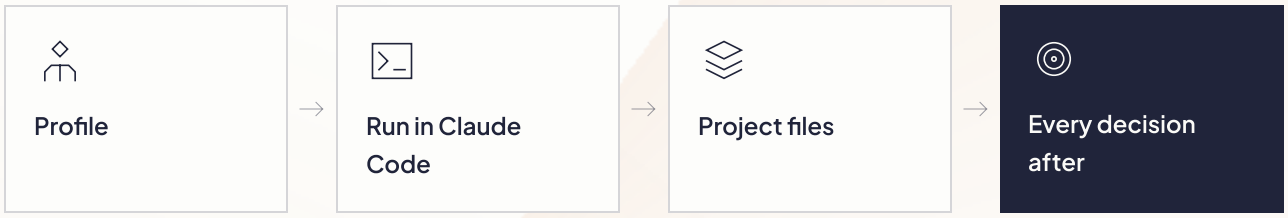
Synthetic users are not a test. They are *context.*

Running synthetic users in Claude Code, and what happens after the run: profiles, tasks and emotions that stay in the project and feed every product decision.





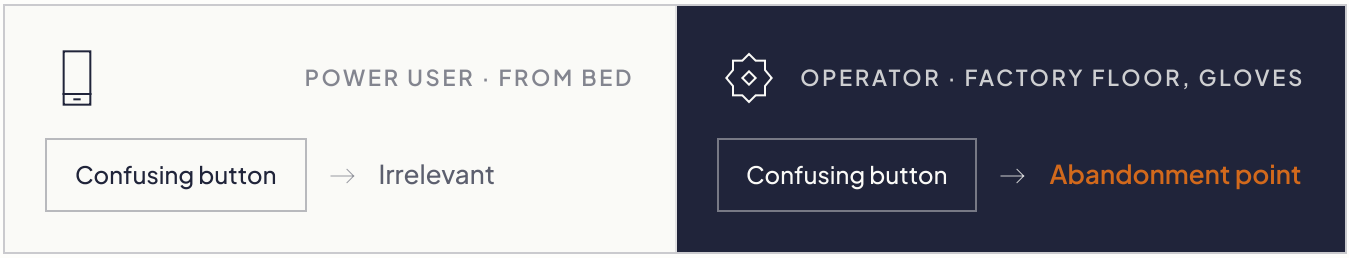
Pablo Manzoni
Product Designer at Kaizen Softworks

Why “find the usability issues” is *not enough*

Give a model a URL and ask it to “find the usability issues.” It works halfway. And the “halfway” is the interesting part: it gives you a generic list, correct in the abstract, useless in practice.

A usability issue matters because of who encounters it and under what conditions.

Using an app from bed is not the same as using it on a factory floor. Urgency changes, lighting changes, attention changes, previous knowledge changes. The same confusing button can be irrelevant to a power user and an abandonment point for an operator wearing gloves.



KEY INSIGHT

The AI does not evaluate the interface. It acts as a *specific person* in front of the interface.

The person brings the context with them. And the context turns a list of defects into a list of priorities.

The analysis was never *the point*

I built this to find usability issues, and it does that. But after using it on real projects, the run turned out to be the smallest part of the value. What matters is what the AI does with it afterwards.

WHAT I THOUGHT I WAS BUILDING	WHAT IT ACTUALLY BECAME
A test	**A context layer**
URL in, report out. Read once, acted on, archived.	Profiles, inferred tasks, business research and emotional arcs that the AI reads every time a product decision is made.

1 **Ask for issues**
A generic list. Correct in the abstract, useless in practice.

2 **Give the AI a person**
Context turns a list of defects into a list of priorities.

3 **Run it inside Claude Code**
Every run leaves profiles, tasks and reports as files in the project.

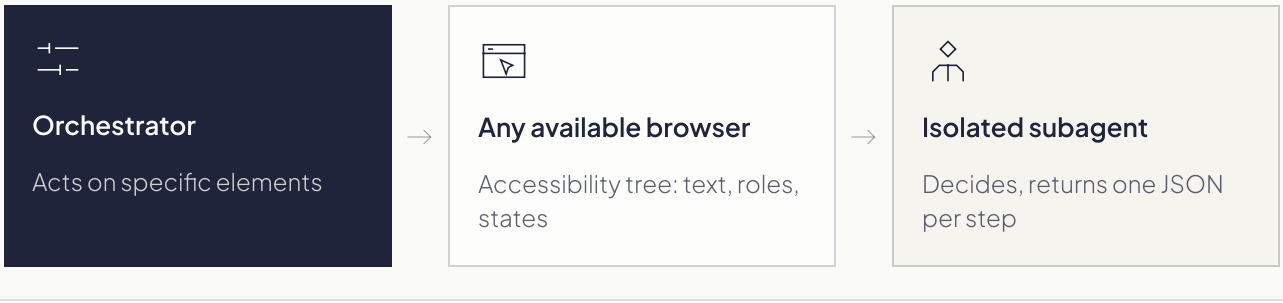
4 **Feed those files back**
Skills and agents load them as context for design, features and strategy.

KEY INSIGHT

A synthetic user is used *always*: while deciding design, functionality, usability and strategy. The end is not the analysis. It is what comes after.

Anatomy of a *simulation*

An orchestrator drives the browser and reads each screen as an accessibility tree: text, roles, states, a reference per element. No guessing pixels. It does not need a specific browser. It uses whichever one is already there.



BROWSER, PICKED AUTOMATICALLY

1. Playwright MCP, if you already have it
2. Claude desktop's built-in browser
3. Claude in Chrome, always in a new tab
4. Read-only: public pages and links, no forms

Nothing to install. Playwright is optional, no longer required.

FIVE OPERATIONS, ANY DRIVER

`OPEN` `READ` `ACT` `WAIT` `SHOT`

Skills describe browser work in these five verbs. The driver only translates them.

The decision on each screen is made by an isolated subagent, which returns a JSON for each step:

STEP OUTPUT · JSON

```
{
  "action": "...",
  "clarityLevel": "High|Medium|Low",
  "doubtDetected": true,
  "reason": "...",
  "abandoned": false,
  "estimatedTimeSeconds": 40,
  "emotionalState": "...",
  "memory": "..."
}
```

Two rules make this look more like a person and less like an oracle.

1 The evaluator never sees the end.

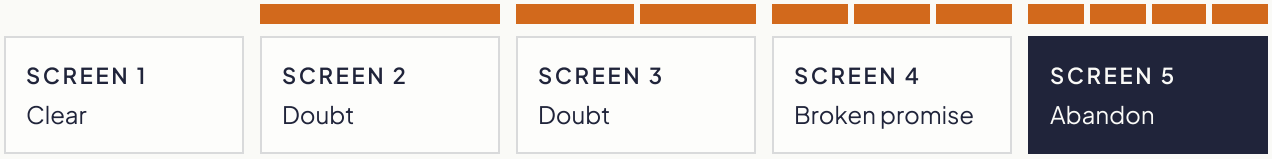
The evaluator receives one screen at a time, without knowing how many are left or what comes next in the flow.



If the interface leaves room for a mistake, the synthetic user makes the mistake. It clicks where a person would click, not where it is convenient to click in order to complete the test. This is where the framework's forbidden assumptions live: the agent cannot assume backend logic or mentally complete what the screen does not show.

2 Emotion is memory, not decoration.

The memory field travels from one step to the next. The emotional state is inherited and accumulates. A frustration +1 persists. This detects something that is structurally invisible to any test that evaluates screens separately.



Screen five does not necessarily fail because of screen five. It fails because the user gets there with accumulated frustration.

Nine subagents, *one job each*

Each subagent gets a clean context. It knows the minimum required to do its job.

That ignorance is deliberate.

The agent acting as the user does not know what the orchestrator knows. It cannot compensate for bad design with knowledge a real person would not have.

SUBAGENT	WHAT IT DOES
`synthetic-screen-evaluator`	Acts as the user on one screen and returns the JSON for that step
`synthetic-flow-synthesizer`	Reads the complete run and writes the report. It never simulates again
`synthetic-profile-generator`	Generates a complete profile from an approved spec, choosing from a controlled vocabulary
`synthetic-autopilot-synthesizer`	Consolidates N runs and classifies findings by convergence across users
`heuristic-persona-generator`	Creates the 3 persona raters based on the business being evaluated
`heuristic-expert-evaluator`	Detects violations of the 10 heuristics using forced enumeration
`heuristic-persona-rater`	Scores each finding from the experience of ONE persona. It runs ×3
`heuristic-report-synthesizer`	Builds the final report using the already computed numbers
`summary-synthesizer`	Distills every full report into a one-page action summary. It never judges again

Adapting a human test: the *heuristic evaluation*

A textbook heuristic evaluation uses three to five human evaluators because each one finds different problems. My first experiment copied that literally, and it was mediocre. Two synthetic detection runs got coverage extremely high, but produced an unmanageable list: dozens of valid issues, very few important ones.

The final design separates detection from prioritization.

1 An expert finds violations

Based on Nielsen's ten heuristics, it is forced to give a verdict for every heuristic on every screen, with textual evidence:

- Violation
- Clean
- Not observable

Forced enumeration breaks the habit of reporting only what stands out.

2 Three personas decide what matters

Generated from the business being evaluated. They score findings without seeing the expert's conclusions:

- Power user
- Average user
- Low digital literacy

The same issue matters differently depending on what each persona brings.

PRIORITIZATION FORMULA

business impact × usability impact

Tiebreaker: agreement between personas

06 USAGE

One command, *three questions*

```
user simulation https://your-app.com
```

1	What do we run?	User simulation · Heuristic review · Both
2	Which users?	Inferred: it reads your site and builds 3 realistic users · Or your own profile
3	What scope?	One screen · Or a flow: it proposes tasks, you pick one or write your own

Then it runs on its own. No more questions until the summary.

HARDCORE MODE · THE THREE ENGINES BEHIND IT, FOR FULL CONTROL

simulation-run	EXPERIENCE	Your own profile, URL and task
simulation-auto	EXPERIENCE	Inferred users you can curate: add, remove, edit tasks and context
heuristic-test	INSPECTION	Screen, flow or site crawl, page cap, credentials

SYNTHETIC USER BUILDER · SYNTHETIC.TUGGSY.COM

The profile describes how someone decides, never what they have to do. That is why the same profile can be reused across tests, and across decisions.

One page to act on, full reports *one click away*

Every run ends with a one-page summary written by the ninth subagent. It does not judge again. It chooses and merges: the same problem seen by three users and by the expert becomes one finding with a strong “seen by” column.

IN ONE LINE

2 of 3 users booked a court; checkout’s “Process” button stopped the first-timer.

#	WHAT TO DO	FIXES	SEEN BY	CRITICALITY
1	Rename “Process” to “Pay $120” on checkout	F1	U1 U2 · Expert	High
2	Show court availability before asking for login	F2	U1 U3	Medium

What we found | How they felt | Friction map | What works

Merge before listing

Agreement between users and expert is the strongest signal.

Breadth × severity

What blocked everyone outranks what slowed one person.

Top three or four

The rest stays in the full reports. A summary with twelve rows failed.

The files are *the product*

Every run leaves Markdown inside the project. Keeping everything as accumulating .md files is the most strategic decision in the plugin: Markdown is what an AI reads natively, and every run adds to the same body of knowledge.

PROJECT STRUCTURE

```
user-simulation-tests/
├── <date>-<domain>-summary.md  ← the one-page summary
├── simulation/
│   ├── profiles/   ← synthetic users (.md + .builder.json)
│   └── results/    ← per-user reports + consolidated report
└── heuristic/
    ├── personas/   ← the 3 raters + business research
    └── results/    ← prioritized findings
```

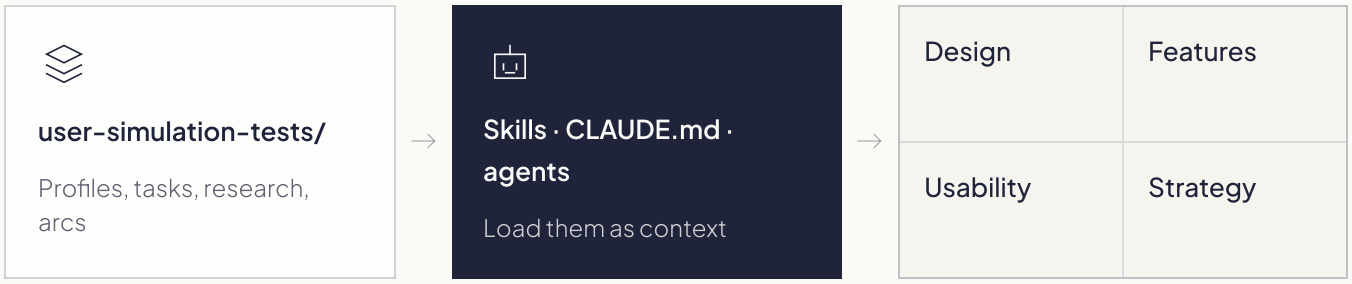
WHAT EACH FILE BECOMES AFTER THE RUN

Profiles	→	How each type of user decides: role, context, limits, emotional starting point
Inferred tasks	→	An inventory of the real jobs people come to do
Business research	→	Domain context the AI no longer has to guess
Results and arcs	→	Known friction, and where emotions turn
Summary	→	A prioritized seed for the backlog

Different runs, different lenses, one project. Crossing them answers *what actually matters*, and keeps answering it later.

The synthetic user *stays in the project*

The AI consumes the synthetic users through skills and through Claude's agents and project instructions, which add them as context to every request. The profiles stop being test inputs and become part of how the project thinks.



The question changes. Not "is this usable?" but "is this right for this person, with this focus, for this business goal?"

PROMPTING WITH A SYNTHETIC USER IN CONTEXT

```
› Reorder this screen for the operator profile. They come to check today's
  orders, not to explore.

› The business goal here is engagement. What should the next action below this
  content be?

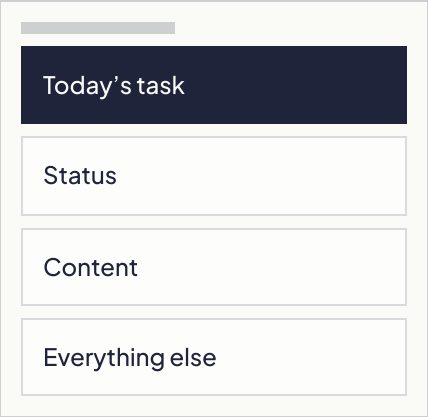
› Would the first-timer understand the price before reaching checkout? Move it if
  not.
```

Same app, *different screens*

With a profile in context, you can ask the AI to reorder a screen around a type of user and their focus, which goes beyond usability. And you can align it with the business goal. The content is the same. The order changes.

USER FOCUS

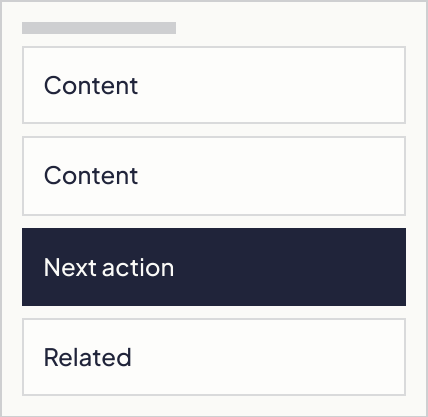
Operator



Reorder by what this person came to do, more than by usability.

BUSINESS GOAL

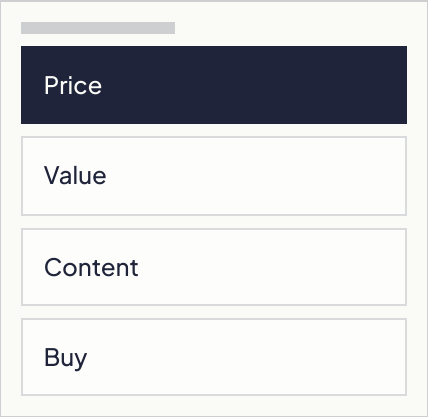
Engagement



There has to be a button for the next action right below.

BUSINESS GOAL

Sales



Key information, like pricing, is clear much earlier.

KEY INSIGHT

The application is the same. The emotions the synthetic profiles go through shape how the screens are drawn. That is *designing in the age of AI.*

Models and *costs*

What worked for me for the subagents:

BEST AVAILABLE MODEL	MEDIUM AND FAST	LIGHTEST
Reports, consolidation, the summary and the heuristic expert. That is where the judgment lives.	The screen evaluator. Many short, constrained calls, and the profile already restricts the decision.	The raters are the lightest case.

100k–400k
TOKENS PER COMPLETE RUN

3–5 min
PER SIMULATED USER

4–6 min
HEURISTIC REVIEW

That is the cost of a test that previously required coordinating the schedules of three professionals. Now it can run against every iteration of the product, and each run leaves the project with more context than before.

It is still early, but it already runs

Every rule in the framework became an architectural constraint: clean context, one screen at a time, emotional memory, forbidden assumptions.

One command, three questions, and the browser you already have. The plugin is open source: github.com/PabloManzoni/user-simulation.

If your synthetic user abandons on screen three, it is not failure. It is the signal.

And when the run is over, keep the user around:

The report is read once. The user *stays.*

PREPARED BY



Pablo Manzoni
Product Designer at Kaizen Softworks

OPEN SOURCE

github.com/PabloManzoni/user-simulation

© 2026 KAIZEN SOFTWORKS