].

### 3) Use AI to enter strategy meetings with a position, not just a summary

Facilitation is useful early, but senior leaders are looking for people who simplify under complexity, stay calm under pressure, and bring others toward a point of view [6].

1. Ask AI to build context: gather data and research, inspect the codebase for build constraints, and map cross-functional incentives [6].
2. Pressure-test your position: ask for the strongest counterargument and what you might be missing [6].
3. Open the meeting with a view: "Here's where I've landed on this, and here's why," then invite feedback [6].

**Why it matters:** The ceiling for many PMs is not collaboration; it is the lack of a visible, defended point of view [6].

**How to apply:** Do the research and counterargument work before the meeting so the discussion starts from a candidate direction rather than blank-page synthesis [6].

### 4) Design guardrails and human escalation from day one

1. Add **input guardrails** for malicious prompts, sensitive data, and rate limits [7].
2. Add **planning guardrails** to validate actions, require approval for high-stakes operations, and limit scope [7].
3. Add **tool guardrails** for permissions, authentication, and destructive-action confirmation [7].
4. Add **output guardrails** for hallucination checks, sensitive-info filtering, source attribution, and compliance [7].

5. Define fallbacks such as graceful degradation, human-in-the-loop, safest-option defaults, retries, and circuit breakers [7].
6. Route to humans for high-stakes decisions, low-confidence cases, learning loops, and regulatory requirements [7].

**Why it matters:** Amazon's internal assistant used this pattern at scale—with PII redaction, action approval, access controls, source attribution, and human escalation—while serving millions of users [7].

**How to apply:** Treat guardrails, fallback behavior, and human escalation as part of v1 design, not a later hardening pass [7].

## Case Studies & Lessons

### 1) Facebook Share Bar: polished execution could not rescue a bad premise

Facebook's Share Bar wrapped external links in an iframe and added a sharing layer on top. The team spent time refining the mechanics and details, but users hated the result because it felt like Facebook was hijacking their web experience [8]. The lesson Soleio draws is bigger than UI polish: the best teams ask not only how to improve the experience, but whether the feature should exist at all [8].

**Why it matters:** Craft can hide premise risk until very late [8].

**How to apply:** Add a premise review before refinement work: is the feature respectful, useful, and trustworthy for the user—not just elegant on screen? [8].

### 2) Dropbox Carousel: late user truth got expensive

Dropbox's Carousel team invested heavily in features and polish before discovering a core adoption blocker: many users were reluctant to give Dropbox access to their full camera roll because they feared it would consume storage quota [8]. The core issue was not execution quality; it was that the team confronted the critical user concern too late [8].

**Why it matters:** Trust and adoption assumptions become expensive when they surface after months of work [8].

**How to apply:** Put the riskiest customer assumption in front of users while the product is still cheap to change [8].

### 3) Amazon's internal assistant: multi-agent systems work when routing and guardrails are first-class

Amazon built an internal AI companion for its global workforce across IT support, HR queries, learning recommendations, documentation, and productivity workflows [7]. The architecture uses a coordinator to route requests to specialized domain agents, integrates enterprise tools, and layers in guardrails plus

human escalation [7]. Reported results include **millions of monthly active users**, **70%+ weekly return**, **30%+ support-ticket reduction**, and **sub-second response times** [7].

**Why it matters:** This is a concrete example of agent systems creating measurable operational impact when orchestration, tools, and safety are designed together [7].

**How to apply:** Start with a small number of agents, invest in coordinator logic early, and monitor agent performance, tool usage, and user satisfaction as the system expands [7].

### 4) iPod and Facebook Groups: launch is where reality starts

> "Builders build. Then they ship. Then they solve what breaks." [9]

Tony Fadell's iPod example is a reminder that the first launch is not the final product: the first version shipped in **9 months**, then improved over multiple generations until it became durable enough to help pave the way for the iPhone [9]. Soleio describes a similar decision pattern on Facebook Groups: with roughly **90 days** and many stakeholders, the team chose a direction, shipped, and layered improvements later instead of stretching the project into endless consensus-building [8].

**Why it matters:** Shipping with a point of view matters, but staying with the product after release is what builds trust and longevity [9].

**How to apply:** Separate launch criteria from perfection criteria, then reserve explicit time for scaling, support, and iteration after release [9, 8].

## Career Corner

### 1) AI PM roles are paying and interviewing differently

Aakash Gupta says AI PM roles at Anthropic and OpenAI pay **north of $1M/year** in total compensation versus roughly **$280K** for a senior PM at a typical Series C SaaS company [10]. The interview loops also differ from standard PM prep: OpenAI uses AI product-sense and metrics cases, including a prompt about doubling ChatGPT image creation weekly actives from **175M to 350M** in three months with only three engineers, while Anthropic adds a dedicated safety-and-ethics round [10]. Across both, candidates are expected to discuss accuracy-latency tradeoffs, ML-engineer collaboration, and safety as part of the build process [10].

**Why it matters:** Conventional PM frameworks alone are a weaker fit for these roles [10].

**How to apply:** Shift prep toward AI-specific cases and stories; Gupta frames **40 focused hours** on those dimensions as the highest-ROI prep investment [10].

**2) AI product coaching is becoming a practical accelerator**

Marty Cagan argues foundation models have reached the point where they can act as practical product coaches with 24/7 access, and he estimates time-to-competency may fall from about **three months** to **less than half that** because coaching is available whenever needed [3]. The setup matters: specify whether you want **product-model** rather than **project-model** advice, tell the model to act as a coach that challenges rather than affirms, prioritize sources such as Teresa Torres, Shreyas Doshi, and SVPG, and load strategic context like vision, strategy, and team topology [3].

**Why it matters:** Coaching is scarce in many orgs; this makes structured practice more available [3].

**How to apply:** Use early sessions to build product sense around KPIs, users, industry dynamics, and techniques like Opportunity Solution Trees. For leadership nuance and politics, Cagan still recommends human coaches [3].


*Marty Cagan on AI Product Coaching (21:51)*

**3) Past a certain level, facilitation is not enough**

Facilitation helps PMs grow early, but the leadership bar changes when the room needs conviction rather than synthesis [6]. In one example, a PM named Marcus had been passed over as "collaborative" but not strategic; after doing the preparation work and entering a strategy session with "I believe we're solving

the wrong problem. Here's why," the room worked within his frame and his skip-level called it the kind of leadership the team needed [6].

**Why it matters:** Senior leaders are judged on whether they simplify, direct, and bring others along—not only on whether they make discussion smoother [6].

**How to apply:** Keep the collaborative posture, but do the private preparation that lets you enter the room with a clear, defensible view [6].

## Tools & Resources

### 1) A repo-based PM planning system you can fork

A public GitHub repo, pm-planning-system, includes `PLANNING-TEMPLATE.md`, worked examples, `CLAUDE.md`, a rollout playbook, a pilot measurement template, a weekly review cadence, and a planning review skill [2].

**Why it matters:** It gives teams a concrete starting point for the planning-in-git workflow [2].

**How to apply:** Fork it and write the next feature plan in the same repo where the code lives [2].

### 2) A practical guide to classifying agent ideas

Lenny's recommended guide, Not all AI agents are created equal, is built around a 5-minute triage process, tool guidance, tailored success metrics, and warning signs that you picked the wrong architecture [11].

**Why it matters:** It is useful when an AI backlog mixes quick automations with multi-month agent bets [11].

**How to apply:** Use it to classify backlog items before roadmap or staffing discussions [11].

### 3) The `.claude/` production-agent pattern

The Product Compass article Your .claude/ Folder Is a Production Agent argues that the same `.claude/` folder can serve as the deployable unit, with `CLAUDE.md` for identity, markdown skills for reusable behaviors, MCPs for tool access, hooks for safety and observability, and sub-agents as separate folders [5].

**Why it matters:** It reduces the mental translation between local experimentation and shipped behavior [5].

**How to apply:** Treat agent behavior as versioned product config, not scattered prompt snippets [5].

**4) Design-system-aware AI prototyping**

Recent AI prototyping tools are increasingly able to work from an existing design system; Sachin Rekhi highlights this as a major shift of the last six months and points to Bolt's enhanced design system agent as a recent example [4].

**Why it matters:** Higher-fidelity prototypes reduce rework between discovery and delivery [4].

**How to apply:** Route early prototypes through the same design tokens and components engineering already uses [4].

---

**Sources**


1. Not all AI agents are created equal
2. How to Ship Your First Pull Request as a PM
3. Marty Cagan on AI Product Coaching
4. X post by @sachinrekhi
5. Your .claude/ Folder Is a Production Agent
6. What People Get Wrong About Leadership
7. Scaling Multi-Agent Orchestration | Amazon AI Product and Technology Leader
8. How to Spot a World-Class Designer
9. X post by @tfadell
10. substack
11. X post by @lennysan