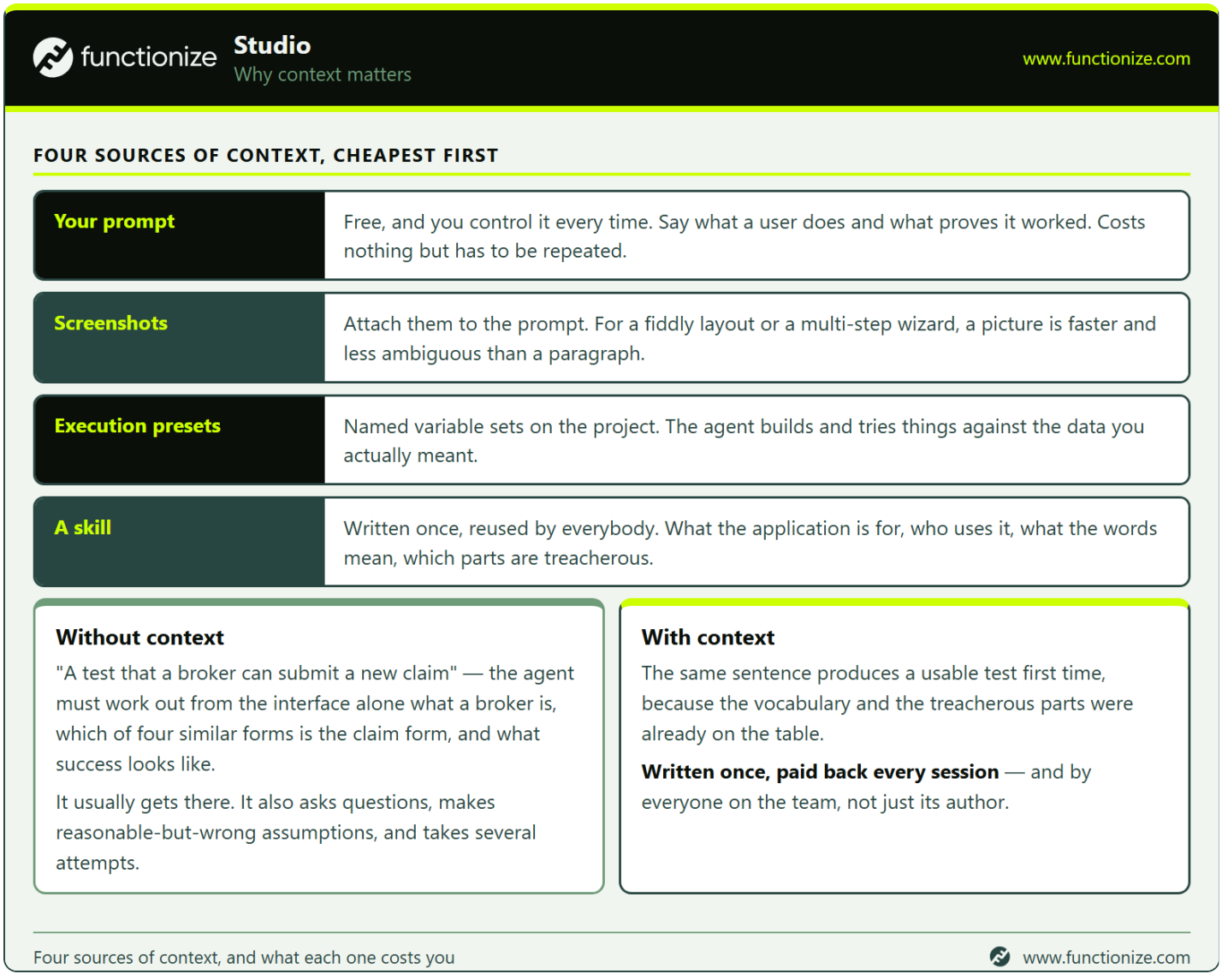
Why Context Matters, and How to Give Studio More of It

Skills · https://docs.functionize.com/hc/en-us/articles/43769475776023-Why-Context-Matters-and-How-to-Give-Studio-More-of-It

The fastest way to get good tests out of Functionize is not a better prompt. It is giving the agent the context a new colleague would need — what your application is for, who uses it, what the words mean, and which parts are quietly treacherous.



What each source of context costs you, and what it buys.

The problem context solves

Ask for "*a test that a broker can submit a new claim*" against an application the agent has never seen, and it has to work out from the interface alone: what a broker is, which of four similar-looking forms is the claim form, what a valid claim needs, and what success looks like.

It will usually get there. It will also ask you questions, make reasonable-but-wrong assumptions, and take several attempts. Multiply that across a team and it is the difference between automation that accelerates and automation that feels like hard work.

Give it the context up front and the same sentence produces a usable test first time.

Where context comes from

1. Your prompt

The cheapest context, and the one you control every time. A prompt that names a starting URL, describes what a user does rather than how the page is built, and states what "passed" means will outperform a vague one every time — see Exercise 2 — Write Assertions That Prove Something.

But prompt context dies with the prompt. The next person starts from nothing.

2. Project and test settings

Environment URLs, authentication, cookies and headers are context too — they tell the agent how to reach your application at all. Set them once at project level and every test inherits them. See Settings: Where Everything Lives and What to Change First.

3. A skill — the durable one

A **skill** is a short document describing your application that Claude reads automatically whenever it works on it. Roles, key screens, business rules, vocabulary, known-good test data, and the traps that produce flaky tests.

Unlike a prompt, a skill is written once and helps everyone, forever. It is the difference between one person knowing how the claims system works and the whole team's tooling knowing it.

The full case is in Why Skills Matter.

Build one yourself

Writing a skill is not an engineering task — if you can explain your application to a new starter, you can write one. It is a folder with a single text file in it.

We have a step-by-step guide with a complete, copyable example: How to Build a Skill in Claude for Your System Under Test.

It covers the file format, what to include and what to leave out, how to install it in Claude Desktop or Claude Code, and how to hand it to business users and colleagues so they can write tests in their own words.

Or we will build it for you, at no charge

We build skills for customers' systems under test as part of onboarding — SAP, Oracle, Salesforce, Workday, Guidewire, or your own in-house application. You give us a walkthrough and access; we write the skill and hand it over for your team to own and extend.

There is no cost and no commitment attached to it. Details, and how to request one, are in Complimentary Skill Building for Your System Under Test.

Connecting your own tools

Context also flows the other way. Connecting your AI tool to Functionize over **MCP** lets you create, run and diagnose tests from the editor where your code already lives, with your repository's context available to the same assistant.

Start with Getting Started with the Functionize MCP Server, or work through Exercise 7. Studio also has the configuration built in, under **Integrations** → **MCP** in the left sidebar.

Where to start

1. Get your project's environment and access settings right, so tests can reach the application.
2. Write half a page of skill covering the one workflow you test most. Use it for a week.
3. Add to it whenever you catch yourself explaining the same thing twice.
4. Ask us to build the full skill for your main system under test while you do.

The teams that get the most out of Functionize are not the ones with the cleverest prompts. They are the ones who wrote down what their application actually does.