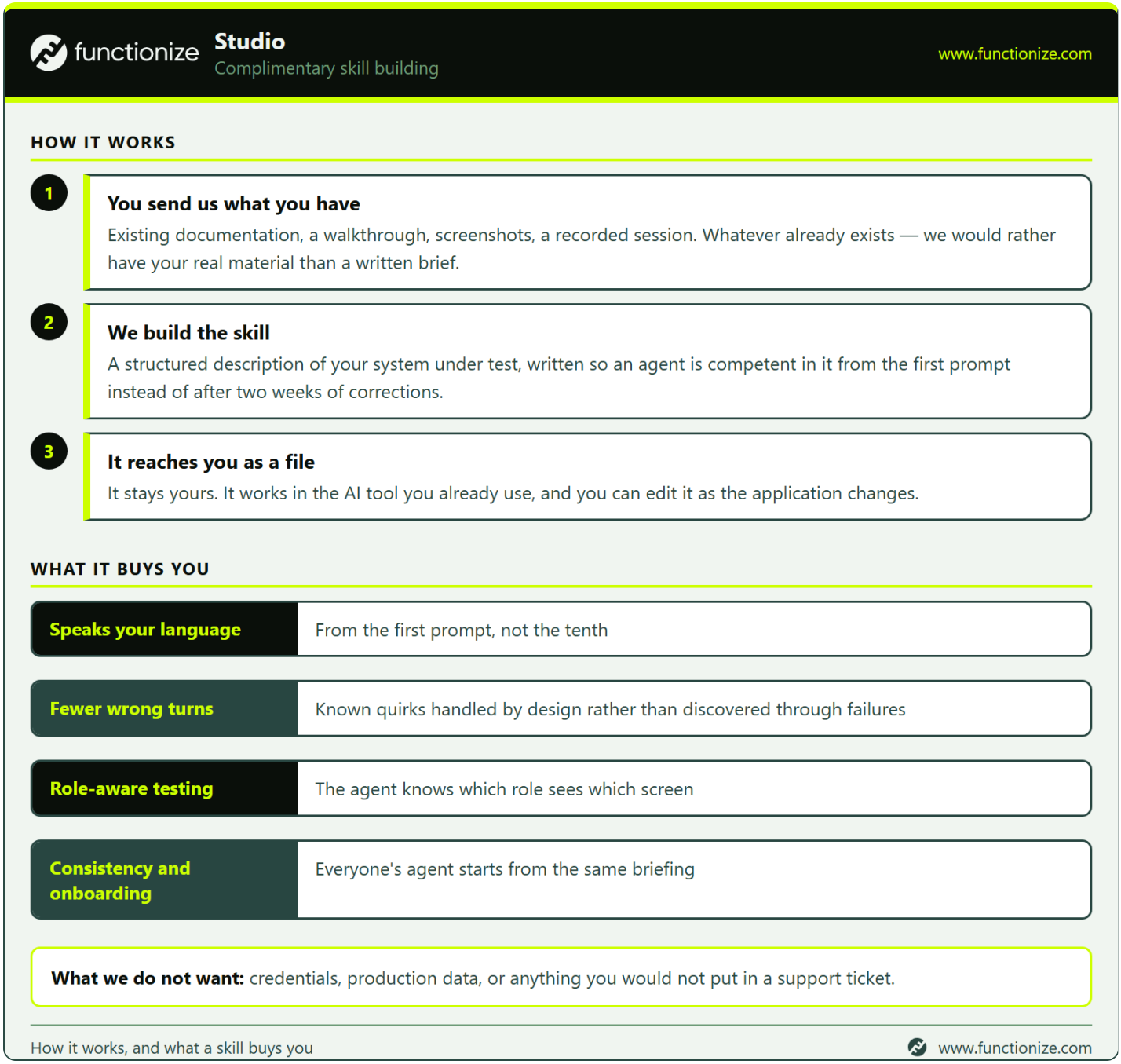
Complimentary Skill Building for Your System Under Test

Skills · https://docs.functionize.com/hc/en-us/articles/43753391837847-Complimentary-Skill-Building-for-Your-System-Under-Test

An AI agent that knows nothing about your application has to discover it — one wrong turn at a time. A skill fixes that. It is a package of context about your specific system under test that makes the agent competent in your application from the very first prompt, instead of after two weeks of corrections.



What a skill actually is

A skill is a structured description of your application, written so an AI agent can use it. Not marketing copy and not a user manual — the working knowledge a new member of your QE team would need in their first week, in a form the agent can act on.

It covers the things that are obvious to your team and invisible to everyone else: what your domain terms mean, which user roles can see what, how someone actually logs in, the flows that matter, what test data has to exist first, and the handful of quirks that make your application behave unlike any other.

Once built, the skill is **installed in the AI tool each person uses**, in whatever place that tool keeps its skills. It sits beside your Functionize MCP connection rather than traveling through it: the MCP connection lets the agent act on

Functionize, and the skill gives it the knowledge of your application to act sensibly.

Why it is worth doing

The agent speaks your language immediately

Every enterprise application has its own vocabulary. If your team says "benchmark", "safeguard", "wavelength carrier" or "suppression", the agent needs to know what those mean in your system — not what they mean generally. With a skill, your first prompt can use your own words and be understood.

Fewer wrong turns

Without context, an agent explores. It opens the wrong screen, follows a path that looks plausible but is not how your team works, and needs correcting. With the core flows written down — how a user is created, how an assessment is imported, how an order is released — it goes more or less where you would have gone.

Known quirks are handled by design, not discovered through failures

This is the part customers underestimate. Every application has behaviors that will defeat an automated test until somebody accounts for them: a dropdown that only shows records created before the current session, options that render in a different order each time, screens that need a moment before they are interactive, a login that occasionally returns a stale token and needs a retry.

Written into a skill, each of those becomes a rule the agent follows. Left out, each one becomes a failed test, an investigation, and a support ticket.

Role-aware testing

If your application behaves differently for a system administrator, an organization administrator, a standard user and a read-only user, the agent should know that before it writes a test — not discover it when a step fails for a permissions reason nobody flagged.

Consistency across the team

Without a skill, every tester teaches the agent their own version of your application, and the quality of the output depends on who was prompting. A skill is shared. Everyone's agent starts from the same knowledge, which makes the results consistent and the tests reviewable.

Faster onboarding

The same document that makes the agent effective makes a new human tester effective. Several customers have found the skill-building exercise valuable on its own, simply because it forces knowledge out of people's heads and into a form the whole team can use.

It works in the tool you already use

A skill is not tied to a particular product. It works in any tool that supports skills — Claude Code, Claude, Cursor, GitHub Copilot, Gemini CLI and Codex among them — and it is installed per tool, by each person or by whatever central provisioning your organization uses. Your team keeps working where it already works.

It stays yours, and it stays current

As your application changes, the skill is updated. Because it is a single shared artifact, one edit corrects everyone who is using it — rather than correcting each tester's habits individually.

That only holds if the skill is distributed centrally. If each person has copied the file into their own tool, an update reaches them only when they copy it again. Put it somewhere shared from the start: your organization's tool provisioning, an internal plugin, or a repository everyone pulls from.

What we need from you

Roughly two to three pages of information. Most teams can produce it in a single sitting, because it is all knowledge they already have.

Business context	What the system does, who uses it, and the terms your team uses for things.
User roles	The permission levels, and what changes between them.
Environments	Which environment tests should run against — and, just as importantly, which they must never touch.
Authentication	How users sign in: SSO, local credentials, multi-factor, and how test accounts are set up.
Core flows	The handful of journeys that matter most, described as **goals and business steps** — "a guest orders an item and receives a confirmation number" — not click-by-click instructions. The agent works out the mechanics.
Test setup and data	What must already exist before a test runs, and any preconditions worth naming.
Known quirks	The behaviors that catch people out. Be blunt here — this section pays for itself.
What to leave out	Anything not relevant, and anything you would rather we did not model.

What we do not want

Never send real credentials, license keys, or production secrets. Use placeholders. A skill describes how authentication works; it does not need a working password to do that. If a document you are about to send contains a real secret, remove it first.

The same applies to production data. Describe the shape of the data, not actual customer records.

How to request one

Contact your Functionize account team, or raise a ticket asking for a skill-building session for your system under test. We will send you the outline above, review what you send back, and build the skill with you.

How the finished skill reaches you

A skill built for your application is yours and is delivered to you directly — as a file, to install in your tools. It is not published in our public Claude Code plugin, which carries only our general platform and prompting skills.

From there, put it somewhere your team pulls from centrally rather than emailing copies around, so that an update reaches everyone.

If you have not connected your AI tool to Functionize yet, start with Getting Started with the Functionize MCP Server. The connection and the skill are separate things, and you want both.

For the deeper guide on what context to gather and why — including which sources have the biggest impact — see Why Skills Matter.