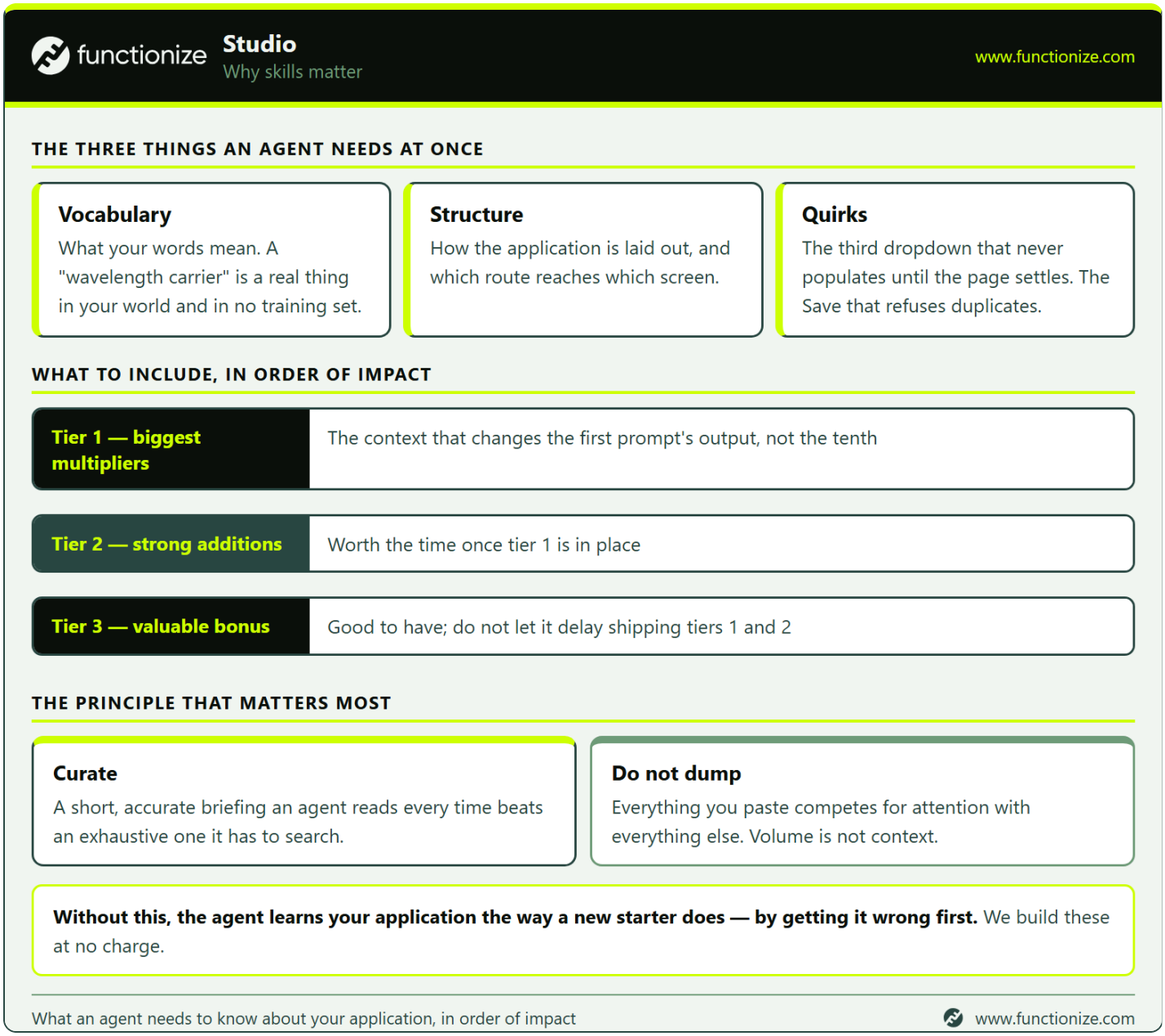
Why Skills Matter — Giving Your AI Agent the Context It Needs

Skills · https://docs.functionize.com/hc/en-us/articles/43753708136599-Why-Skills-Matter-Giving-Your-AI-Agent-the-Context-It-Needs

An AI agent pointed at your application knows everything about software in general and nothing about *yours*. It does not know that a "wavelength carrier" is a real thing in your world, that your Save button quietly refuses duplicates, or that the third dropdown never populates until the page has settled. It finds all of that out the way a new starter would — by getting it wrong first.



The three things an agent needs to know at once

Good test generation depends on the agent holding three kinds of knowledge simultaneously:

Functionize	How the platform works	We supply this — as a skill through our Claude Code plugin, and as built-in guidance from the MCP server in every other tool
How to prompt it well	What makes a durable, low-maintenance test rather than a brittle one	We supply this — same two routes
Your application	How it behaves, what its words mean, and what "success" actually looks like	Only you have this

The first two we bring. The third is the gap, and closing it is what a skill does.

The one principle that matters most

Everything in a skill exists to help the agent understand **how your application thinks** — its data, its timing, the real names of things, and what a successful outcome looks like. It is not there to record click-by-click steps or page selectors. The platform works out the mechanics on its own; the skill's job is to express what the user is trying to accomplish.

The richer the context, the smarter the test — never the longer the clickstream.

This is worth sitting with, because it is the opposite of how most teams document automation. A hundred pages of recorded steps will produce worse tests than two or three pages explaining what the application is for.

What to include, in order of impact

Tier 1 — the biggest multipliers

Bug tickets and defect history	Often the single most valuable source you have. Closed bugs make excellent regression tests, recurring bugs reveal your most fragile areas, and reproduction steps are ready-made test flows.
Manual test cases and QA plans	Direct raw material for coverage. Your "expected results" become the test's verifications.
Specs, PRDs and acceptance criteria	These define intended behavior and what "done" means — exactly what is needed to write meaningful success checks rather than tests that only confirm a page loaded.

Source code	Read for four things in particular: the data model (what a test creates versus only reads), how login and SSO work, where the interface loads or refreshes asynchronously, and the label and text files that hold the real names of screens and fields.

Tier 2 — strong additions

API spec and database schema	How to set up and verify test data, and how records get cleaned up afterward.
Roles, permissions and personas	Enables role-specific flows, and the negative tests where a user *should not* be able to do something.
Existing automated tests	Any framework. Flows worth migrating, and assertions already proven in practice.
Production usage and analytics	So coverage prioritizes what users actually do most, rather than what is easiest to automate.
Release notes and cadence	What changes often, and therefore needs regression — and when to run it.

Tier 3 — valuable bonus context

Support tickets and customer complaints	Real-world failure modes, often flows QA never thought to test.
Architecture and flow diagrams, design systems, Figma	See below — diagrams do something no amount of prose does.
Wikis, runbooks and onboarding docs	The tribal knowledge that is hard to find anywhere else.
Feature flags, SLA targets, demo scripts	Conditional flows, and your golden-path journeys.

Why diagrams earn their place

Teams often leave diagrams out because they feel like presentation material rather than documentation. They are some of the most efficient context you can provide.

An architecture diagram tells the agent which systems are involved in a flow, so it understands that creating an order touches three services and that the confirmation may arrive asynchronously. A flow or state diagram shows which transitions are legal, so it can write tests for the ones that should be blocked as well as the ones that should work. An entity or data-model diagram explains what depends on what, which is exactly what determines whether a test can be run twice without resetting anything.

A single diagram frequently conveys structure that would take several pages of prose, and it carries the relationships rather than just the names.

Give diagrams as text. There is no evidence that agents read images embedded in a skill, so a screenshot of an architecture diagram is wasted. Describe the diagram in words, or better, express it as Mermaid — which is plain text, reads cleanly, and renders as a diagram wherever you need one for people. If all you have is an image, send it to us and we will transcribe it for you.

Curate, don't dump

A focused skill beats an exhaustive one. A handful of well-explained example flows plus the context above will outperform hundreds of pasted test cases. Give the skill enough to understand your application and it will generate the rest.

What you do not need to provide

Click-by-click instructions. The agent works those out for itself. Supplying them tends to make tests more brittle, not less.

Real passwords, keys or secrets. Credentials are stored securely inside the project and referenced safely — never written into a test, and never needed in a skill. Use placeholders. The same applies to production data: describe its shape, not actual customer records.

The power move: connect the sources live

Static exports go stale the moment your application changes. Where you can, give your AI tool a live connection to the source — your code repository, issue tracker, knowledge base and API — through *their own* MCP servers, alongside the Functionize one.

To be precise about what does what: **a skill is a document that teaches the agent about your application. It does not generate tests, and it does not connect to anything.** The agent generates the tests, reading the skill for domain knowledge and reaching live sources through whatever MCP servers you have connected. The benefit of connecting those sources is that the agent sees your application as it is today, rather than as it was on the day someone exported a document.

We build these for you, at no charge

For commercial applications — SAP, Oracle, Salesforce, ServiceNow, Workday and other widely used platforms — we provide complimentary skill packages, so your team starts with the domain knowledge already in place rather than building it up from scratch.

For bespoke and internal applications — the systems nobody outside your company has ever seen — we build the skill with you. That is the more common case, and it is where a skill makes the biggest difference, precisely because no general-purpose model has ever encountered your application before.

Either way there is no charge. Most teams can assemble what we need in a single sitting, because it is knowledge they already hold; our part is turning it into something an agent can act on.

Your gather-it checklist

Tier 1

- Bug tickets and defect history
- Manual test cases and QA plans
- Specs, PRDs and acceptance criteria
- Source code — data model, authentication, async behavior, labels

Tier 2

- API spec and database schema
- Roles, permissions and personas
- Existing automated tests
- Production usage and analytics
- Release notes and cadence

Tier 3

- Support tickets and complaints
- Architecture and flow diagrams, design system, Figma
- Wikis, runbooks and onboarding docs
- Feature flags, SLA targets, demo scripts

You do not need all of it. Start with Tier 1 and the diagrams you already have.

Request a skill for your system under test

Two ways to start, whichever suits you:

- **Email matt@functionize.com** with the application you want covered.
- **Raise a support ticket** asking for a skill-building session for your system under test.

We are also happy to walk through any of this on a call and help you assemble the material — it is usually quicker than doing it alone, and we know which parts matter most.

If you have not connected your AI tool to Functionize yet, start with Getting Started with the Functionize MCP Server.

A skill is installed in your AI tool, beside the MCP connection — not through it. The two are independent: the MCP connection gives the agent the ability to act on Functionize, and the skill gives it the knowledge of your application to act sensibly. You install the skill wherever that tool keeps its skills, and our own skills ship in the Claude Code plugin, which also sets up the connection.