

Heuristics of Software Testability in the Agentic SDLC

Version 1.5, 2026, by Lalitkumar Bhamare
Extending Heuristics of Software Testability, v2.9, 2025, by James Bach, Satisfice, Inc.

AI agents now take part in building and testing software, alongside human testers. This document extends James Bach's heuristics of software testability to that setting.

Its central argument is that testability now decides more than how easily a product can be tested. It decides whether an AI agent's work on that product can count as testing at all. Testability for agentic development is therefore the precondition for agentic testing, and an investment decision for every team that adopts AI agents.

CONTENTS

Why testability needs rethinking when AI agents take part in delivery	2
AI agents as testers	3
Practical testability	5
Testability dynamics when AI agents take part	8
Addressing testability when AI agents take part	10
Principles that apply across the guidewords	12
Guidewords for analysing testability	13
Products that learn	21
Conclusion and next steps	22
Definitions and attribution	23

Why testability needs rethinking when AI agents take part in delivery

More and more software is built through AI, and more and more of it is tested with AI agents. Testers use AI agents to explore a product, generate and run checks and analyse results. In some teams, part of the testing is delegated to AI agents while human testers orchestrate it: they set the mission, supply the context and judge what comes back. How much any tester can find out, human or AI agent, still depends on the product's testability.

James Bach's *Heuristics of Software Testability* ends its dynamics with a plain instruction: the tester must ask for testability, because no one else can be counted on to. That request has always travelled through people. When it works, a tester stuck on a state they cannot see raises it at the stand-up, and by the next sprint someone has added the log line.

Each of these ways of working moves the obstacle further from the person who would have asked. A human tester working alone meets the state they cannot observe, the environment that will not reset or the requirement with no clear intent, and asks someone, or reports that the product is hard to test. A tester assisted by AI agents meets fewer of these obstacles first-hand, because the AI agent meets them first. A tester who orchestrates delegated testing sees only what the AI agent chooses to report.

Figure 1. Three ways of working, and how far each puts the obstacle from the person who would ask

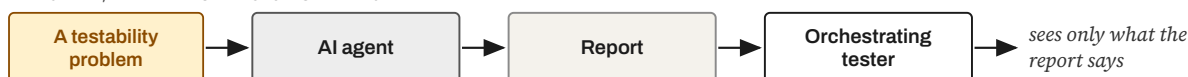
WORKING ALONE



ASSISTED BY AI AGENTS



DELEGATED, WITH A HUMAN ORCHESTRATING



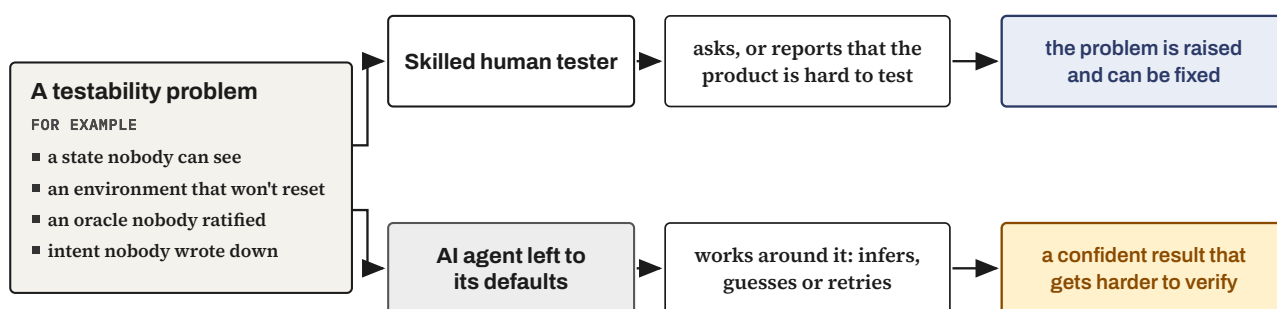
The more of the testing that passes through an AI agent, the less of the obstacle reaches a person directly.

What the AI agent does with the obstacle depends on how it was built and instructed. Left to its defaults, it tends to complete the task: it works around the obstacle, infers what it cannot see, and reports a result that seldom says how much of it rests on inference.

Bach warned about exactly this kind of help. A tool can make testing seem easier while the testing gets worse; in his words, "Bliss may be ignorant." An AI agent left to its defaults is the most capable such tool testing has had. The request a tester was counted on to make no longer arrives by instinct, and where the software is also built through AI, the problems nobody raised accumulate faster than people can review the work.

Figure 2. The same testability problem, met by a skilled human tester and by an AI agent left to its defaults

THE SAME TESTABILITY PROBLEM, TWO TESTERS



With a skilled human tester, the problem surfaces as a question or a complaint. With an AI agent left to its defaults, it becomes part of a confident result that gets harder to verify. What an AI agent built differently does with the same problem is the subject of the section on addressing testability.

The heuristics of testability hold as they stand, and human testers remain at the centre of testing. What has changed is the context in which the heuristics are applied. Whether AI agents can test at all turns out to depend largely on how testable the product is built, and that is the case this document makes for investing in testability now. It first shows where the new context changes practical testability, then traces how AI agents shift the dynamics of testability, then sets out where the request for testability has to come from, and finally reads his guidewords in that light.

AI agents as testers

Before going further, a position is needed on whether AI agents test at all. The question deserves care, because the answer rests on one of the most useful distinctions in the craft.

James Bach and Michael Bolton separate testing from checking. Checking is the mechanistic verification of propositions about a product, and a machine can perform it completely. Testing is evaluating a product by learning about it through experiencing, exploring and experimenting. It includes checking, draws on tacit knowledge, and in their account is a human activity that tools can support but not perform. They have also warned that large language models are built to produce a quick answer rather than to inquire, and that output which can be mistaken for expertise has to be judged by people who have it.

I hold the same distinction, and my own work with AI agents in testing confirms how much that distinction matters. Much of the confusion about AI in testing comes from treating checking, which machines can do well, as if it were testing. Any claim that AI agents test has to survive that distinction.

What an AI agent does sits between the two. It goes beyond checking. An AI agent can form a hypothesis about where a product is broken, try inputs nobody specified, and change its approach when a result surprises it. It can perform tasks, run experiments, observe what happens and

analyse the results, and from that draw inferences and tentative conclusions. None of this is the application of a fixed decision rule to a known observation, so calling it checking understates it.

It also falls short of testing in the full sense, in ways that matter:

Tacit knowledge	it has what was written down for it and what its training approximates. It does not know what it does not know about this product, this team and these users, and it rarely says so.
Learning	what it learns in a session is lost when the session ends, unless something was built to keep it.
Significance	it can find that something differs from an expectation, but its sense of whether the difference matters, and to whom, is borrowed from whoever wrote its instructions.
Inquiry	its default pull is towards producing an answer. Left to its defaults, it tends to complete the task where a tester would stop and ask.
Responsibility	it cannot be held responsible for what it reports.

This document therefore makes a narrower claim. AI agents perform parts of the testing activity. What they produce is an approximation of the testing done by the practitioner whose judgement was encoded into them: possibly useful, never equivalent, and as good as that encoding. How close the approximation comes depends on conditions, and most of those conditions are testability conditions:

Oracles are explicit and reliable. Where a problem can be recognised from something the AI agent can read or run, its inferences rest on something that can be checked. Where recognising the problem needs judgement nobody wrote down, the approximation weakens.

The product can be observed and controlled through its tools. What an AI agent cannot see or reach, it can only infer.

The context is explicit. Mission, risk and users are written down where the AI agent can load them.

The encoding is sound. The judgement of a skilled practitioner is in its skills and definitions, together with permission to stop.

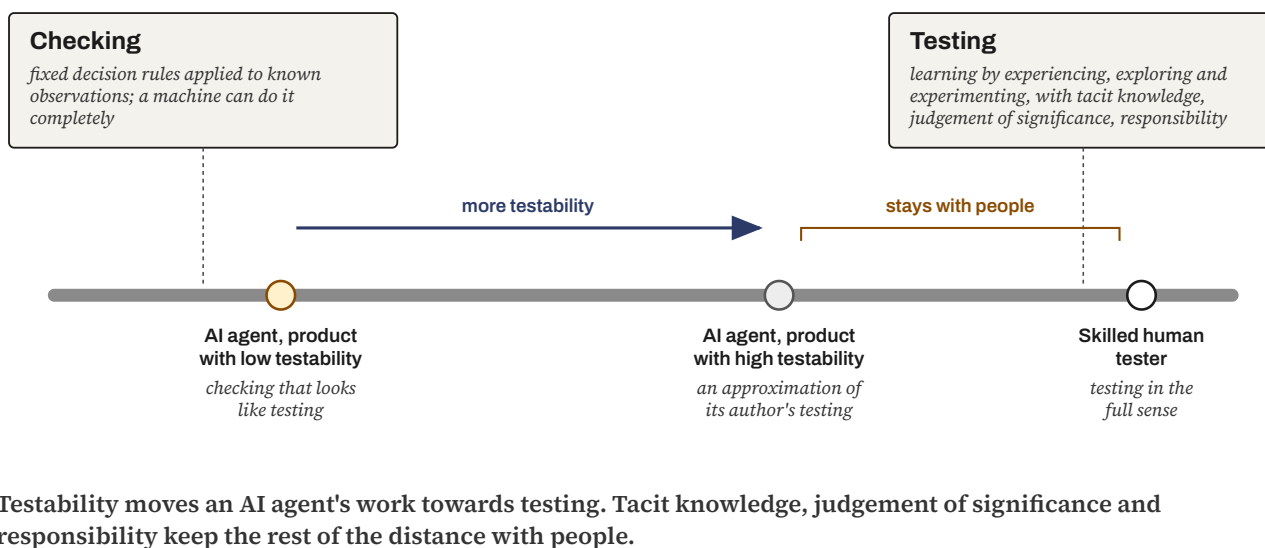
The approximation is measured. Seeded defects it should find, the same assessment run more than once, and comparison with a skilled tester's findings show how close it comes.

Where these conditions hold, an AI agent's testing can come close to that of the person who encoded it, for the part of the work that person was able to encode. Where they do not, its output is closer to checking that looks like testing, and it should be read that way.

This turns the usual question around. Whether an AI agent can test is usually asked about the AI agent: its model, its prompts, its tools. The conditions above place much of the answer in the product and the project. The same AI agent, with the same encoding, comes closer to testing on a product whose states it can observe, whose oracles it can run and whose context is written down,

and falls back towards checking on a product that offers none of these. Testability therefore decides more than how easily a product can be tested. It decides whether an AI agent's work on that product can count as testing at all.

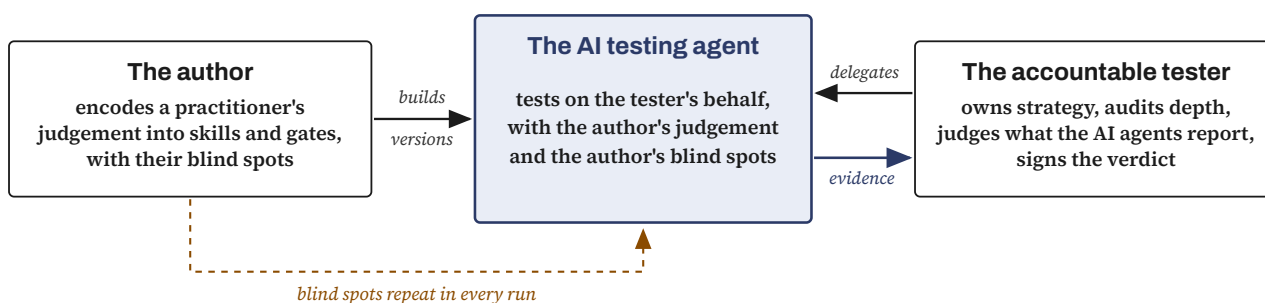
Figure 4. Where an AI agent's work sits between checking and testing



The consequence is an investment decision. An organisation that wants AI agents to test, and to do more than check, has to invest in the testability of what it builds, and has to design that testability with AI agents among its testers. Testability for agentic development is the precondition for agentic testing. Without it, the most capable AI agent available produces checking that looks like testing, faster and in greater volume than before.

The tester's role is different. Being responsible and answerable for the testing, owning its strategy, auditing its depth and judging what the AI agents report, cannot be handed to a model, because nobody can hold a model responsible the way they hold a person responsible. Deciding, case by case, whether an AI agent's output counts as testing or only as checking is part of that role. The activity can be shared. The role stays with a person. The section on addressing testability returns to how the judgement inside an AI agent is built.

Figure 3. An AI testing agent sits between the person who built it and the person who answers for its results



Its testing skill comes from the first person. Accountability for what it reports stays with the second.

Practical testability

The practical testability of a product is how easily it can be tested* by a particular tester and test process, in a given context†. Both the tester and the test process have moved. The tester now takes several forms: a human, a human assisted by AI agents, or AI agents orchestrated by a human. The

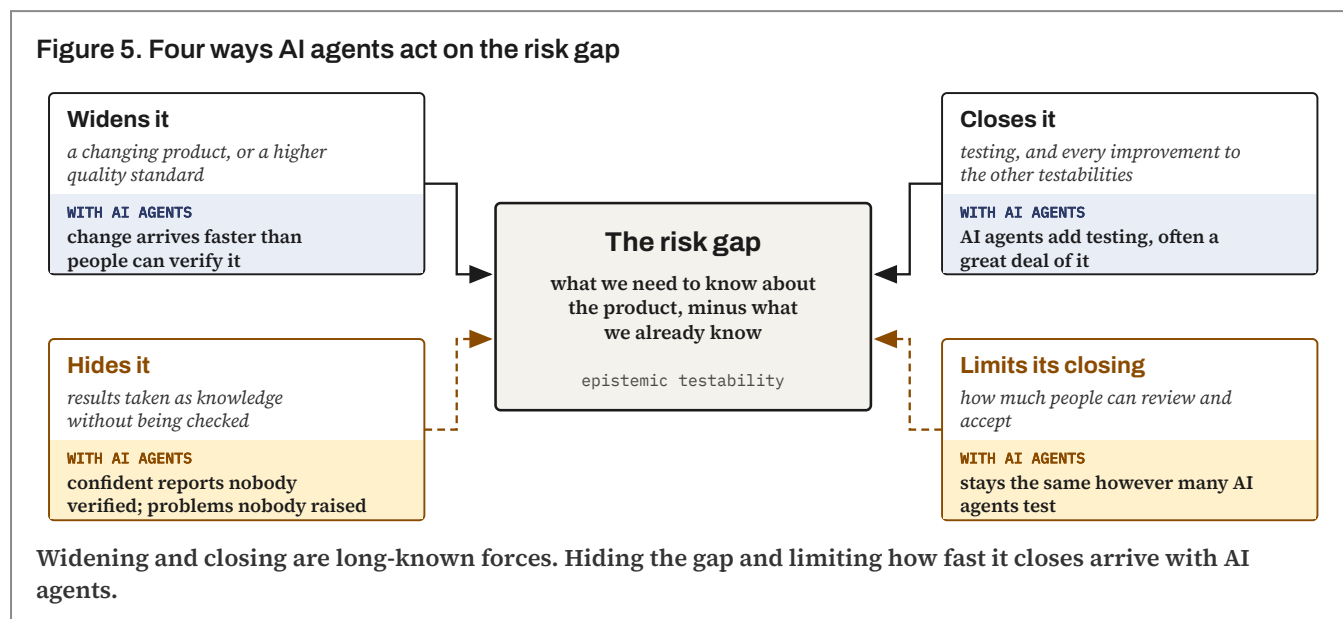
test process now runs partly inside AI agents. What separates the forms is how much of the testing passes through an AI agent, so the properties that matter are those in which an AI agent differs from a human tester:

Property	Human tester	AI agent
What it can observe	Anything their senses and tools reach	Only what its tools hand it: text, structured data, screenshots and, with some tools, audio and video
What an observation costs	Time and attention	Tokens and time, on every look
What it knows about this product	What it has learned, including what was never written down	What is in its context or can be retrieved into it; its general knowledge comes from training and may be out of date
What it does when blocked	Asks someone, or reports that the product is hard to test	Depends on how it was built and instructed; left to its defaults, it tends to work around the obstacle and report a result
How consistent it is between runs	Varies with fatigue, attention and what the tester has learned since	Varies with sampling, prompt and model changes, and memory; the variance can be measured by re-running
Who answers for its result	The tester	Should be a person who reviews and signs the result; unless someone is named, nobody does

Practical testability is a function of five other testabilities. Each of them shifts when testers with these properties take part. Each example is invented.

Testability	What it comes from	What changes when AI agents take part	Example
Epistemic	<i>The gap between what we know about the product and what we need to know, also called the risk gap</i>	The gap can widen faster than before, and it can look closed while it is still open	"We merge sixty AI agent pull requests a day. Nobody can say what is in them."
Project-related	<i>The physical, social and informational conditions of the project</i>	An AI agent's harness, permissions and environments become its working conditions, and it pushes back only through channels the harness provides	"Each AI agent gets its own sandbox, so destructive tests put nobody else's work at risk."
Value-related	<i>The quality standard and how well we know it</i>	A standard or an oracle guides an AI agent only if the AI agent can read it or run it	"The acceptance criteria are executable, so the AI agent can check them."
Subjective	<i>The tester and the test process</i>	Part of the tester's skill is now designed by someone else, and its consistency varies from run to run	"Since the knowledge graph went in, the AI testing agent no longer starts from zero."
Intrinsic	<i>The product itself</i>	What the product does not expose to an AI agent's tools is out of its reach, and what it exposes only on screen is read at the highest price	"Structured logs with correlation IDs let the AI agent read state directly."

The first of these is where the others meet. Epistemic testability, the risk gap, is the reason for testing in the first place, and the other four decide how fast it closes. AI agents act on the gap in four ways. Two are long known; the other two arrive with AI agents:



Testability is plastic and multi-dimensional, and no single metric expresses it. With AI agents in the team this matters more: a healthy pass rate and falling test depth can drift apart, and a single score is where that drift would hide.

Testability dynamics when AI agents take part

Testability is dynamic. The testabilities act on one another: some forces widen the risk gap, others close it, and improving one testability can cost another. AI agents change several of these relations and add new ones.

AI agents widen the risk gap faster than people can close it.

Every change to a product, and every rise in the quality standard, adds to what we need to know. When AI agents build the product, change arrives faster than people can verify it, and the gap widens on every day the rate of change exceeds the rate of verification. Requirements nobody could test widen it further: an AI agent handed one invents an oracle downstream, and no signal returns to the author.

Human review sets the pace at which the gap closes.

Any improvement to testability speeds up the closing of the gap. With AI agents, the gap closes only as fast as their results become knowledge, and a result becomes knowledge when a person can trace it, sample it and trust it. Review capacity therefore caps the rate, however many AI agents test, and making their output reviewable is among the strongest improvements available.

An AI agent can make testing seem easier while it gets worse.

A tool that makes testing feel easier is not necessarily making it better. An AI agent left to its defaults is the strongest case yet: thousands of passing checks, confident reports, no complaints about testability. A rising pass count and falling defect detection are compatible, and the dashboard shows only the pass count.

An AI agent that tests well costs more to build and to run.

A better test strategy usually asks for more effort and skill than the one it replaces. The same holds for AI agents: one built to recognise testability problems, stop on them and log what it worked around costs more than one that simply completes the task.

Making a product testable for AI agents has costs of its own.

Redesigning a product to make it more testable can introduce bugs and delay independent testing, and AI agents make such redesign cheap to attempt, so its bugs arrive sooner and in greater number. Interfaces added so that AI agents can test cheaply, such as APIs and command lines, also drift from the interface users use. Testing follows the cheap path, and the user interface accumulates untested behaviour.

Realistic testing now passes through a model.

Real users, data and environments bring realistic conditions at a cost in time, money and the risk of mishandling personal data. With AI agents, whether data containing personal information may be processed by the model in use, hosted or local, becomes a project constraint: the more realistic the data, the fewer AI agents may touch it. The trade runs the other way too. Isolation that makes destructive testing by AI agents safe moves the test environment away from production.

Testability built for AI agents serves everyone.

A product that is easier to test is easier to support, debug and evolve. The structured interface that lets an AI testing agent read state also serves the AI coding agent, the support engineer and the human tester debugging at 2 a.m.

The request for testability has to be designed in.

Asking for testability has always been the tester's job, because nobody else can be counted on to do it. A human tester asks. An AI agent asks only if asking was built into it: a way to recognise the problem, a channel to raise it and a reason to stop. Where it was not, the request does not arrive, and nothing announces its absence. What can still be built in is a trace: an AI agent that logs what it could not observe, could not reset, inferred and retried leaves a record that stands in for the complaint. A case from the author's own platform work shows what an untraced workaround costs, in a parser rather than an AI agent: it matched the real section of a document, discarded the match and returned placeholder values, and was found only because someone read the code.

An AI testing agent has an author.

Its testing skill is encoded by the people who wrote its skills and definitions, together with their blind spots, which then repeat in every run. An AI testing agent is therefore a product in its own right, to be built, reviewed and versioned like one, and its testability is part of the team's testability.

Richer observability raises the cost of each observation.

More logs help an AI agent see, and every line costs tokens to read. A product whose common questions have cheap answers keeps both in balance.

Persistent memory raises subjective testability and lowers repeatability.

An AI agent that remembers earlier sessions tests with more product knowledge and reproduces its own results less reliably. Deciding which events may change its memory, and logging the variance between runs, keeps the trade visible.

Independence between maker and checker costs time and money.

Separating the AI agent that writes code from the AI agent that checks it raises the authority of the check and the cost of every change. The components a team cannot afford to fail are the place to pay it, and where a team chose not to is worth recording.

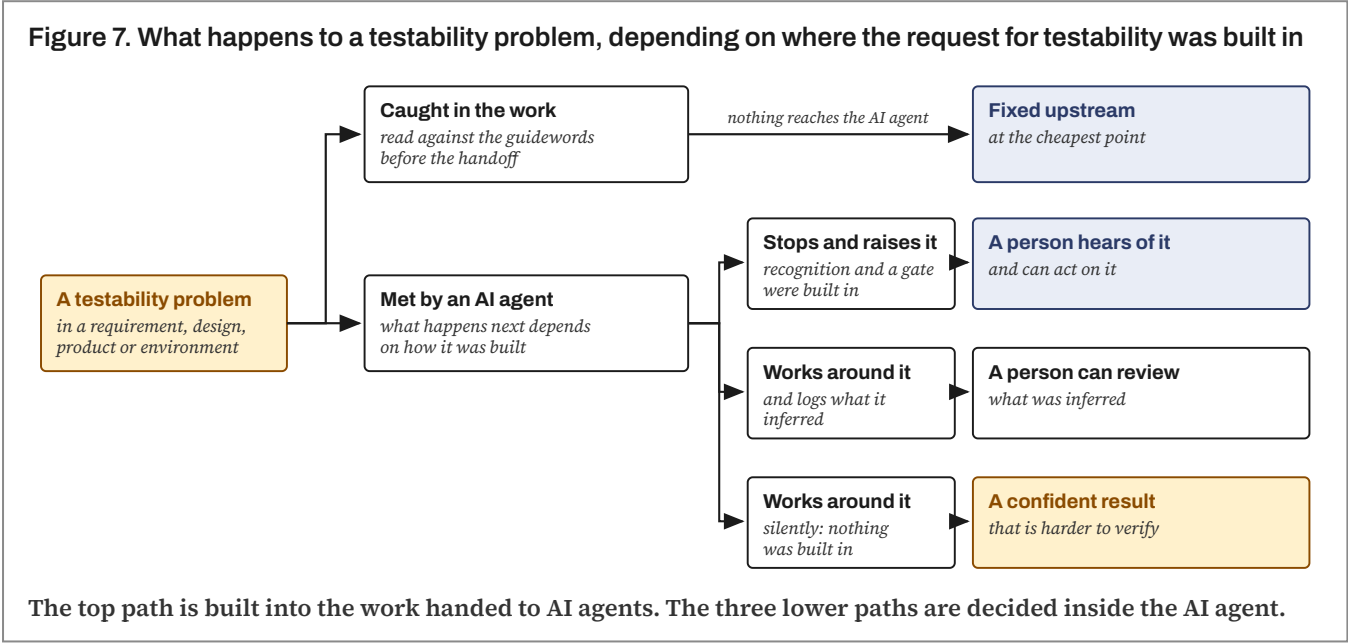
Figure 6. Each trade is paid in budget or in knowledge

WHAT YOU BUY		WHAT YOU PAY WITH	WHERE IT SHOWS
Richer observability	→	Observation cost (tokens per look)	<i>on the invoice</i>
Maker-checker independence	→	Speed and money	<i>on the invoice</i>
Persistent memory	→	Repeatability	<i>not on any invoice</i>
Tighter sandboxing	→	Realism	<i>not on any invoice</i>
Swarm throughput	→	Verified knowledge	<i>not on any invoice</i>
Interfaces built for AI agents	→	Untested user-facing reality	<i>not on any invoice</i>

Budget costs appear in cost reports. Knowledge costs appear nowhere unless someone measures them.

Addressing testability when AI agents take part

The dynamics come down to one question. Where does the request for testability come from when the tester who meets the problem is an AI agent, or a person who sees the product through one? It can come from two places: the work handed to the AI agent, and the AI agent itself. Both are investments in testability. The first makes the product testable enough for an AI agent's work on it to count as testing. The second makes the AI agent able to tell when its work does not.



In the work handed to AI agents

Testability is decided wherever work is prepared for a tester. When that tester may be an AI agent, each artefact needs to be testable in its own right, because the tester downstream may never ask for it to be made so:

Requirements	the author can name how each requirement will be checked, at the time of writing.
Design specifications	a build can be checked against the specified design tokens, states and behaviour.
Code and product	states can be observed and reached by command, and one behaviour can be understood without loading the whole system.
Environments and data	resettable by command, isolated per AI agent, and lawful to process with the model in use, hosted or local.
Test assets	the strategy is readable by AI agents, and oracles are reliable and precise.

In the AI agent itself: designed agency

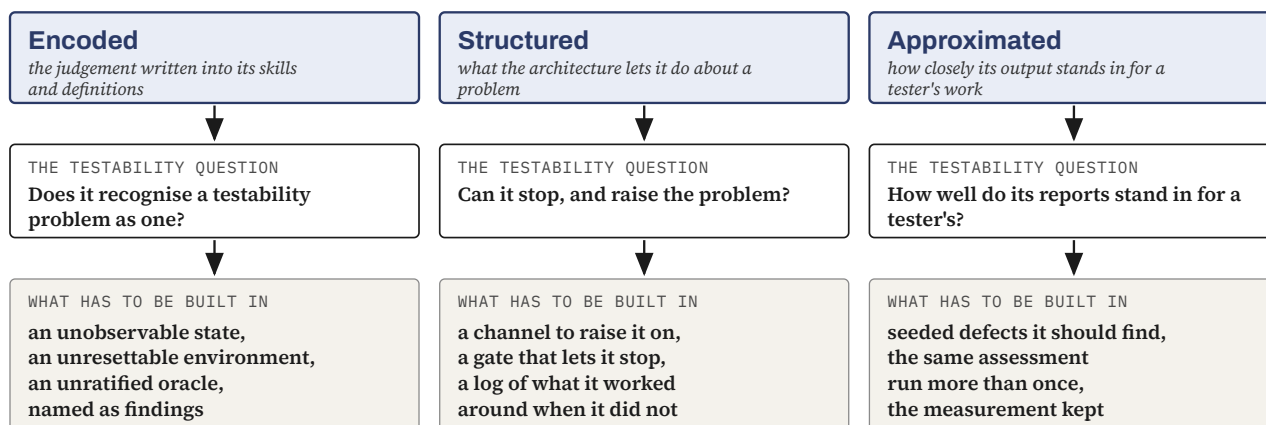
An AI agent's agency is designed. I have described this as *designed agency*†: its judgement is **encoded** by the practitioners who wrote its skills and definitions; what it can do about a problem is **structured** by the architecture it runs in; and what it produces is an **approximation** of a

practitioner's work, as good as that encoding and that structure allow. Each of the three decides something about testability.

Encoded	an AI agent recognises a testability problem only if that recognition was written into its skills and definitions. An unobservable state, an unresettable environment or an unratified oracle is a finding to name only if the AI agent was taught that it is one.
Structured	an AI agent raises a problem only if the architecture gives it a channel to raise it on, a gate that lets it stop, and a reason to stop rather than complete. Where it does not stop, a log of what it could not observe, could not reset or inferred is the trace a person can review.
Approximated	how closely an AI agent's reports stand in for a tester's judgement is a measurement, never an assumption: seeded defects it should find, and the same assessment run more than once.

The author's own platform work holds an example of each outcome. In a controlled run, an AI agent instructed to report every command it ran investigated a flaky check and refused to approve a merge; the stop was designed in. The counter-example is the parser described among the dynamics, a component that behaved the way an AI agent left to its defaults tends to. Nothing had been designed in to make it stop or say so. What separated the two outcomes was design.

Figure 8. Designed agency applied to testability



Each aspect of designed agency decides one thing about whether the request for testability arrives.

What this means

- For teams without AI agents: every guideword applies as it always has. Each problem a team fixes today because its human testers asked is one an AI agent will not later work around in silence.
- For teams adding AI agents: testability problems surface less often on their own. The requests a human tester would have made belong in requirements, specifications, skills and gates before the handoff. The testability of what they build decides how much of their AI agents' work counts as testing.
- For those who build AI testing agents: the request for testability is part of the product they build. An AI testing agent that cannot recognise a testability problem, raise it or stop on it passes the problem on as a result.
- For testers: the testing activity can be shared with AI agents. Strategy, the audit of depth and the judgement on what AI agents report stay with a person.

Principles that apply across the guidewords

Some consequences of AI agents taking part apply to many guidewords at once. They are stated here once, so that each guideword can say only what is specific to it.

What exists for an AI agent is what it can reach.

Information, tools, versions, environments, standards and knowledge of users serve an AI agent only when it can load, retrieve or call them. What lives only in people's heads, in meetings or behind a screen is absent for it, and it seldom says so.

Every observation has a price.

An AI agent pays in tokens and time each time it looks. A screenshot costs the most and says least precisely what state the product is in, and output a check prints is paid for again when it is read.

The request for testability has to be designed in.

An AI agent recognises, raises and stops on a testability problem only as far as its design allows, and its workarounds are visible only if they are logged.

Results have to be reviewable by a person.

Every output an AI agent hands to a person for acceptance is itself something to be tested, and its size and shape decide whether it can be.

Guidewords for analysing testability

The guidewords serve both places: reading the work before it is handed to an AI agent, and deciding what an AI agent has to be built to notice. Each guideword first states what any tester needs. "With AI agents" follows only where AI agents change what the guideword asks; where it does not follow, the guideword applies unchanged. Guidewords marked as additions are new in this document.

Epistemic Testability

Created by the gap between what we know about the product and what we need to know.

Prior Knowledge of Quality The more we already know about the product, the less testing remains to be done.

WITH AI AGENTS

An AI agent's report adds to that knowledge only when a person can trace it to the run, the build and the AI agent that produced it, and check a sample. An untraceable "all checks passed" is a claim. In one case, a test suite whose runner matched no test files at all exited green, having tested nothing.

Tolerance for Failure Where the consequences of failure are low, less testing may be needed to find failures.

WITH AI AGENTS

Tolerance also decides how much an AI agent may do unreviewed. Declared per component, it sets where a person signs off; one setting for the whole product gives the payment flow the same autonomy as the marketing page.

Change Control Frequent, disruptive change forces retesting and makes what we knew about the product obsolete. Controlling change usually helps, but it hurts when testers are not allowed to update their environments, tools and data.

WITH AI AGENTS

AI agents can change a product faster than people can verify the change; testing keeps up where, for the components that matter, the rate of change is held within the rate of verification. The caveat applies to them too: change control that stops an AI agent updating its own test environment and data stops it testing the change.

Human Reviewability

Addition, extending Smallness to the output a person must review

Work is easier to evaluate when it is shaped for a person to review: small, ordered by risk, evidenced and summarisable in minutes.

WITH AI AGENTS

This applies to every output an AI agent hands to a person: code, reports, summaries and verdicts. A four-thousand-line change with a one-line description cannot be reviewed claim by claim, however good the code.

Project-Related Testability

Created by the physical, social and informational conditions of the project.

Mission Alignment	Testers and the project agree, reasonably, on the service testing will provide and how it will be done. WITH AI AGENTS That agreement reaches an AI agent only through what it loads, such as the strategy, its skills and its gates. It picks up nothing in the corridor.
Prudence	A business that is responsible about releases takes product risk seriously and supports testing. WITH AI AGENTS A rule such as "an AI agent may merge when checks pass" is a release decision. Prudence means making it per component and on purpose.
Supportive Culture	Testers do their best work when they are respected, independent and not made to feel like troublemakers. WITH AI AGENTS An AI agent built to be helpful has the opposite difficulty. It has no fear of being a troublemaker; it was designed not to become one. Its equivalent of a supportive culture is explicit permission, in its instructions and gates, to bring bad news: to report that something cannot be tested, or to refuse to approve.
Developer Availability	Being able to talk to, and perhaps influence, developers, designers and managers makes for good testing. WITH AI AGENTS Influence is how a request for testability gets acted on. An AI agent has it only when it has a route to the people who can change the product, an expected response time and an instruction to ask before it guesses.
Information Availability	Testers get the information they need to test well. WITH AI AGENTS Design rationale that lives only in a meeting is unavailable until someone writes it down where it can be retrieved. Whether an AI agent read what does exist can itself be checked.
Tool Availability	Testers have the tools they need to test well. WITH AI AGENTS A tool is fully available when it can be called, with a documented interface and permission granted. A tool with only a graphical interface can be driven, but slowly, expensively and less reliably.
Test Item Availability	Testers can reach and use every relevant version of the product. WITH AI AGENTS A version an AI agent cannot deploy into its own environment tends to be tested late or not at all.
Sandboxing	Testers can do any testing worth doing, destructive testing included, without disrupting users, other testers or development. WITH AI AGENTS That takes an isolated environment per AI agent, no production credentials and a known blast radius, because a destructive shortcut taken by an AI agent happens at machine speed, possibly in several places at once.

Environmental Controllability

Testers can control every variable around the test that might matter.

WITH AI AGENTS

Clock, seed, network and data state need to be settable by command and restorable in one step, and AI agents working in parallel must not corrupt each other's environments.

Time

Too little time destroys testability; testers need time to think, prepare and cope with surprises.

WITH AI AGENTS

An AI agent's time is a budget of tokens, context, tool calls and feedback-loop latency, and it gets time to think only if the budget allows. A test suite spends that budget twice, once to run and once to read: a fifty-minute suite that prints thirty thousand lines is expensive both ways.

Leanness

Complexity accumulated in work products, and bureaucratic process, take time away from testing.

WITH AI AGENTS

An AI agent pays for clutter on every task, in context spent on noise, so technical debt acquires a price per task.

Action Gating & Reversibility

Addition

Testing is safer when irreversible actions are known and gated behind approval, and everything else can be undone.

WITH AI AGENTS

Sandboxing protects others from the AI agent; gating protects the work from it. Inside its own sandbox an AI agent can still drop a table it later depends on, or trigger an action whose effect leaves the sandbox, such as an email or a payment.

Value-Related Testability

Created by the quality standard and how well we know it.

Low Expectations

The lower the quality standard, the easier the mission of testing.

WITH AI AGENTS

An AI agent works to the standard it can read. A standard held only in people's heads guides the people and leaves the AI agent with whatever standard it infers.

Oracle Availability

We need a way to recognise each kind of problem; a specification is one oracle, and people and tools are others.

WITH AI AGENTS

A person is an oracle an AI agent rarely consults unless its workflow sends it there. "It should feel responsive" is an oracle a person applies with judgement; an AI agent needs it turned into something it can check, such as a latency limit for named interactions, and something is lost in the translation.

Requirement Testability

*Addition, a case of Oracle
Availability*

A requirement is testable when its author can say, while writing it, how it will be checked.

WITH AI AGENTS

An AI agent that receives an untestable requirement seldom goes back to the author unless its workflow provides the route. It invents an oracle at the most expensive point, and the next requirement arrives just as untestable.

Design-Spec Testability

*Addition, a case of Oracle
Availability*

A design is testable when the built product can be checked against it as specified: tokens, states and behaviour.

WITH AI AGENTS

Without that, an AI agent can check a build only against general heuristics or against earlier screenshots of the same build.

Oracle Authority

We benefit from oracles that find problems people will consider important.

WITH AI AGENTS

An AI agent that writes its own oracles decides what counts as important, and tends to decide by what it can check. A check written by the AI agent that wrote the code agrees with that code by construction. Authority comes from someone who knows what matters: a person who confirms the oracle, or an independently built AI agent applying criteria a person set.

Oracle Reliability

We benefit from oracles that can be trusted over time and across conditions.

WITH AI AGENTS

A flaky check taxes an AI agent either way. Left to its defaults, it may retry until the check passes and report success; built carefully, it spends many runs establishing that the check is unreliable. Both costs come out of the testing budget, so flakiness is cheaper measured and quarantined beforehand.

Oracle Precision

We benefit from oracles that point to specific problems.

WITH AI AGENTS

A failure message helps when it states what broke and where, in text the AI agent can act on. "AssertionError: false is not true" sends it, and every reader after it, searching.

Oracle Inexpensiveness

We benefit from oracles that cost little to obtain and run.

WITH AI AGENTS

The cost of a check is its seconds and tokens multiplied by the thousands of times it will run. A full browser session with a screenshot is the expensive path, best kept for what nothing cheaper can see.

User Stability & Unity

The less users change and the more they agree, the easier the testing.

WITH AI AGENTS

Other AI agents are increasingly among a product's users, and they change whenever their models change. A partner's AI integration agent is a user worth describing.

User Familiarity	The more we understand users and identify with them, the easier it is to test for them. WITH AI AGENTS An AI agent can be given personas, journeys and usage data, which serve understanding. It cannot identify with a user. That part of familiarity stays with the human tester, and it is one of the reasons the human tester stays in the loop.
User Availability	The more we can talk to users and watch them, the easier it is to test for them. WITH AI AGENTS An AI agent cannot sit with a user. It can put a question to a person who knows the users, if its workflow gives it someone to ask and the answer arrives within its task.
User Data Availability	Access to natural data makes testing easier. WITH AI AGENTS Natural data is usable only where it may lawfully be processed by the model in use, hosted or local; the trade between value-related and project-related testability applies with new force.
User Environment Availability	Access to natural usage environments improves testing. WITH AI AGENTS Browsers, devices and assistive technology are available to an AI agent when they can be started by command. A phone on a desk is out of reach until it is in a device lab.
User Environment Stability & Unity	The fewer user environments and platforms there are, and the less they change, the easier the testing.
Subjective Testability	
<i>Created by the tester and the test process.</i>	
Engagement	A bored or distressed tester is a poor one; a good tester is engaged intellectually and emotionally, and confident of finding any problem that exists. WITH AI AGENTS An AI agent is neither bored nor distressed, but its attention to instructions degrades over long contexts. Its confidence is of a different kind: an engaged tester is confident of finding problems, while an AI agent left to its defaults reports confidently whether or not it searched hard.
Involvement	Testing is easier when the tester is close to development and works well with the team; testers held away from development lose efficiency badly. WITH AI AGENTS An AI agent is involved when it is wired into delivery events, such as the requirement, the design decision and the change, with standing to comment on each. Kept away, it tests without that context and seldom mentions that it lacks it.

Product Knowledge	Knowing a lot about the product, including how it works inside, makes testing much easier; where that knowledge is missing, exploration builds it quickly. WITH AI AGENTS Exploration builds an AI agent's knowledge for one session only, unless what it learned is kept as loadable context, such as instruction files, a code knowledge graph or a record of past findings. Otherwise every session starts with a cold read of the repository.
Technical Knowledge	Programming ability, knowledge of the technology and tools, and an understanding of how software gets developed make testing easier. WITH AI AGENTS A model's technical knowledge is broad and ends at its training data. Where the versions in use are newer, supplied documentation is what stops it testing against interfaces that have since changed.
Domain Knowledge	The more we know about the users and their problems, the better we can test. WITH AI AGENTS An AI agent knows users' problems in general, from training, and this product's users only from what it is given. Without that, it tends to infer the business rules from variable names and carry on.
Testing Skill	General testing ability makes testing easier, notably experiment design, modelling, product element factoring, critical thinking and test framing. WITH AI AGENTS Each of these is present in an AI agent only as far as someone encoded it in its skills and definitions, together with that author's blind spots. Its skill is known only once it has been measured, for example against seeded defects.
Test Strategy	A well-designed strategy can greatly reduce the cost and effort of testing. WITH AI AGENTS An AI agent follows the strategy it can read; without one, it spends its effort where effort is easiest. The result can be two thousand tests on getters and none on the payment flow.
Supporting Tester Availability	A supporting tester helps test the product without being responsible for the testing; developers can be good ones. WITH AI AGENTS The supporting tester is the role closest to what AI agents can be: they help, and the responsibility stays with the tester. No AI agent should be the supporting tester of its own work.

Tester Repeatability

Addition

Knowing how much a tester's findings vary between runs on the same build makes the findings easier to use. Human testers vary too, and in exploratory testing the variation is deliberate.

WITH AI AGENTS

An AI agent's findings vary with sampling, prompt, model changes and memory, and the variance can be measured by re-running. In one measured case, two runs agreed on every verdict and shared six of sixteen findings. One run is a sample; several, combined, come closer to the problems that exist.

Intrinsic Testability

Created by the product itself.

Observability

We need to see the product; ideally every fact about its states and behaviour, and their history, is readily available.

WITH AI AGENTS

"readily" now carries a price. Facts exposed as structured, filtered data are cheap for an AI agent to see; facts available only on screen are seen at the highest cost and the lowest precision.

Controllability

We need to reach the product's behaviour: any input, any state or sequence of states, on demand. Non-deterministic behaviour detracts from testability.

WITH AI AGENTS

A state reachable through an API, command line or tool is controllable for an AI agent; one reachable only through a screen tends to be tested last. Non-determinism now has two sources, the product and the tester, and they compound. A setting the product accepts but does not apply is worse than no setting, because every test configured through it tests a fiction.

Algorithmic Simplicity

The more complex and sensitive the relation between inputs and outputs, the more there is to examine.

WITH AI AGENTS

An AI agent can examine far more, which tempts teams to accept complexity they would never ask a person to test. The output someone must review grows with it.

Algorithmic Transparency

If nobody knows how the product produces its output, as is typical of machine learning, much more of the input and output space must be sampled.

WITH AI AGENTS

The tester may now be an opaque model as well. Two opaque systems need more sampling than one, and the sampling has to be budgeted. Provenance recorded for each answer, sources and confidence in a trail nobody can silently rewrite, is the transparency an opaque product can still offer.

Algorithmic Stability

If the product can change without radically disturbing its logic, past results stay valid; this is a particular problem in machine learning.

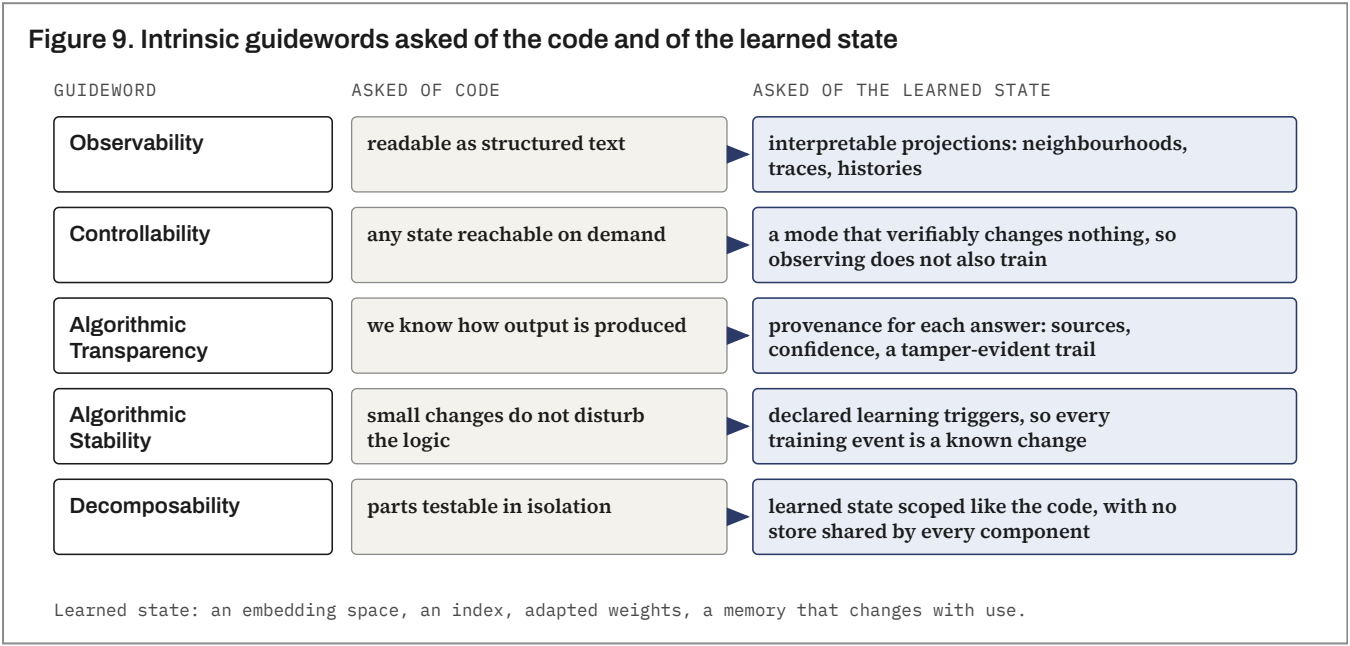
WITH AI AGENTS

A product that learns in use is stable only as far as its learning triggers are declared. An undeclared training event is an undeclared change.

Explainability	A product whose behaviour can be explained to outsiders is easier to test. WITH AI AGENTS An explained design lets a reviewer check an AI agent's reasoning against the intended design, as well as its output.
Unbugginess	Bugs slow testing, because they must be reported, worked around or waited on. WITH AI AGENTS The three responses become a design decision. An AI agent that stops and reports keeps the finding; one that works around the bug without saying so loses it; one that edits the test until it passes hides it.
Smallness	Less product means less to examine and fewer interactions to go wrong, and the same holds for the output we must review. WITH AI AGENTS The more an AI agent must load to change or test one behaviour, the more it pays and the more its attention degrades. The point about output now carries the most weight; see Human Reviewability.
Decomposability	Parts that can be separated make it easier to focus testing, investigate bugs and retest after change. WITH AI AGENTS Parts are decomposable for AI agents when several can test in parallel without colliding on ports, files, branches or databases.
Similarity (to known and trusted technology)	The more the product resembles what we already know and trust, the easier it is to test. WITH AI AGENTS A stack the model knows well is easier for it to test, with a catch: a stack that looks familiar invites confident use of an interface from a similar framework that this one does not have. Documentation of what differs closes both gaps.
Untrusted Content Exposure <i>Addition</i>	To a human tester, content the product displays is data to examine. WITH AI AGENTS Content can carry instructions. A product review that says "ignore previous instructions" can redirect the AI agent testing the product, so everything the product shows its tester is part of the attack surface. Where the AI agent has persistent memory, one stored entry can deliver the attack again in every later session.

Products that learn

The intrinsic guidewords already cover machine learning: opaque algorithms need more sampling, and learning makes logic unstable. Two things are new when AI agents take part. Products now learn in use, while they are being tested, through embeddings, indexes, adapted weights or a memory that changes with use. And the AI agents doing the testing learn too. Learned state cannot be meaningfully read in its raw form, so several intrinsic guidewords have to be asked twice, once of the code and once of the learned state:



The same questions apply to an AI testing agent with persistent memory. Its memory is learned state, and these guidewords decide whether anyone can tell what it has learned and why its verdict changed between runs.

Conclusion and next steps

Testability has always depended on a tester who asks, and on not mistaking a tool that makes testing feel easier for one that makes it better. Both carry more weight now. The tester who asks has not gone away. What has changed is the setting the tester works in: software built through AI, testing assisted by AI agents, and in places testing delegated to AI agents that a human tester orchestrates.

In that setting, the request for testability can no longer be left to instinct. When a human tester meets a testability problem, it is still raised. When an AI agent meets it, it is raised only if someone designed the AI agent to raise it. Otherwise it becomes a confident result that is harder to verify.

There is a second reason to invest. Whether an AI agent tests or only checks depends less on the AI agent than on the product it meets. A product built with AI agents among its testers, observable and controllable through their tools, with oracles they can run and context they can load, is what turns an AI agent's work into an approximation of testing worth having.

Testability therefore has to be addressed earlier, in the work handed to AI agents, and deeper, in how the AI agents themselves are built. The guidewords serve both. The first steps are small enough to start this week:

Analyse one product. Take a product your team already tests and walk through the guidewords with the human testers and the AI agents who work on it. Note where each was blocked and what each did about it.

Check the handoff. Before work goes to an AI agent, read the requirement, the design specification and the environment against the guidewords. Fix what a human tester would have asked about.

Look inside your AI agents. For each AI agent that tests, find out whether it can recognise a testability problem, where it can raise one and when it is allowed to stop. Where it cannot stop, log its workarounds.

Measure the approximation. Seed defects your AI agents should find, run the same assessment more than once, and compare the results with a skilled tester's.

Name who answers. For every result an AI agent produces, name the person who reviews it and signs it.

These guidewords are heuristics. The cases behind them come from one platform, one harness and one model. Read Bach's original, apply these where your testers and AI agents work, and share what holds and what breaks. Evidence from other teams and other contexts is what will sharpen them.

Definitions and attribution

* Bach defines *testing* as "evaluating a product by learning about it through experiencing, exploring, and experimenting".

† Bach defines *context* as all of the factors of a situation that should be considered when solving a situated problem. For an AI agent, one factor dominates: whatever cannot be loaded or retrieved into its working context (repository, instructions, tools, memory) does not exist for it.

‡ *Designed agency* is set out in Lalitkumar Bhamare, "Congratulations on your Agent. But does it have agency?", **LinkedIn**, 31 March 2026, <https://www.linkedin.com/pulse/congratulations-your-agent-does-have-agency-lalitkumar-bhamare-67kxf/>. It describes judgement deliberately designed in by practitioners who knew the work deeply enough to encode, structure and measure it: encoded, structured and approximated agency.

§ James Bach and Michael Bolton, "Testing and Checking Refined", satisfice.com, 2013; Michael Bolton and James Bach, "What Are We Thinking in the Age of AI?", conference presentation, 2025.

Cases. The cases described (the test suite that tested nothing, the parser that returned placeholder values, the AI agent that refused a merge and the repeated assessment under Tester Repeatability) were observed in the author's own platform work in October 2026, with one harness and one model. They illustrate the guidewords and do not test them. Of the three testability questions that AI agents in delivery raise (the product's testability for AI agents, the testability of the environment an AI agent works in, and the testability of an AI agent's own output), this document treats the first in most depth, the second through Sandboxing and Environmental Controllability, and the third through Human Reviewability and Tester Repeatability.

The five testabilities, the underlying testability dynamics and every guideword not marked as an addition derive from James Bach's work, from Heuristics of Software Testability, v2.9, Satisfice, Inc., 2025. They are paraphrased here, with short quotations marked; read the original at satisfice.com. The additions are Human Reviewability (extending his note on output under Smallness), Action Gating & Reversibility, Requirement Testability and Design-Spec Testability (cases of his Oracle Availability), Tester Repeatability and Untrusted Content Exposure. The additions, the reading of the dynamics and guidewords for AI agents, the new dynamics, the principles, the risk-gap model, the cases described and the designed-agency lens (encoded, structured, approximated; see ‡) are Lalitkumar Bhamare's own work.