

Welcome to the dark testing factory:



6 things you need to know

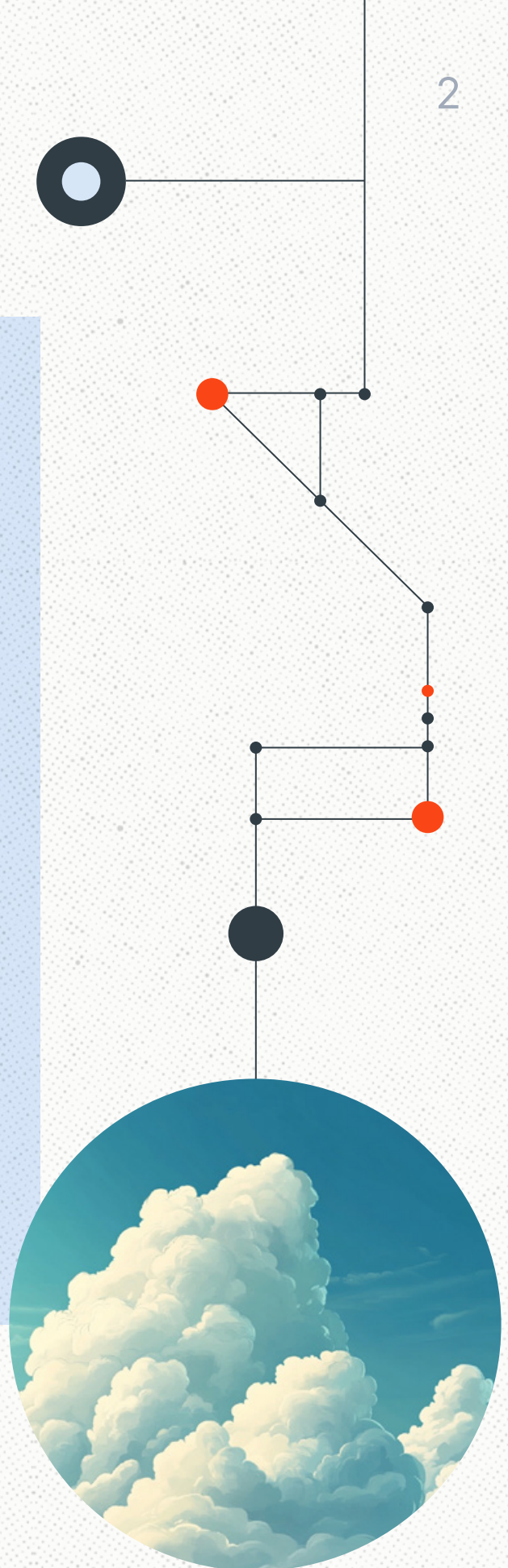
A self-operating quality system for accelerated software delivery:
autonomous testing at scale, guided by human context & judgment



UiPath

Content

- 3** The oldest testing problem is getting worse
- 4** The old model cannot absorb the new velocity
- 5** Autonomous testing moves from feature to factory
- 7** Humans do not run the factory, they teach it
- 7** Autonomy runs only on accessible knowledge
- 8** Real factories do not function in straight lines
- 9** Lights-out software testing meets lights-out software delivery
- 10** Six essential characteristics of a dark testing factory
- 11** The first dark factories are already emerging
- 12** The building blocks are already here
- 12** A fabric alone is not a dark testing factory



The oldest testing problem is getting worse.

If I had to reduce everything I have learned about software testing to one single sentence, it would be this: the time needed for testing is always more than the time we get.

A truth so reliable that it feels like a law of physics.

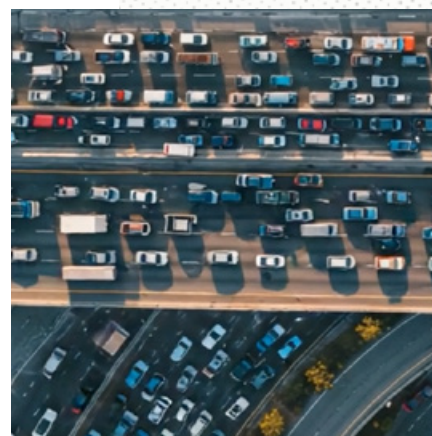
And this problem is not getting better, it's getting worse. AI is empowering software developers to create more code faster than ever before, while constant revisions create more change.

And more code plus more change means more risk that must be tested. The pressure is moving downstream: development accelerates, while testing is struggling to keep up.

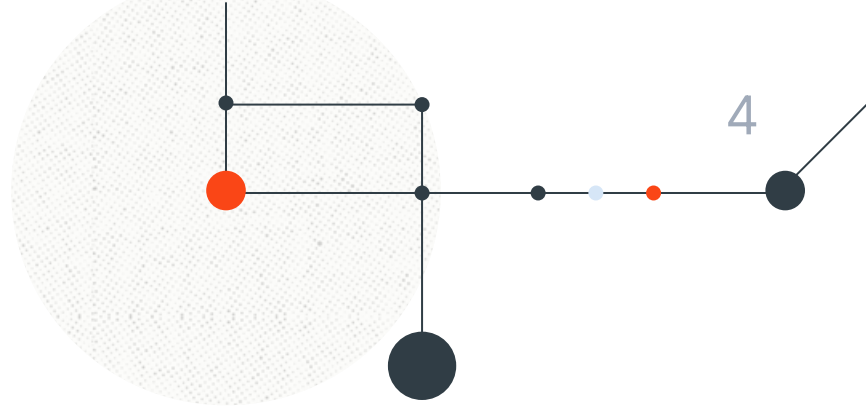
Faros AI calls this the Acceleration Whiplash. After analyzing two years of telemetry from 22,000 software developers across more than 4,000 teams, its **2026 AI Engineering Report** found that development productivity is rising: 60% of AI-generated code is now accepted into codebases. Developer output rose sharply, with task completion up 34%, epics completed per developer up 66%, and code-related tasks up 210%.

But quality is moving in the wrong direction: bugs per developer increased by 54%, the incident-to-PR ratio tripled, and oversight weakened, with 31% more PRs merging without review. This is the whiplash: development is accelerating faster than the quality system around it.

My read is not that AI is the problem, but that AI is making an old problem impossible to ignore: Testing has too often been treated as a second thought, something to perform after the "real dev work" is done. But when development speeds up this dramatically, testing must speed up just as fast. Not by adding more effort, but by changing the operating model of testing itself.



The old model cannot absorb the new velocity.



That is where the dark testing factory comes in, and it's not a villain origin story for software testing and AI. It's a new operating model for quality engineering in an AI-accelerated world.

In manufacturing, a dark factory describes a production line automated enough to run without people on the floor. In software testing, it translates to this: a self-operating quality system that plans, designs, runs, and maintains tests autonomously, end to end.

Think of the dark testing factory as the next evolution of autonomous testing: it's not just a smarter test runner or another agentic testing feature — but autonomous testing as an operating model.

This evolution has happened in stages over the last few years:

1.

Automated testing:

The classic “if this, then that” model of automation. You script every step, the machine follows. The agency sits entirely with you, the human.

2.

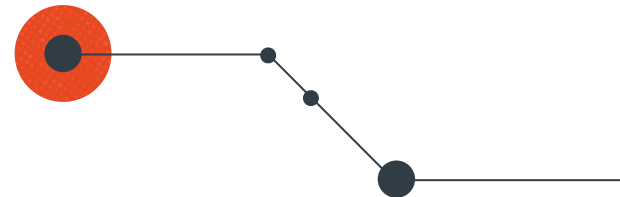
AI-augmented testing:

You prompt AI, it guides. But you still steer the course, review the output, and decide what happens next. The primary agency remains with the human.

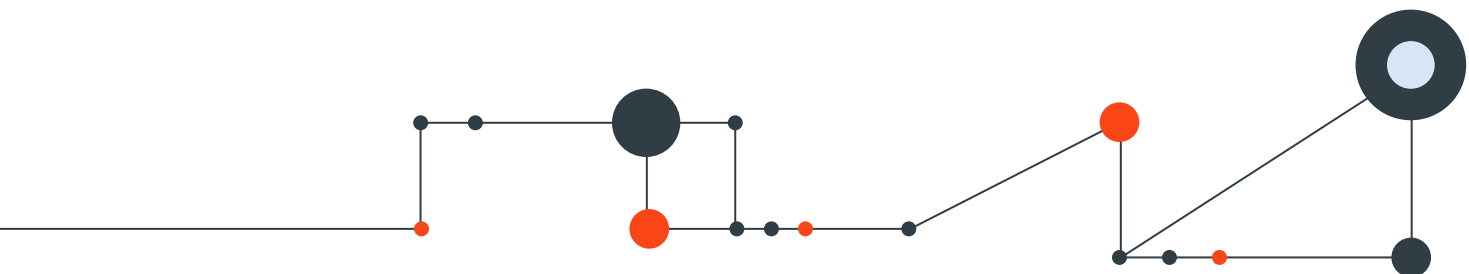
3.

Autonomous testing:

You state intent for a focused capability: execute this manual test, explore this application. The agent takes operational agency, but only inside that lane.

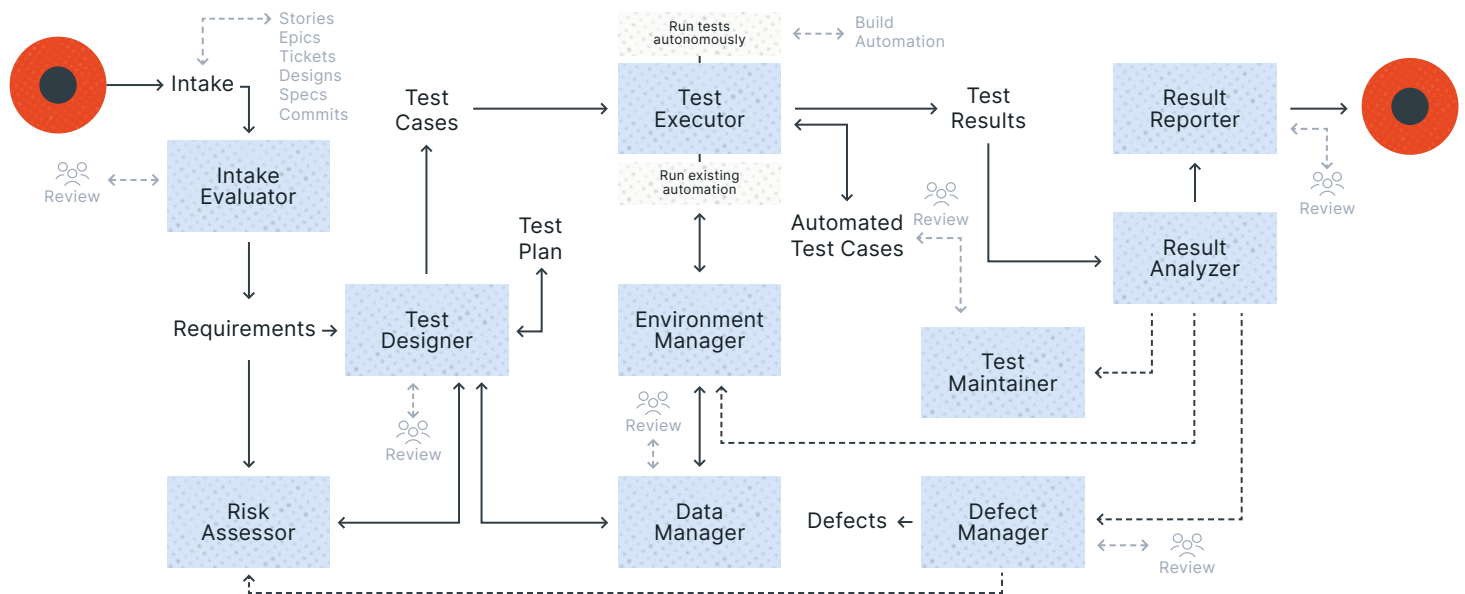


In a **dark testing factory**, autonomous testing stops being a point capability and becomes an operating system. Testing stops being something you run and becomes something that runs.



Autonomous testing moves from feature to factory.

Before we unpack the model, it helps to start with a mental picture without getting lost in detail. The diagram below is a blueprint — not a product architecture. It shows the factory's operating layer: the fabric. This is where agents, automations, and humans perform the work. The full dark testing factory needs a few more critical components, and we will explore those later.

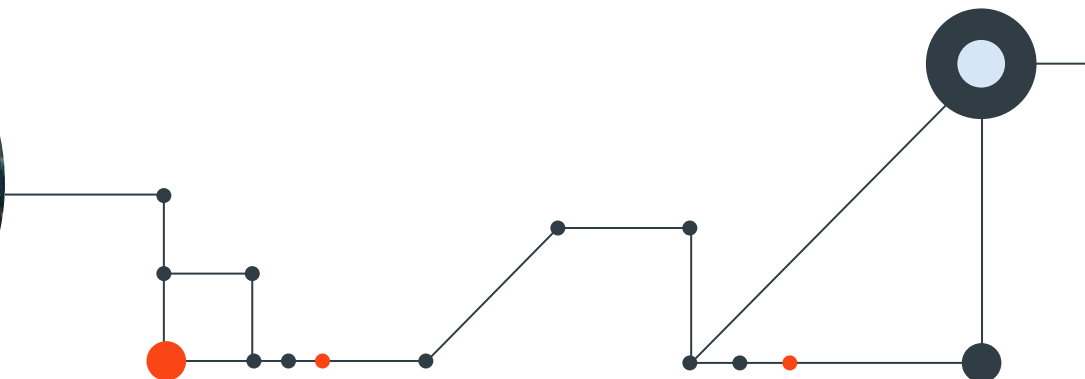


This blueprint abstracts the patterns we have learned from customer implementations to show the key idea: intent flows in, agents and automations coordinate the work, and humans step in only when required. A simple way to read the blueprint of the fabric is from left to right:

- It starts with intake: new or updated user stories, specs, production incidents, or commits entering the fabric. The **intake evaluator** refines this information and checks whether the input is clear, complete, consistent, and testable, then derives requirements for testing.
- From there, the **risk assessor** estimates what matters most based on business impact, change context, and historical signals. This is one of the hardest and least solved parts of the fabric: it must be grounded in enough business context to understand not only what has changed but also how much it matters (e.g., compliance risk, operational risk).

- The **test designer** then turns the requirements into new tests and updates the existing test portfolio: adjusting tests affected by the change, removing outdated tests, or merging redundant ones, and shaping the test plan and strategy for test execution.
- Then the fabric prepares for execution. The **data manager** creates or identifies the data needed to run the tests. The **environment manager** ensures that the right environment is available and that the supplied data can be used there. Depending on the setup, that may mean configuring an existing environment, provisioning a new one, tearing it down after the run, or triggering a rebuild when something is broken.
- The **test executor** decides how to run based on the risk profile: existing automated or manual tests, or newly designed manual tests. Existing automated tests can run directly. Existing or newly created manual tests are run fully autonomously through computer-use technology, without prebuilt automation. During execution, the fabric captures context and automates manual tests on the fly so future runs become faster and cheaper.
- The loop then closes through analysis and maintenance. The **result analyzer** interprets the outcomes, classifies issues, and triages them. Found defects flow into the **defect manager**, which compiles the evidence, reproduction context, impacted requirements, logs, and risk information, then prepares or files the defect for the right team.
- Automation issues flow into the **test maintainer**, which fixes broken automated tests (e.g., those that cannot be auto-healed), refactors automated tests to improve stability or execution speed, and flags tests for retirement when past execution history suggests they may no longer be needed (e.g., tests that have not been executed for many cycles).
- Environment issues are redirected to the **environment manager**, which handles blockers like unavailable services, missing data, or broken dependencies for a rerun.

Every result updates the risk model, making the next cycle more informed. Humans still review and approve where needed, but that distinction matters: only when required. That sounds simple, but it is also where the idea of the dark testing factory is easiest to misunderstand.



Humans do not run the factory, they teach it.

Before this starts sounding like a plan to remove humans from testing, let's be clear: that is not the point. A dark testing factory is not humanless. Humans are moved to where they create the most value: not inside the routine operating loop, doing the mundane work — but above it.

In a dark factory, testers no longer click buttons, rerun failed tests, connect requirements to test cases, assemble regression test sets, prioritize what to run based on the last cycle, chase environment issues, escalate routine failures, or file defects by hand. Instead, they define intent, resolve ambiguity, supply missing knowledge, and handle exceptions where judgment matters.

Put simply, humans do not disappear, but get a much-deserved promotion. As the dark factory handles repetitive tasks, humans have more time to strengthen the knowledge the factory relies on. They move away from 'doing' and towards 'orchestrating'.

The dark factory combines the scale, consistency, and tireless execution of agentic AI with the human ability to provide context, exercise judgment under uncertainty, and decide what matters.

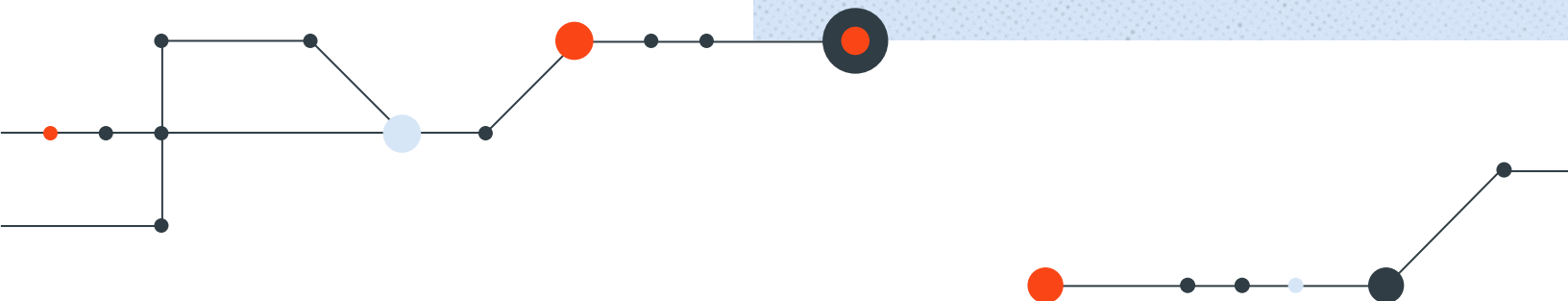
Autonomy runs only on **accessible knowledge**.

Let's zoom in on that last point, the information humans provide. The deeper constraint won't be (and isn't) technological; it's the data and knowledge problem.

A dark testing factory can only reason over information it can access, trust, and interpret. But in enterprises, most of the knowledge is tacit: business rules live in people's heads, undocumented exceptions are learned through experience, design documents may lag behind implementation, acceptance criteria are incomplete, and production behavior may diverge from what the official artifacts say. AI does not magically eliminate this gap; at best, it makes the gap visible.

This is why the human role becomes more important, not less. Every human intervention teaches the factory something: a missing rule, a source-of-truth decision, a clarified intent, an approved exception, a better test oracle. In a dark testing factory, this knowledge becomes reusable, machine-readable, and part of how the factory operates and how it gets darker over time.

Human judgement is the necessary, critical foundation for dark factory knowledge that becomes sharper cycle after cycle, exception by exception — until fewer and fewer situations require the lights to come back on.



Real factories do not function in **straight lines**.

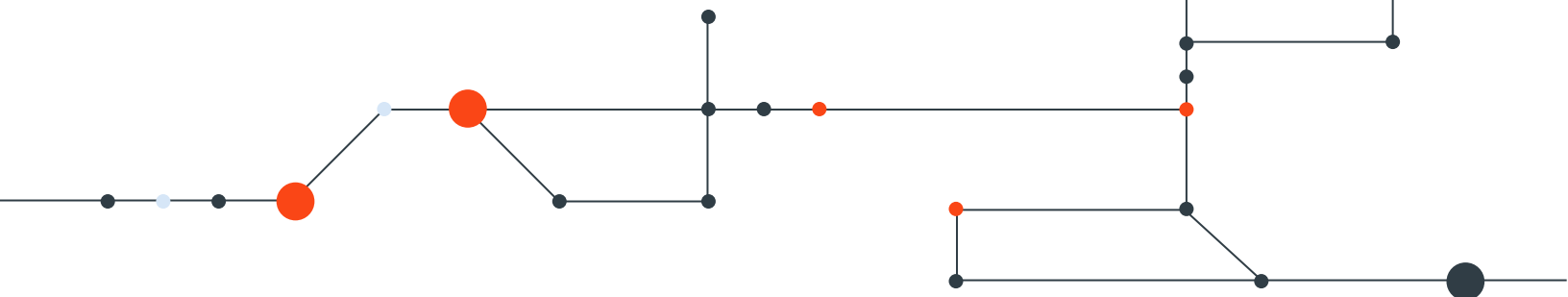
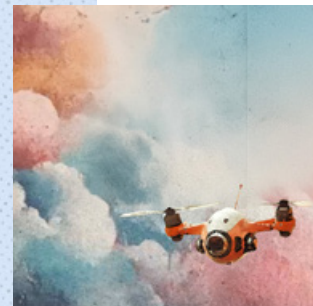
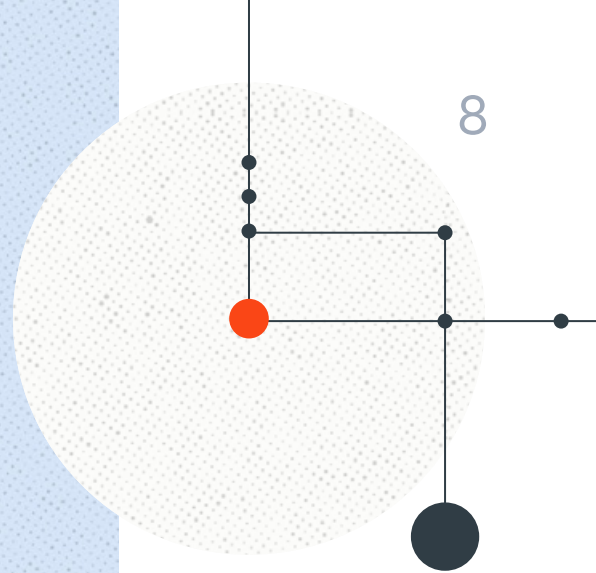
The diagram above is intentionally simplified. It may look like a predefined sequence of steps: agents here, automations there, and humans at approval gates. But a dark testing factory is not a linear pipeline where step 1 always leads to step 2 and never loops back.

It is a closed, nonlinear system: a loosely coupled ensemble of agents, automations, and humans continuously deciding the next best action based on what the factory observes and learns fully autonomously, and what humans have clarified along the way.

A failed test run may trigger test repair, data reset, environment rebuild, defect filing, or human escalation. A human response may send the factory back to redesign, regenerate, re-execute, or update its knowledge base. A new result may change the risk model. A failed test may change the next regression test set. A clarified business rule may become a new test oracle.

That ability to loop, branch, learn, and recover is the real value of the factory: it doesn't just follow a predefined path. It adapts as reality changes, continuously reshaping the work around the outcome the factory is designed to achieve.

It's not a pipeline, but a living loop — and that loop must live where software changes happen.



Lights-out software testing meets **lights-out software delivery**.

A dark testing factory cannot sit just next to the software delivery lifecycle like a separate software testing island. Software delivery is a chain, and the chain still moves at the speed of its slowest link. If development, build, deployment, and operations become more autonomous while testing remains a flood of manual handoffs and interventions, the whole system still slows down there.

That's why the factory must live inside the flow of delivery, listening to signals like code changes, build results, deployment events, production incidents, defects, and release decisions.

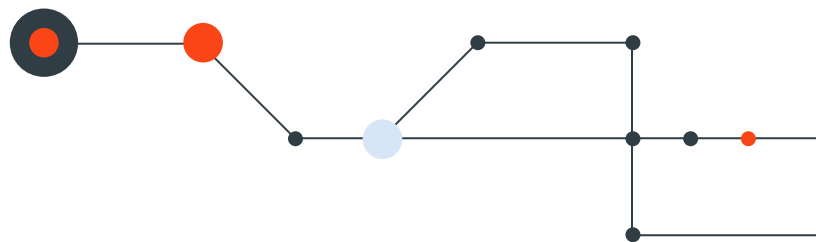
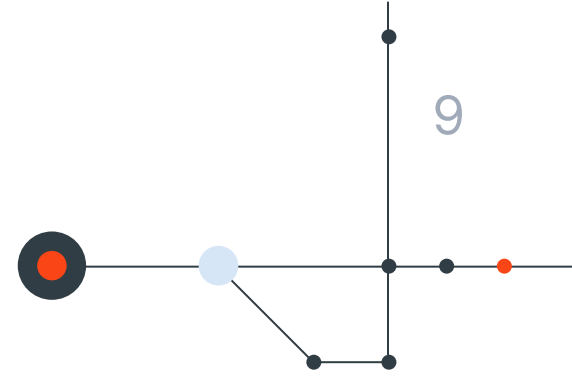
This is where Mirco Hering's idea **of lights-out delivery** is useful. In his framing, the question is not simply how much AI ought to be added to software delivery, but what kind of AI-driven delivery model we are building toward? Is it one where humans continuously observe and direct the work, or one where parts of the system can proceed autonomously within the right guardrails?



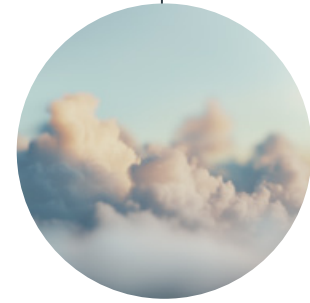
My take is simple: when we design dark testing factories, we should expect an intensive phase of lights-on learning: Humans will observe, redirect, correct, and approve. But the goal is not to keep those checkpoints permanently in the loop, but turn those interventions into factory knowledge, so the system becomes more capable the next time it runs.

We are not pursuing speed at the expense of control. We are building a framework that lets us move fast with confidence, where safety, security, and governance are not barriers to delivery, but the guardrails that makes rapid delivery trustworthy.

For software testing, this is exactly where the dark testing factory fits: it is the quality engineering layer of that lights-out delivery model. As delivery becomes more autonomous, software testing must move with it. The factory listens to the same signals the delivery pipeline already produces, and based on those signals, it decides what testing work needs to happen next. A dark testing factory is a continuous quality system embedded into the broader flow of software delivery.



Six essential characteristics of a dark testing factory.



Taken together, these ideas give us a practical way to think about a dark testing factory. Not an exhaustive checklist, but the patterns we see emerging from customer implementations:

Humans above the loop

Humans set policy, resolve ambiguity, supply missing knowledge, intervene on exceptions, but don't push buttons

Always-on operation

The factory runs continuously, 24/7, decoupling testing throughput from shifts, holidays, and time zones

Closed-loop quality

Failures are detected, diagnosed, retried, reworked, rejected, and/or escalated by the factory itself

Risk-driven scheduling

The factory decides what to test next based on change impact, production telemetry, business risk, and policy

Self-improving system

Human decisions become reusable, machine-readable knowledge, making the factory darker over time

Fixed-cost scaling

Additional coverage, variants, and releases become software/configuration changes, not proportional headcount

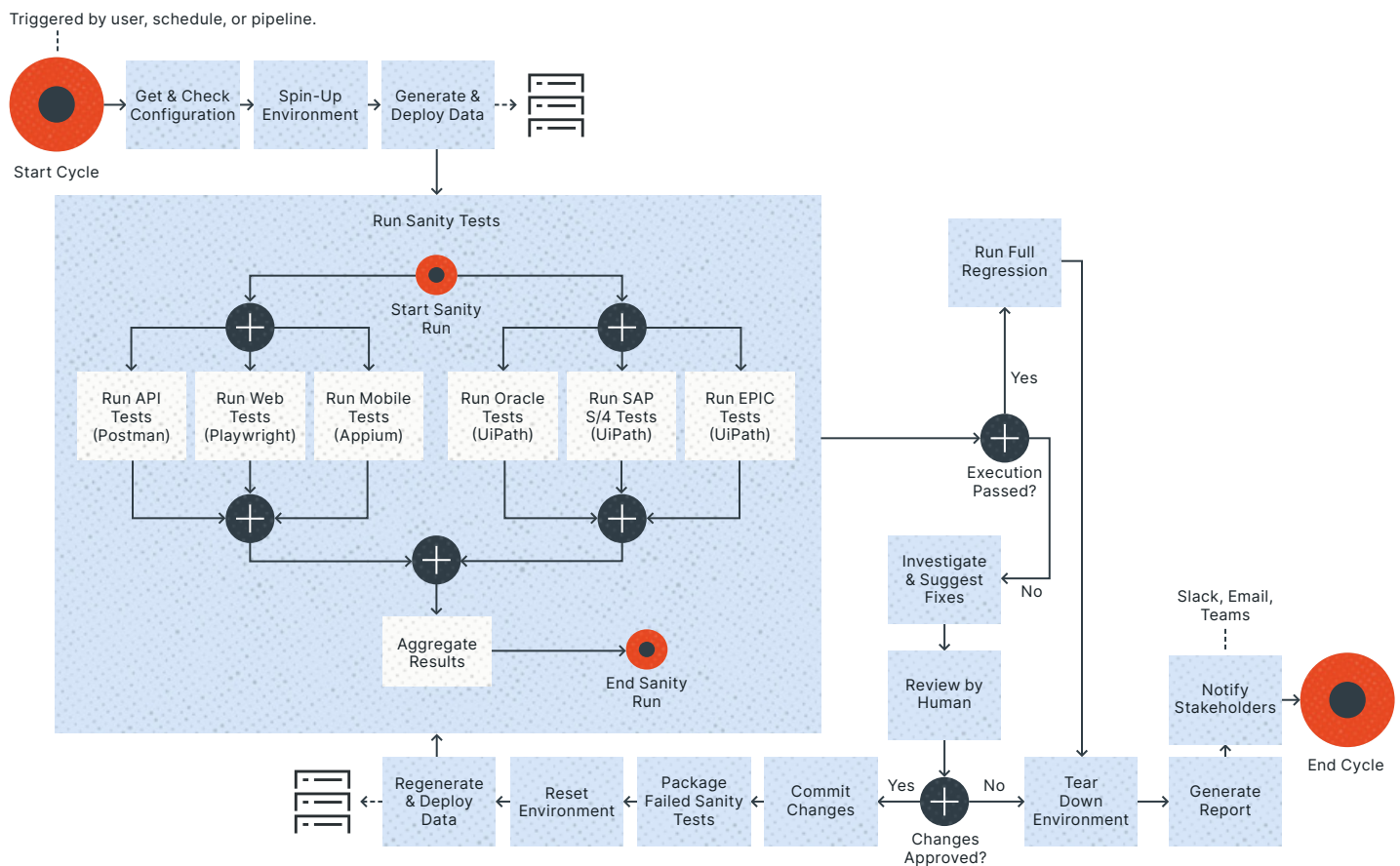
It removes humans from routine operation, runs continuously inside delivery, closes the loop on quality, schedules testing work based on risk, and improves through every human intervention. Fixed-cost scaling means repeatable testing effort becomes reusable software and knowledge, so deeper or broader test coverage does not require proportional headcount.

This means that the factory turns testing from labor that repeats into software that scales.

The first dark factories are **already emerging.**

So far, we have talked about the why and the what. The next question is obvious: how much of this is real? Our honest answer is: none of our customers have built the full dark testing factory yet. It's still ahead of us, and our move toward it is inevitable.

We already see forward-thinking customers pushing hard toward this model, often because their broader software delivery is already becoming more autonomous. In that sense, the dark testing factory is not a concept we invented, but a name for the direction our most advanced customers are headed in.



Today, these factories are early fabrics: autonomous end-to-end flows where automations spin up and tear down entire test environments, agents generate and deploy test data, automations execute tests across different stages and tools, and agents analyze results, suggest fixes, and route issues. Humans review and approve the actions taken by the agents, but more of the routine operating work is handled by non-human actors.

The screenshot above shows one slice of that operating layer in UiPath Test Cloud, leveraging our orchestration layer UiPath Maestro. Here you can see how the lights start to dim.

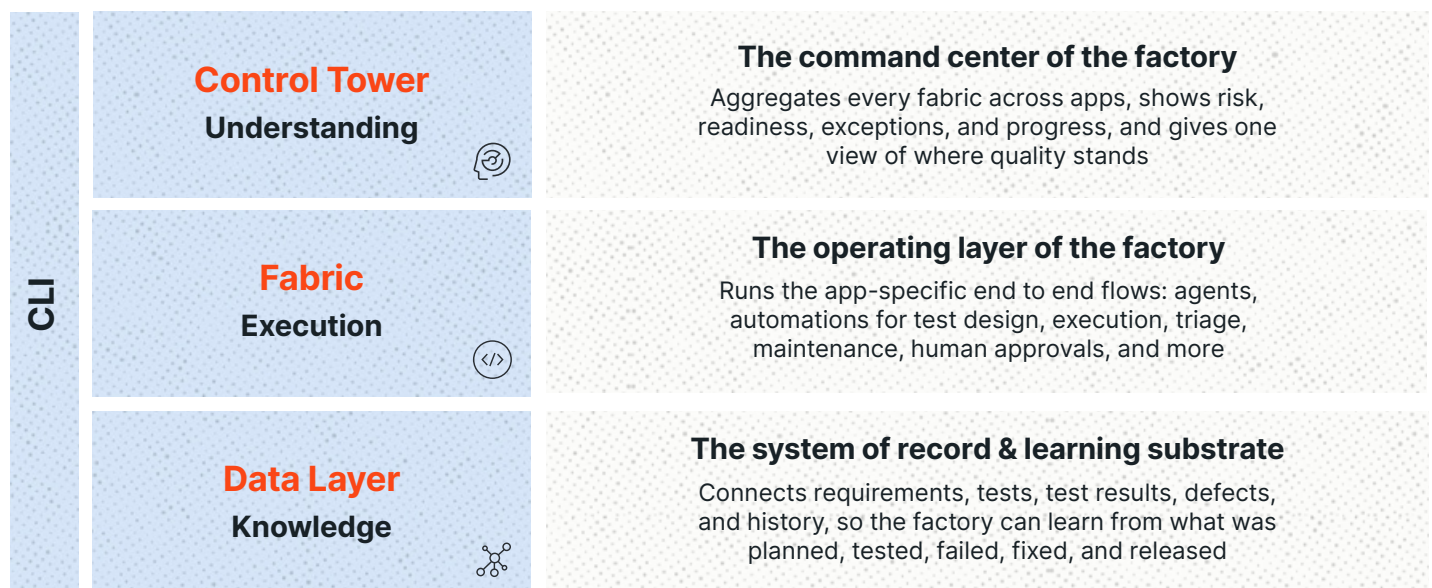
The building blocks are already here.

Our customers build these fabrics by combining several capabilities of **UiPath Test Cloud**. Test Manager anchors the quality knowledge: requirements, test cases, test sets, results, and defects. Maestro orchestrates long-running, autonomous flows. Agent Builder creates specialized agents. Studio provides the automations that execute work across apps, environments, and tools.

The main point is:
the building blocks for the fabric of the factory are already here, and this is what customers are already leveraging. But the fabric is only one layer of the factory.

A fabric alone is not a dark testing factory.

The fabric is where the execution happens, but execution alone is not the factory. What we have learned from customer implementations is that bringing a dark testing factory to life requires a three-layer architecture: execution needs understanding above it and knowledge below it. The following chart breaks down three key components of the dark testing factory: the control tower, the fabric, and the data layer:



A fabric alone is not a dark testing factory. (continued)

1.**In the middle is the fabric:**

the operating layer. This is where agents, automations, and humans coordinate the actual testing work for each application or domain.

In practice, one dark testing factory will often contain multiple fabrics: one for enterprise apps such as SAP, one for web, one for mobile, one for desktop, and so on. That is not only because the apps are different, but because the teams, delivery models, environments, tools, test data needs, and governance constraints around them are different too.

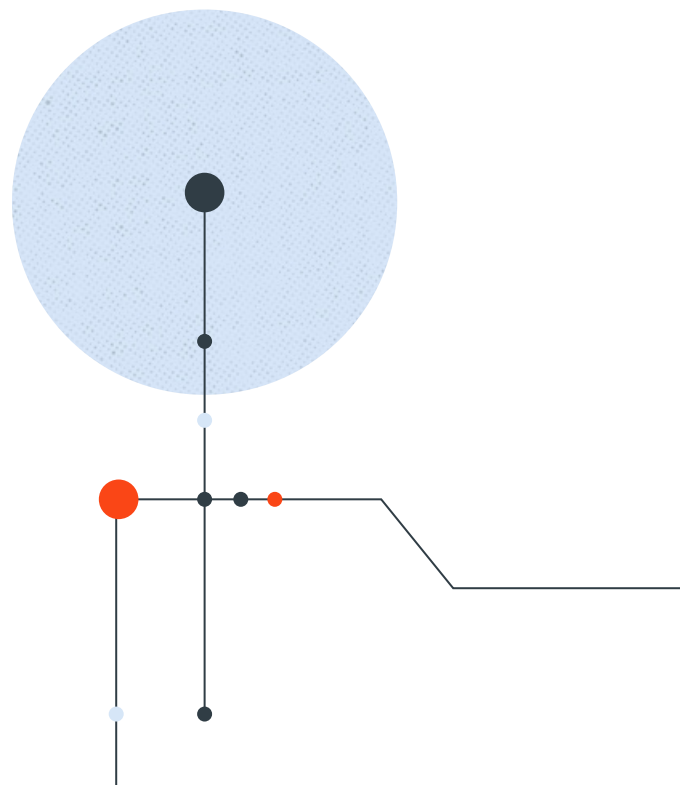
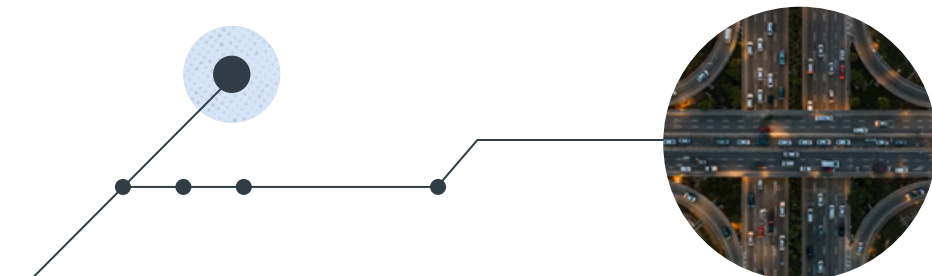
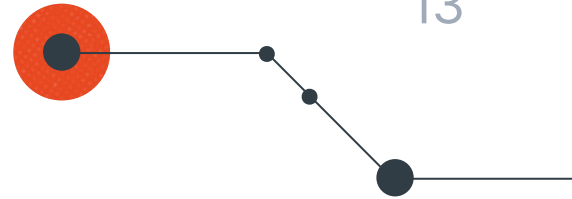
3.**Underneath the fabric is the data layer:** the system of record and learning substrate.

Today, this is anchored in UiPath Test Manager, where requirements, test cases, test sets, test results, and defects come together. This is not just storage. It is the underlying data model the factory learns from. Past results, recurring failures, defect patterns, flaky tests, and release history can inform risk profiles and help the fabric decide where to focus next.

2.**Above the fabric is the control tower:**

the command center of the factory, a place to understand quality, risk, readiness, exceptions, and progress across all fabrics.

But it is more than a dashboard. It is where autonomy is governed at the fabric level: risk profiles, policies, escalation rules, and approval thresholds define when a fabric can proceed on its own, when it must ask for human approval, and when it must escalate. This is the component we are now building out through customer implementations. This layer decides how much autonomy a fabric gets, the fabric decides how to execute within those boundaries, and agents, automations, and humans perform the work.

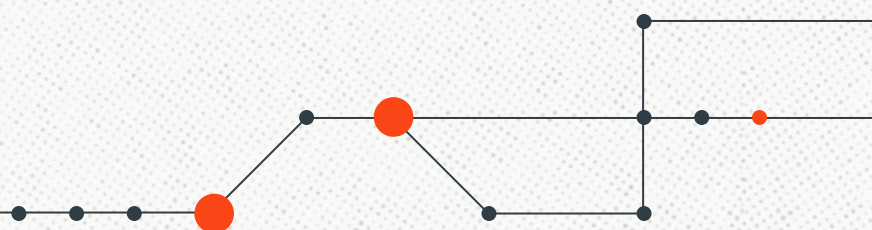


Understanding above, execution in the middle, and knowledge underneath:

That is what turns isolated pockets of autonomous testing into a self-operating system for quality engineering.

And that is the direction we are traveling right now: The first fabrics are already there, customers are starting to dim the lights, and the dark testing factory is moving fast from vision to reality.

Curious where this goes next? Follow **UiPath Test Cloud** for more on the future of testing.



About UiPath

UiPath develops AI technology that mirrors human intelligence with ever-increasing sophistication, transforming how businesses operate, innovate, and compete. The UiPath Platform™ accelerates the shift toward a new era of agentic automation—one where agents, robots, people, and models integrate seamlessly to drive autonomy and smarter decision-making. Powered by the UiPath Platform™, UiPath Test Cloud augments testers with enterprise-ready, production-grade, test automation for 190+ technologies and AI agents for agentic testing—the next evolution of software testing.

Explore UiPath Test Cloud

UiPath