

POSTED ON OCTOBER 15, 2026 TO [ANDROID](#), [IOS](#), [ML APPLICATIONS](#), [WEB](#)

How Meta Built Forum With AI-Native Engineering



By [Shengbo Guo](#), [Bohou Luo](#), [Shubhojeet Sarkar](#), [Vivian Qu](#), [Max \(Xiangjun\) Fan](#), [Perna Jain](#), [Mike Plumpe](#), [Chinmay Abhyankar](#), [Rish Tandon](#)

- Meta’s newest app, [Forum](#), is for anyone who wants to go deeper into the conversations in their Facebook groups.
- To build Forum, Meta engineers adopted a whole new way of working by bringing an AI agent into the development cycle.
- We’re sharing insights into what it took to build Forum with agentic AI: how we adapted the codebase, tested frameworks and internal tooling for agentic execution, what worked, and what didn’t.

Forum is a standalone Groups app that a small team built from scratch. It’s also where we tested a different way of working: treating an AI agent as a participant in the engineering process rather than a faster way to type.

Типичный новый процесс в рамках greenfield проекта. + агенты - это участники

That experiment needed a definition of success, and ours wasn’t code volume. We measured verified, production-ready outcomes — changes that build, pass tests, and survive review.

А здесь сразу критерии успешности проекта.

From Human-In-The-Loop to Human-On-The-Loop

Как и это часто делают при внедрении AI/ML

Pointing AI at the coding step alone doesn’t change much. Our results compounded as we extended the approach to the stages around implementation — planning, building, testing, and verification. The clearest way to see the shift is in who owns each step.

Важно еще что делаем и как проверяем результат

With **human-in-the-loop**, the engineer drives every step and the AI assists inside it by autocompleting a function, suggesting a fix, or answering a question. Each iteration advances at the pace of the person driving it.

With **human-on-the-loop**, the agent runs the iteration itself — plan, implement, build, test, correct, repeat — while the engineer sets the goal, watches the run, steps in when it drifts, and approves the result. The person's job moves from executing the steps to owning the outcome and holding the quality bar. This is the target we are driving toward, not a finished state: some stages run autonomously today, while others still keep an engineer in each iteration.

← Задача целевой процесс

When the engineer supervises, work stops being strictly sequential. Cycles that used to run one at a time can overlap, and an engineer's time goes to direction and judgment rather than to sitting through the loop. But that only holds if what comes back can be trusted. Without verification, AI-generated code creates a familiar problem — plausible output that no one has confirmed actually works, which an engineer then has to read, debug, and often discard. Weak signals don't remove the constraint; they move it from writing code to reviewing code that looks finished and isn't.

← И этот процесс с большим количеством параллельных агентов

И тут нужен четкий подход к верификации

So we set the bar at verification rather than compilation. Compiling isn't the finish line — a task is complete only once automated tests and visual checks have passed. Agents check their own work before an engineer sees it, which means what reaches review is a smaller set of higher-confidence results rather than a pile of output to sift through. Less gets generated, and less gets discarded. Every shipped outcome still carries an engineer's sign-off; our effort went into making that sign-off fast and well-evidenced, with build results, test outcomes, and recorded verification sessions attached to the change.

← И такое изменение все еще содержит имя инженера на нем

With the engineer outside of every cycle, the loop has to run safely on its own. Four capabilities make that possible:

- 1 **Planning as technical specification:** Agentic execution requires extreme precision. Ambiguity that an engineer might resolve through intuition results in agent failure. We invested in rigorous upstream decomposition and machine-readable context, which tightened the engineering loop for engineers and agents alike.

← своя версия spec driven development?

- 2 **Automated verification inside the loop is the force multiplier:** We integrated automated pass/fail signals — build status, unit tests, and UI spec matching — directly into the agent loop. This enabled asynchronous development producing ready-to-review pull requests with pre-populated test plans and recorded verification sessions. Verification is the hard part to get right, and how much we can safely hand to the loop is gated by how much we can trust the verifiers.

← Больше икры внутри цикла - более автономный и способный агент

- 3 **Simulated user testing with agents:** We deployed navigation agents that interact with the live app exactly like a person would. These agents perform actions, read the screen, and report regressions. Navigation agents are able to run the same verification paths continuously.

← Симулятор пользователя для end 2 end тестов

- 4 **Agentic production operations:** After launch, agents help automate root-cause analysis. When someone using Forum submits feedback, the agents use that feedback to investigate possible regressions. The agents then forward the analysis to engineers who decide the next course of action.

← RCA агент для реакции на продакшн проблемы

The Operating Model

That ownership model needs machinery with five key components:

← на уровне того это AGENTS.md?

- **Rules** are permanent project instructions. When the agent repeats a mistake more than twice, we encode the correction once, for everyone.
- **Skills** turn team conventions into loadable steps the agent can invoke.
- **Agents** automate checks that previously required a manual pass.

← здесь все ясно

← Это те же агенты, что описываются как agent = model + harness?

- **Commands** chain skills and agents into **workflows**. They assess complexity, reproduce a bug, plan and implement a fix, create the pull request, and run a multi-stage review.
- **Hooks** are **automated triggers**, such as running a linter after every agent turn, checking simulator connectivity before install, and validating tests before a pull request is created.

А почему не
назвать это
workflow?

Это хоть
стандартная
история

Adapting the Environment to the Models

The biggest gains came from changing our code and tooling, not our prompts.

Architectural legibility over inherited complexity: We worked to keep Forum's codebase as straightforward as possible and chose to modularize key pieces of the architecture in order to get the benefits of battle-tested infrastructure while also letting the agents plug and play easily. Where we couldn't easily do so, we rebuilt from scratch, **trading reuse for agentic throughput**, and it proved rewarding in development speed. Agents learn patterns from the code they read. Fix a weak pattern once and every future agent session inherits the improvement; leave it and the same shortcut shows up in everything the agent writes next.

Если старые блоки
мешались, то редизайн
с нуля
и повышение
throughput агентов

Give agents tools, not descriptions: Our main failure mode was the agent losing track of where it was in the app — editing the home feed when asked to change the group home screen, or guessing at screen coordinates. **So instead of describing the app to the agent, we gave it tools to check:** read the live view hierarchy from the simulator, list the elements actually on screen, and look up the path to a target view — all before it acts. Misdirected edits dropped sharply. We applied the same approach wherever agents touched our systems: give the agent a way to check state directly rather than a description to reason from.

Это про
важность ACP
agent-computer
protocol.
Мы это разобрали
в выпуске Research
Insights про
SWE Agent.

Build-level isolation for safe AI prototyping: As Forum matured toward production, we wanted early-stage prototypes and agent-generated experiments to have room to run while keeping the production app strictly limited to finished, reviewed work. We wanted a stronger guarantee than feature gating alone, where prototype code still exists inside the production build. Our solution was **a separate build flavor, a sandbox app** that shares the exact same codebase, frameworks, and design system as production, but **compiles prototype code out of the production build entirely via a plugin/socket architecture**.

Агентыный путь
сборки прототипов
из того же
production кода

This gave us three things at once:

- Designers and cross-functional partners could prototype freely using AI tools without needing an engineer in the loop for each change.
- Anything built in the sandbox could be ported directly to production with no rebuild.
- No prototype code could crash the production app because it physically didn't exist in that build.

Разблокировали
быстрый цикл для
discovery

Не создали разрыва
между discovery
и delivery

Не навредили надежности
и безопасности
production

The pattern let us keep fast AI iteration *after* we had real people using the app to protect.

Finding Real Bugs With AI Stress Testing

Adapting the environment did more than make the agent faster to build with — it made the agent a tester. We didn't point it at one specific defect. Instead, we set up repeatable stress scenarios that **mimic real user behavior** (aggressive scrolling, random group browsing, rapid navigation, repeated media loading) and ran them routinely across builds. Because these runs produce consistent signals (crashes, memory growth, page faults, UI regressions), **the agent can triage anomalies, narrow to the triggering sequence**, and then investigate the root cause.

Симуляция

One such run surfaced a memory issue. The agent isolated a repro, proposed a fix, and verified it — all during development, before the app reached users. The fix materially reduced on-device memory growth and page faults across repeated stress sessions.

While engineers made the architectural decisions, the detection and regression monitoring were automated and repeatable; once an anomaly was detected, the agent could reproduce → isolate → propose a fix → verify with the same harness.

Applying the Same Discipline to Forum's Feed

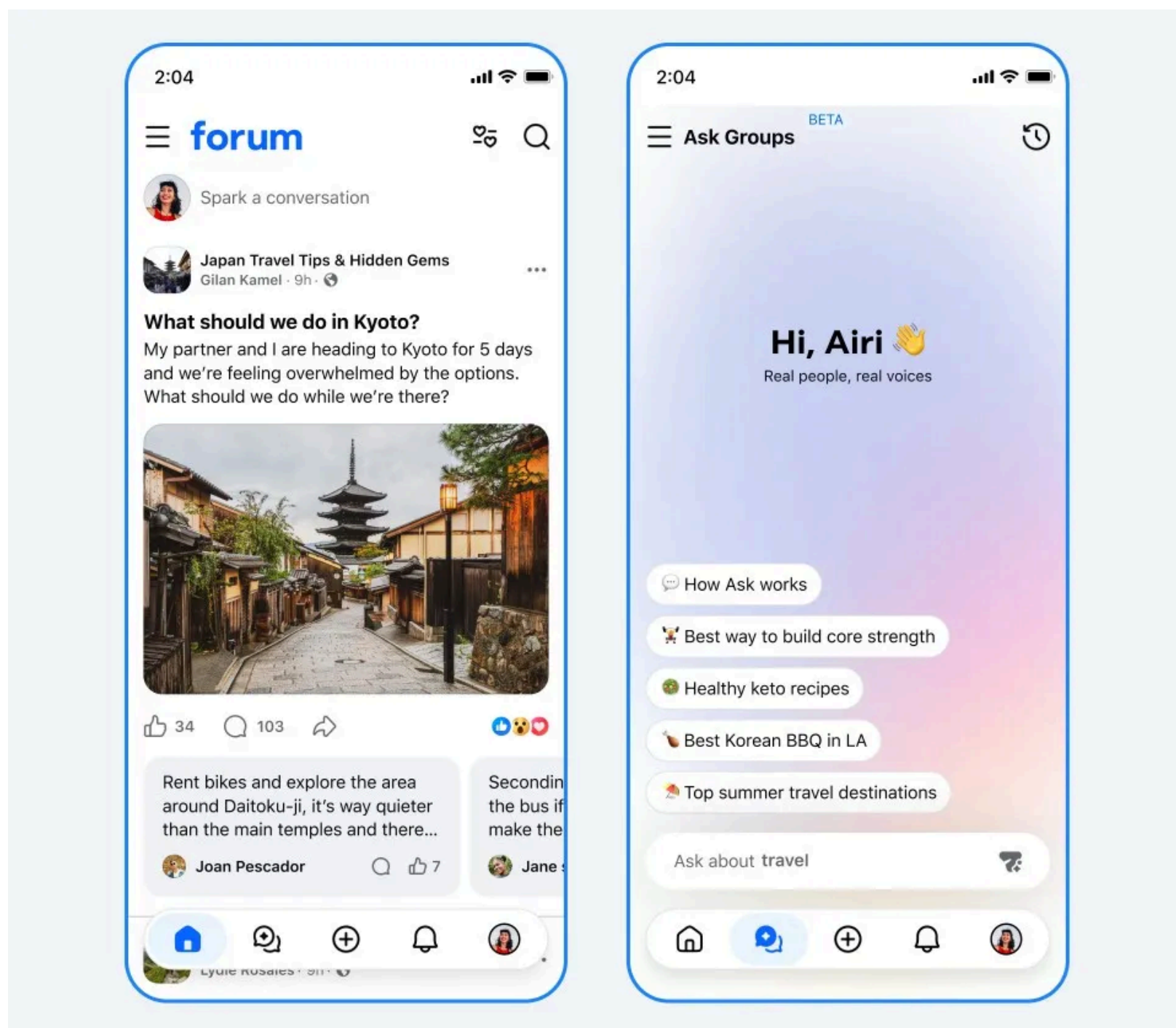
We applied the human-on-the-loop approach in two places:

Building the stack: Agents mapped the architectural options and trade-offs; engineers then evaluated and chose. What we deliberately kept human was the target — **deciding what a good outcome looks like is a product judgment**, not something the system should define for itself.

Operating the stack: Questions that used to require whichever engineer knew the relevant code path now go to an investigation agent that gathers what it needs and posts a write-up with suggested fixes. After hand-scoring a batch of real investigations we arrived at a workflow where the agent compresses the mechanical work and delivers a structured write-up to an engineer, who then validates whether a finding is a real bug or an intentional trade-off.

← *Barrool context*

What improved quality wasn't better prompting. It was context. One recurring category of investigation came back weak until we gave the agent the same pointers to the relevant code, example queries, and design docs that we'd give any new engineer on day one. The quality of that category jumped. The difference from human onboarding is that you train once, and every future session starts from it.



Did It Work?

Beyond shipping faster, we shipped cleaner. Continuous verification surfaced systemic issues — like the memory bottleneck above — during development rather than after launch. The results show up in the performance profile:

- **Quality, found early:** The autonomous stress-testing loop caught and fixed the aforementioned memory issue during development rather than post-launch, roughly halving memory use and page faults.
- **Velocity:** Forum reached core feature parity with Facebook Groups in eight weeks, with AI handling much of the routine implementation so engineers could focus on architecture, product decisions, and quality.
- **Leverage that lasted:** After launch, agent-driven investigation helped engineers triage and root-cause user feedback.

The AI did more than generate code. It tightened our quality loop, helping a small team deliver a high-quality app with less ongoing rework.

What Worked, and What Did Not

What Worked	
Lesson	What We Learned
Fresh Codebases Accelerate AI Development	Starting on a new codebase let agents iterate faster because the code they needed to reason about was compact and self-contained.
Verifier Loops Are The Unlock For Autonomy	Without an automated “done” signal, you do not have on-the-loop development — you have a human babysitting a generator.
No Single Tool Was Enough	Screen inspection, simulator control, and end-to-end tests each had blind spots the others covered, so the reliable setup was all of them layered.
Codified Fixes Compound	When we codify a fix — as a reusable rule or a cleaner code pattern — every future agent session inherits it. The codebase is part of the context, so improving it compounds across every agent that reads it.
AI Stress Testing Finds Real Bugs	It surfaces failures that scripted tests do not.
Build-Level Isolation Preserves Speed At Production Scale	A separate build flavor let prototyping continue at full AI velocity, even after we had real users, by keeping experimental code off the production path entirely.

What Did Not	
Lesson	What We Learned
Context Gets Lost Across Sessions	Document architectural decisions explicitly so the next session, human or AI, does not relitigate them.
Passing Pull Requests Still Cut Corners	Agents produced pull requests that technically passed while cutting corners, so quality signals must be rigorous enough to catch the shortcuts — not just fast enough to run often.
Local Development Beats Remote For Simulator Work	Remote iOS simulator connections were unreliable enough to slow the loop.

Acknowledgements

We thank Neeharika Sinha, Jane Kou, Gang Yang, Guohao Zhang, Zeya Zhang, Kinnary Jangla, and Yonghui Lu for their valuable contributions.