Vetting an AI Software Factory

Idea In Short

Before committing to an artificial intelligence [AI] software factory, ask providers for two things in writing: a walk through every approval gate with a named approver, and the total cost and duration of a finished comparable project in human hours and calendar weeks. A factory suits well-scoped work such as internal tools, new modules, proofs of concept and minimum viable products [MVPs], and it needs more care where requirements are unclear, test coverage is thin or regulation is heavy. Run a small pilot with agreed pass marks, name your approvers and free up their time, and insist on owning the code, the specifications and the documentation. Review and testing, not code generation, set the real speed.

The term "software factory" is about fifty years old. Japanese and American companies used it in the 1970s and 1980s for large, standardized programming operations.1 It is back now as the name for a way of delivering software in which artificial intelligence [AI] agents do most of the building. GitHub told CNBC this autumn that one in three pull requests (proposed code changes) on its platform now involves an AI agent.2 A year earlier it was fewer than one in ten. That leaves every engineering leader with the same job: rearranging delivery around agents. Of the options on the table, the factory goes furthest.

You can now buy an AI software factory from a consultancy, from a tool vendor or from a services company, and the three will describe different things. The questions below are the ones a chief technology officer [CTO] or product owner should have answered before a project goes into one.

What is an AI software factory?

The short answer is a delivery setup where AI agents write most of the code and engineers control what the agents may do. The "factory" part is the fixed line every piece of work travels: specification, then design, coding, review and testing. A one-line fix goes down the same line as a new feature.

The AI software factory model is, at bottom, a new division of labor. Agents produce the code and the tests. People write the requirements, approve the design and sign off each release, so responsibility stays with named engineers.3

BCG Platinion's analysis of the agentic software factory describes a model in which humans set direction while:

> autonomous AI agents build, test, and ship software solutions around the clock

One line in the same analysis matters more to a buyer:

> an agentic factory does not mean an uncontrolled one

Work stops at a checkpoint, called a gate, after every stage. Somebody's name is on each gate, and that person answers for what gets through.

A working factory has four parts:

1. A specification layer. Requirements go in vague and come out as descriptions a tester could check line by line
2. Agents with defined roles: analysis, design, coding, code review and test writing
3. A harness. This is the set of rules, coding standards, architecture constraints and tool permissions that every agent works within
4. Human checkpoints for architecture decisions, security review, quality assurance [QA] on staging and the decision to release

Put together, this is how an AI software factory works. A requirement enters as a written specification. Agents design, code, review and test it inside the harness. A person checks the result at each gate, and only approved work moves on.

How is it different from giving developers coding agents?

Most engineering teams already use coding agents. Stack Overflow's 2026 Developer Survey found that 73% of developers who use AI assistants or agents use them every day.4 Used that way, the agent lives in one developer's editor and helps with that developer's tasks. The team's process stays the same: the same sprints, the same code review, the same release schedule.

A factory changes the process itself. Agents are arranged into a pipeline. Work is cut into units small enough for an agent to finish alone. Engineers stop writing most of the code and work on specifications, the harness and gate checks.

Here is the catch with the first approach. One developer gets faster, so one step gets faster, and the work stacks up in front of the next step. Say a developer went from two pull requests a day to six. The reviewers, testers and release managers further down the line still work at the old pace, so the only thing that has tripled is their queue. This is why the number of a provider's engineers who use AI tools tells a buyer very little.

Why does this matter more in late 2026?

Data published this year shows the pile-up. Faros AI's AI Engineering Report draws on two years of telemetry (activity data from engineering tools) from about 22,000 developers across 4,000 teams. It compares the same organizations at low and at high AI adoption. At high adoption, task throughput per developer went up by about 34%, and pull requests were 51% bigger. Median review time went up fivefold. Bugs per developer rose 54%, incidents per pull request roughly tripled, and deployments per week fell by about 12%.

These were teams that added agents to an unchanged process. Reviewers received more code, in bigger pieces, and fell behind. Quality problems reached production. The report sums up its findings in six words:

> More code. Declining quality. Accelerating incidents.

A factory is one answer to this. It assumes that review and testing are the steps in short supply. So agents also write tests, check code against the harness rules and review each other's output before a person sees it. Work is cut small enough for a reviewer to get through each change in minutes. Whether a particular factory really does this is the main thing a buyer has to check.

How does the work split between agents and engineers?

The split varies by provider, though the published descriptions look alike. The Sombra AI software factory is a useful reference because the company publishes its process and timelines. Sombra engineers set up the architecture, rules and agents once per project. An AI-first business analyst writes the product requirements document. Inside the factory, AI software agents then go through design, code, review and test for each feature and each fix. People handle QA on staging (a test copy of the live system) and the checks before each release.

Sombra's page compares typical timelines for a conventional team and for its factory on suitable projects:

Milestone	Conventional delivery	Factory delivery
First working proof of concept	6-10 weeks	Under 2 weeks
Minimum viable product [MVP] in production	4-6 months	1-2 months
Small bug fix	3-10 days	Under 1 day
Scoped change	1-2 sprints	Often under 1 day

Source: the company's published figures, which apply to projects with a clearly defined scope.

Two cases on the same page give figures from real projects. On an order-to-customer relationship management [CRM] project, a prototype took two weeks against an estimate of three months, at about a sixth of the cost. An incident-reporting system needed 8.3 weeks of human work against an estimated 27, and a traditional build was expected to take more than six months.

Speed and cost are the AI software factory benefits these figures point to. There is a third

benefit that the figures do not show. A big change and a small one get exactly the same checks.

Engineers hold on to the risky calls: architecture, the data model, security, the decision to go to production. The typing goes to the agents, one to write the code and another to test it.

Which projects suit the model?

Some projects gain far more than others. Sombra's page says its factory is best suited to new functionality with a clearly defined scope.

Good candidates:

- Internal tools and portals that replace spreadsheets or software as a service [SaaS] products the company has outgrown
- New modules added to an existing product through clear interfaces
- Proofs of concept and MVPs, where learning fast matters most
- A steady flow of small fixes and scoped changes on a codebase with good tests

Projects that need more care:

- Systems with unclear or disputed requirements, where the hard part is deciding what to build
- Large legacy codebases with few tests, where agents have no safety net
- Safety-critical or heavily regulated software, where every change needs extensive human checking
- Work that depends on knowledge held by a few people and never written down

BCG Platinion draws a similar line between greenfield work (new systems), brownfield work (changes to existing systems) and legacy modernization. In its model each of the three has a separate assembly line and a separate harness. A provider who claims one factory covers them all deserves a few extra questions.

What can go wrong?

You can list the main AI software factory risks on day one, because nearly all of them

appear at the gates. There are five.

Weak specifications

Agents build what the specification says. Hand them a vague specification and you get working, tidy software that solves the wrong problem. Often nobody spots it before user acceptance testing. In a factory this gets expensive fast, because one misread requirement ends up inside every feature built on it. Review of the specification therefore matters most.

Review that becomes a formality

Reviewers who are sent more changes than they can read will approve them unread. That is what the Faros review and incident figures describe. So look at how the factory protects its reviewers: small changes, automated checks that run before a person looks, and a daily limit per reviewer. Queues that keep growing signal rubber-stamping. Ask to see queue length and approval time.

Tests written by the same agent that wrote the code

Let one agent write the code and the tests, and both will contain the same mistake. The tests go green anyway. Give the tests to a second agent, and have a person write the acceptance criteria or at least approve them. Independent authorship makes a green test meaningful. Ask who wrote each.

Drift from the architecture

An agent never notices that its rules are out of date. Change the architecture, forget to update the harness, and the agents carry on building for the old design until the codebase has two styles in it. The cure is boring. Keep the harness under version control, review edits to it like code, and make one engineer its owner. Then ask the provider for that engineer's name, and for the number of harness updates on their last project.

Ownership after delivery

Two weeks of building are followed by years of maintenance, and you may not want the factory for all of them. Put the source code, the specifications and the documentation in the contract. Check, too, that your own engineers or a support team of your choosing could take

the system over. Specifications and agent configuration should transfer too. Without them, extending the code gets hard.

What should buyers ask a provider?

Software factory AI claims are easy to make and hard to check from a sales deck. The questions below help tell a working factory from an ordinary team that uses coding assistants under a new name. Request the answers in writing, not in a demo:

1. Which steps do your agents handle, and which stay with people? Can we watch the actual pipeline run?
2. Show us one recorded approval for architecture, one for security and one for a release. Who gave each?
3. How many lines does a typical change contain, and how many minutes does a reviewer spend on it?
4. Are the tests written by the agent that wrote the code? And is it a person who sets the acceptance criteria?
5. When did an agent last get something seriously wrong on a client project? How was it caught?
6. On a past project like ours, what was the total cost and duration, in human hours and in calendar weeks?
7. On the last day of the project, what exactly is handed to us: code, specifications, agent configuration?
8. Six months after delivery something breaks. Who fixes it, and how?

If time is short, ask only 2 and 6. A provider whose factory is real can take you through every gate and has numbers from finished projects. Get the answers to both on paper.

What does it cost?

Pricing for factory delivery has not settled yet, and providers price it in different ways. Most quotes can still be split into three parts.

The first is setup. It happens before a single feature exists. Engineers work out the architecture, write the rules for the harness, configure the agents and plug them into your repositories and tools. It is a one-off charge, and on a small project it is a big slice of the bill.

The second is specification and review time. These are the people: business analysts and engineers who write requirements, approve designs and check what the agents hand back. Most of the human hours you are billed for should be here.

Last, the agents cost money to run. You are paying for model usage, for compute during builds and tests and for tool licenses. It is usually small next to human time. But it climbs with every iteration, so a sloppy specification costs you twice: once to run and once to fix.

To compare a factory quote with a conventional one, get both in the same units. That means human hours by role, calendar time and any usage costs that will be passed on to you. Sombra's order-to-CRM case, mentioned above, reports a cost roughly six times lower than the conventional estimate. Savings of that size need a clear scope. Expect less on projects with more uncertainty.

How should a first project be run?

Run a pilot first. Keep it small enough to finish in weeks, and real enough that someone in the business wants the result. Six steps cover it:

1. Pick work that has a clear edge to it. An internal tool is a good choice. So is a single new module, or a backlog of small changes
2. Settle the pass marks before anyone starts. You need acceptance tests and a deadline. You also need a budget and a number of defects you can live with
3. Name your approvers for specifications and releases, and free up part of their week
4. Watch the pipeline run live. You want to see a gate turn work away at least once
5. Hold your verdict for a few weeks. Let the result run in production first, or in a staging environment that mirrors production
6. Then put the real numbers next to the first estimate, and next to a conventional quote: hours, cost, defects

Step 3 is the one buyers get wrong most often. A factory moves only as fast as the client's reviewers. When an approval sits in somebody's inbox for a week, the project loses that week, however fast the agents are.

How does this change the build-or-buy decision?

The usual advice on internal tools has been "buy SaaS", because building your own took too long and cost too much. If a factory can put an internal portal into production in weeks, that argument gets weaker for some tools. A tool you build fits the way you work. There is no fee per seat, and you own it.

Keep buying payroll, email and accounting. The ones to look at are in between: the tool you have customized heavily, the tool your staff work around every day, the tool you pay for per seat for hundreds of users. For those, get a factory quote before the next SaaS renewal.

- 1Software factory
- 2About pull requests
- 3The AI revolution in software development
- 4The results of the 2026 developer survey are here

Summary

The factory model tries to fix a problem that this year's data shows clearly. Coding agents speed up the writing of code. When the delivery process stays the same, that speed turns into longer reviews, bigger changes and more incidents. A factory rebuilds the process around agents and puts engineers in charge of specifications, architecture and the gates where quality is decided. When you evaluate the model, look hardest at how the provider handles review and testing, start with a project that has a clear scope and measurable success criteria, insist on owning the code, the specifications and the documentation, and ask for cost and timelines in human hours with calendar weeks alongside. Then run one small real project from specification to release, and have your own engineers stand at the gates.