BA TOOLKIT · CHECKLIST

Agent-Ready Requirements

An 8-point checklist, EARS patterns and a worked before-and-after for writing requirements that AI coding agents and spec-driven tools can build without guessing.

AI agents | Spec-driven development | EARS notation | Acceptance criteria

Prachit Parikh

Business Analyst · Requirements, SQL data validation, UAT

Free download: github.com/prachitparikh/ba-toolkit

Why this checklist exists

AI coding agents and spec-driven tools like GitHub Spec Kit and AWS Kiro build exactly what you write, at machine speed, and they rarely stop to ask what you meant. A vague requirement used to cost a clarifying meeting. Now it costs a confident, plausible build of the wrong thing.

Run every requirement through this list before it goes to an agent or an AI-assisted developer. It works just as well for human teams.

The checklist

- [ ] **One behavior per requirement.** "Validate, save and notify" is three requirements.
- [ ] **Trigger, condition, outcome.** When X happens, if Y is true, the system does Z.
- [ ] **Numbers, not adjectives.** "Fast" becomes "under 2 seconds for 95% of requests."
- [ ] **Named data.** "Customer status" becomes "customer.status, one of Active, On Hold, Closed."
- [ ] **Unhappy paths written down.** Blank fields, timeouts, duplicates, permissions.
- [ ] **At least one concrete example.** Input in, output out.
- [ ] **A test you could run.** If you can't say how you'd prove it, it isn't finished.
- [ ] **Traceable.** Linked to its source (workshop, regulation, ticket) and to its test case.

EARS patterns

EARS (Easy Approach to Requirements Syntax) gives each requirement a fixed shape, so the trigger, the state and the response can't hide inside a long sentence.

Pattern	Template
Event-driven	`WHEN [trigger], THE SYSTEM SHALL [response].`
State-driven	`WHILE [state], THE SYSTEM SHALL [response].`
Unwanted behavior	`IF [condition], THEN THE SYSTEM SHALL [response].`
Optional feature	`WHERE [feature is included], THE SYSTEM SHALL [response].`
Ubiquitous	`THE SYSTEM SHALL [response].`

Before and after

BEFORE

"Flag suspicious transactions for review."

AFTER

"**WHEN** a payment over $10,000 is created for a beneficiary added in the last 24 hours, **THE SYSTEM SHALL** set review_status to PENDING_REVIEW and block release until a user with the Reviewer role approves it."

Example: a $12,500 payment to a beneficiary added 3 hours ago is held; the same payment to a beneficiary added 30 days ago is released.

The first version gets you plausible code. The second gets you the right code, and the example doubles as a test case.

Questions to ask before handing a spec to an agent

1. What should happen when the data is missing or wrong?
2. Who is allowed to do this, and who isn't?
3. What does "done" look like in numbers?
4. Which existing behavior must not change?
5. How will we know, with data, that the agent built what the business meant?

Use it with the rest of the toolkit

- [ ] **Traceability matrix:** link each requirement to its source, design, test cases, defects and sign-off.
- [ ] **UAT plan:** turn every example into a test case with an expected result.
- [ ] **SQL checks:** prove with data that the build did what the requirement said.

> The agent can write the code. Deciding what the code should do, precisely enough to test, is still the analyst's job.

About Prachit Parikh

Prachit Parikh is a business analyst. Most of his work sits where requirements, data and testing meet.

He takes what stakeholders say they need and turns it into requirements and user stories a team can build and test. Each one has acceptance criteria and a clear line to the test that proves it. He checks the data behind them in SQL and Excel, builds reports in Power BI, and plans UAT so a change goes live the way the business expected.

He was a business analyst at Banking Circle, working on data validation, traceability and UAT planning. Before that he worked in customer care at Elevance Health and did business systems analysis on an employee portal and an inventory system at Saklax Manufacturing.

He writes about AI, data and the future of business analysis, and makes short animated explainer videos on YouTube.

Related reading

1. AI agents can write the code now. The bottleneck is the requirements. (Medium)
2. What a good UAT plan actually looks like (Substack)
3. 7 SQL checks I run before UAT sign-off (Dev.to)

> Spotted a gap or have a template that saved a project? Open an issue or pull request on the toolkit repo. Real-world examples (anonymized) are welcome.

Find Prachit Parikh online

WEBSITE
prachitparikh.github.io

HUB SITE
sites.google.com/view/prachit-parikh

YOUTUBE
youtube.com/@PrachitParikh1

BA TOOLKIT
github.com/prachitparikh/ba-toolkit

MEDIUM
medium.com/@prachit.parikh

SUBSTACK
prachitparikh.substack.com

HASHNODE
prachitparikh.hashnode.dev

DEV.TO
dev.to/prachitparikh

WORDPRESS
prachitparikhblog.wordpress.com

BLOGGER
prachitparikh-ba.blogspot.com

GITHUB
github.com/prachitparikh

LINKEDIN
linkedin.com/in/prachitp

KAGGLE
kaggle.com/prachitparikh

BLUESKY
bsky.app/profile/prachitparikh.bsky.social

QUORA
quora.com/profile/Prachit-Parikh-2

ABOUT.ME
about.me/prachitparikh

LINKTREE
linktr.ee/Prachit_Parikh_BA

ISSUU
issuu.com/prachit_parikh

CRUNCHBASE
crunchbase.com/person/prachit-parikh-ct

CREDLY
credly.com/users/prachit-parikh

GRAVATAR
gravatar.com/prachitparikhba