

A Gopher's Guide to Vibe Coding

Daniela Petruzalek

Developer Relations Engineer @ Google Cloud

@danicat83



Who am I?

Developer Relations Engineer at Google, originally from Brazil 🇧🇷, but living in the UK 🇬🇧 since 2019.

My background is in backend and data engineering. Have been doing Go development (on-and-off) since 2017.



Agenda

Introduction to Vibe Coding

Types of Coding Assistants

AI Assisted Developer Workflow

Live Demo: Jules & Gemini CLI

Practical Tips & A Peek at the Future



Andrej Karpathy ✓

@karpathy



...

There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.

6:17 PM · Feb 2, 2025 · **4M** Views



1.2K



3.9K

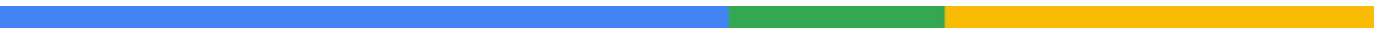


24K



12K





**A coding approach that
relies on LLMs to generate
working code by providing
natural language
descriptions rather than
manually writing it**

Types of Coding Assistants



Code Completion

Traditional IDE
autocomplete with
“Ghost Code”

Gemini Code Assist
or GitHub Copilot on
VS Code



Contextual Chat

Conversational
interfaces on the IDE
via chat add-on

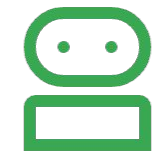
Gemini Code Assist
or GitHub Copilot
Chat on VS Code



Coding Agent

CLI interfaces with a
REPL, context is the
entire project,
“pairing”

Gemini CLI, Claude
Code, Aider



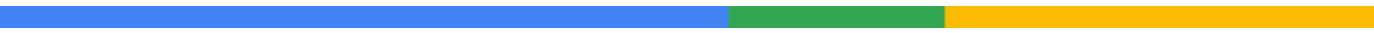
Autonomous

“Fire-and-forget”:
completely async

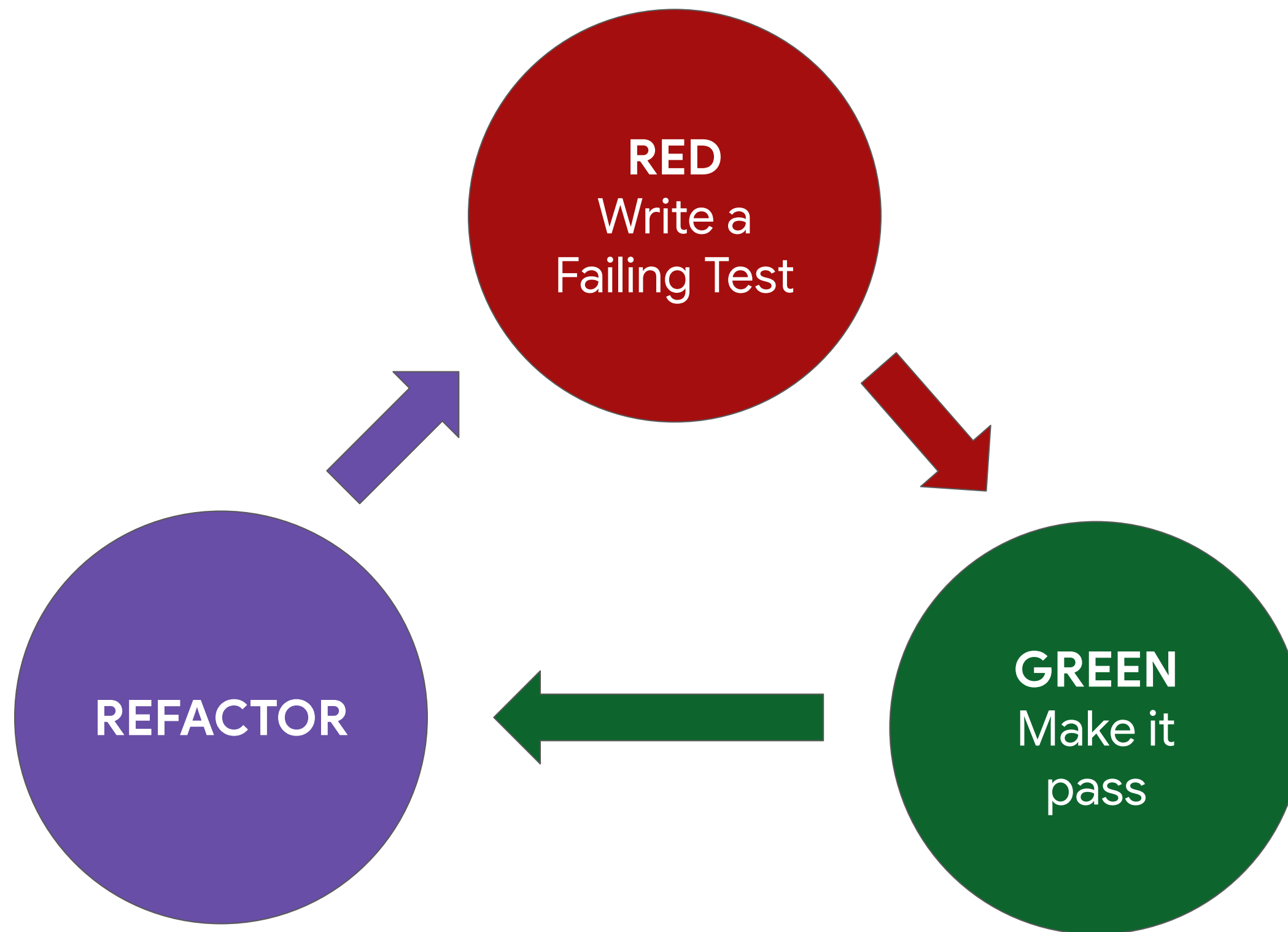
Google Jules,
Devin AI, GitHub
Copilot Agent

Biggest Challenges Today

- LLMs are non-deterministic by nature
- LLM generated code is mostly “not great”
- Prototypes and full systems are different beasts
- Coding environments are not entirely fit for purpose
- Code correctness - does it do what is supposed to do?
- How to ensure code quality and maintainability over time



Vibe Coding with the
“vanilla” models can be
equal parts amazing and
infuriating... Can we make
it better?



Vibe Coding is “TDD on Steroids”

- RED: Start with a feature request (failing test)
 - Keep features small
 - Reduce ambiguity => improve the prompt
- GREEN: Make it work (make the test pass)
 - No code unrelated to the feature should be written
- Refactor (only allowed on green)
 - Code review, test coverage, linters, documentation, ...

Improving Response Quality

- Give the model more up to date / relevant information
 - Prompt / Context Engineering
 - Retrieval Augmented Generation (RAG)
- Grounding with Tool Calling (e.g. Web Search, APIs)
 - Model Context Protocol (MCP)
 - Not only retrieve data, but perform actions

HOW STANDARDS PROLIFERATE:

(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

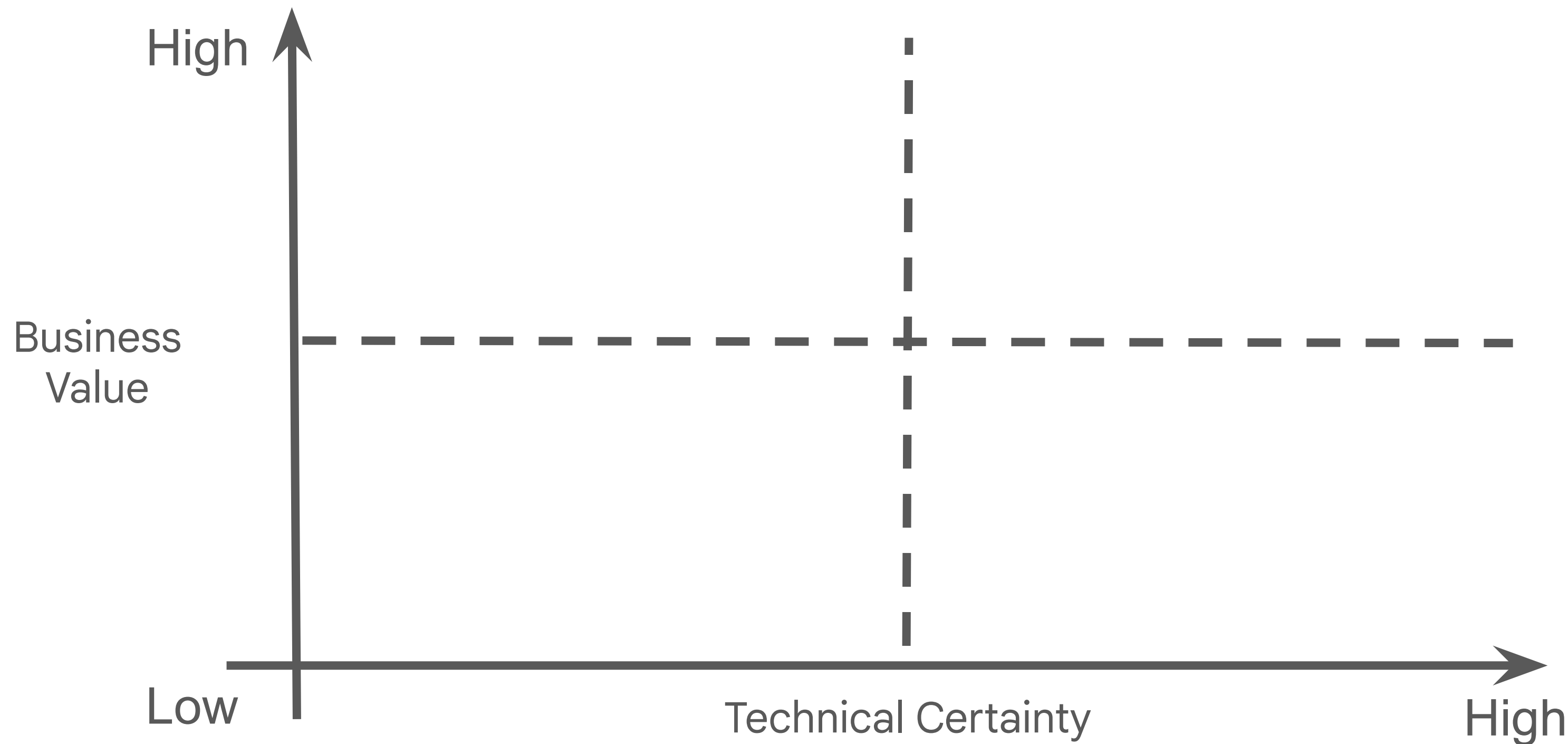
14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.



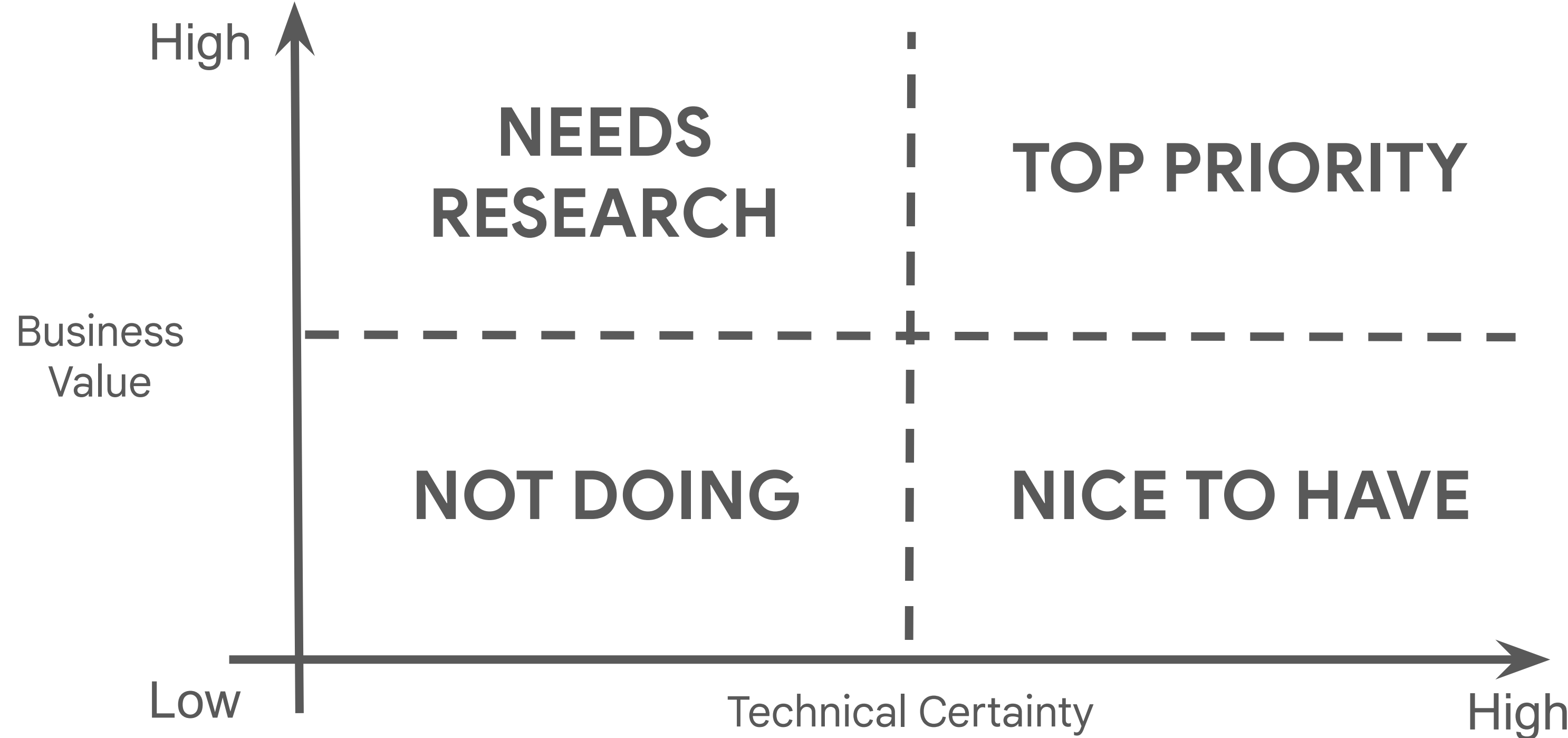
SOON:

SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

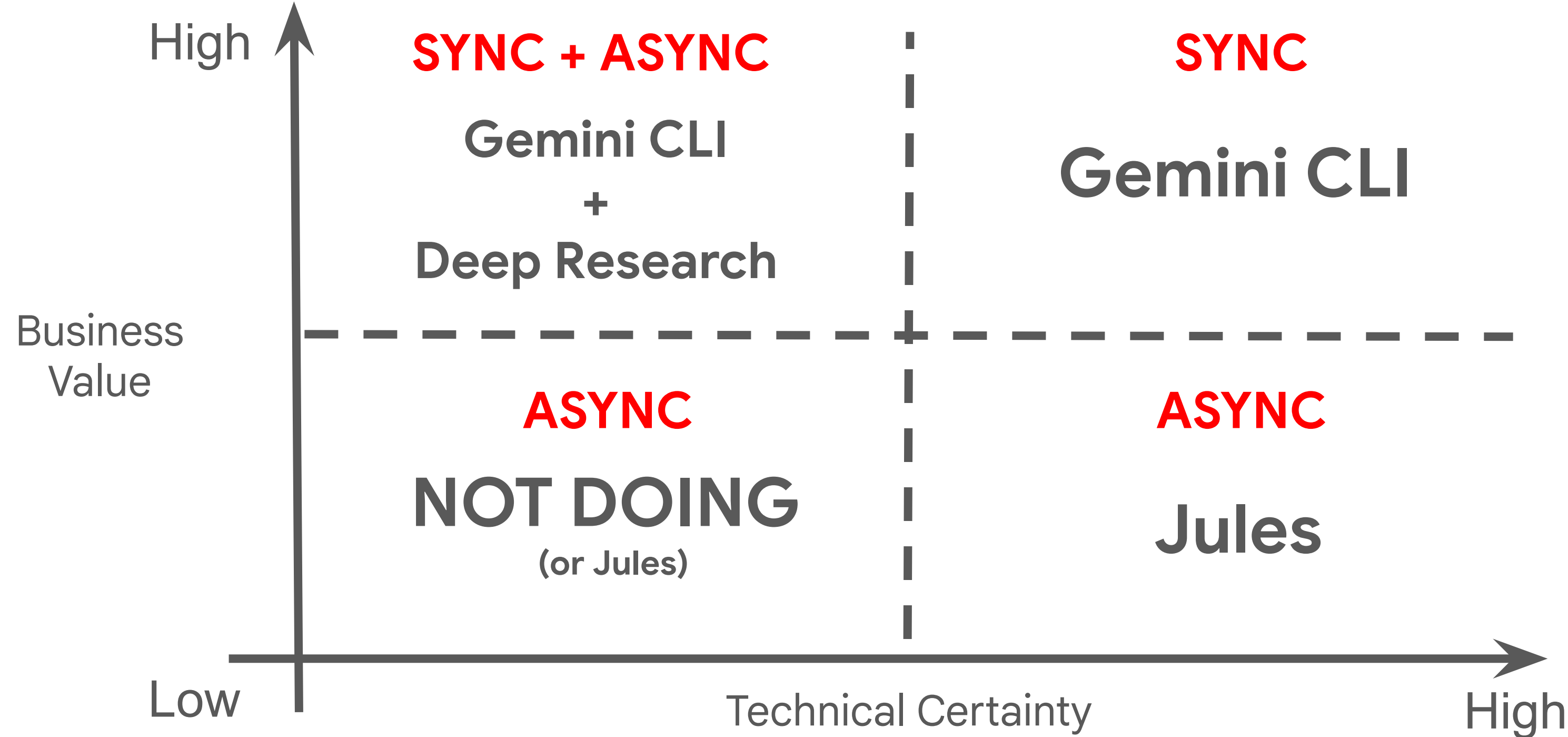
My Current Workflow



My Current Workflow



My Current Workflow

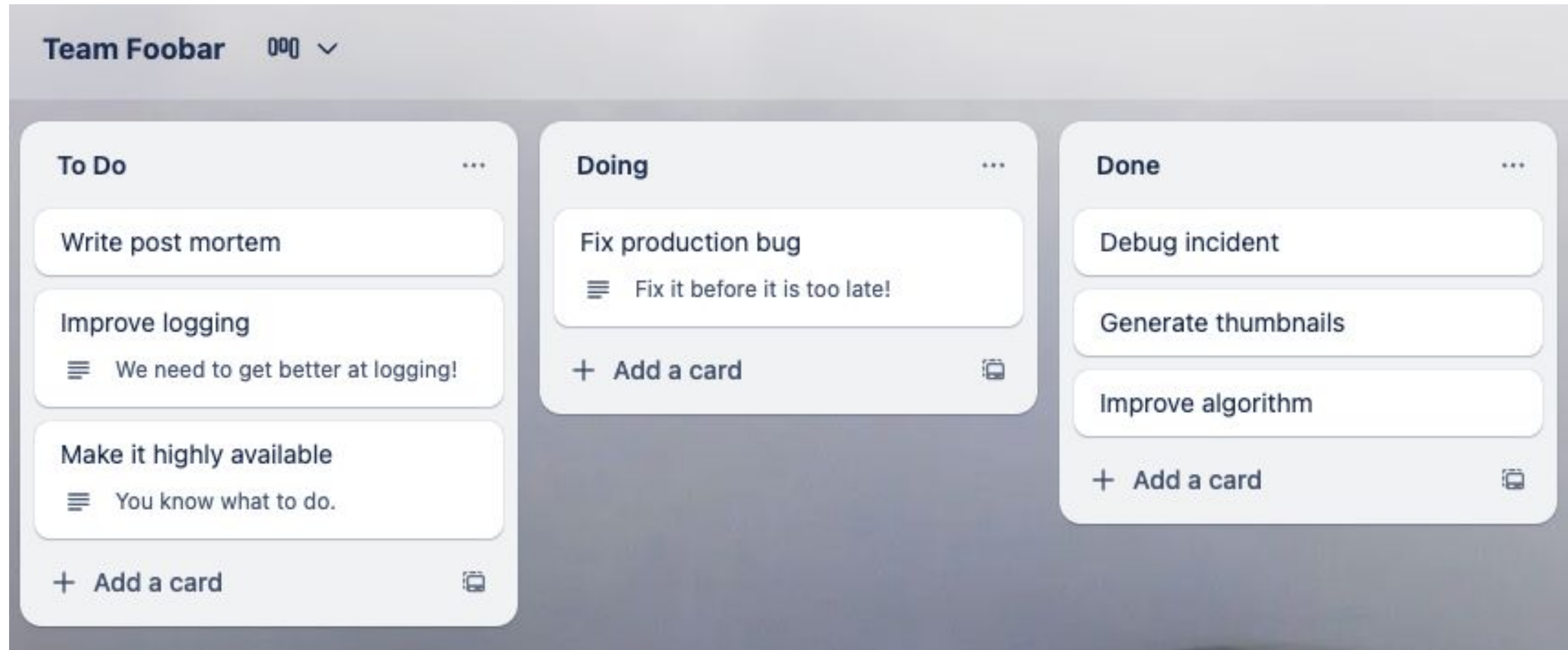


Demo: Jules

<https://jules.google/>

<https://github.com/danicat/testquery/issues/5>

Does your team write stories like this?



Good Prompts = Good Stories

```
ticket_template.txt
1 Context
2 =====
3
4 The first session should be the "context", which shall be composed minimally of: 1) a brief description of what we have, and; 2
5
6 Include references to complementary material when necessary, e.g.: link to github repo, docs, etc, but please make sure the tic
7
8 To Do
9 ====
10
11 This is the list of things we want to do, written in a concise way (bullet points).
12
13 Every item in this list must be actionable and have an acceptance criteria that covers it. e.g.:
14
15 - update foo to add bar as required field
16 - backfill schema xyz with bar = baz
17
18 Not to do
19 =====
20
21 List of things we don't want to do (at least now). We should use this to limit the work in each ticket to be small enough to be
22
23 Complex solutions should be composed of two or more tickets. Reference future tickets when applicable. e.g.:
24
25 - don't backfill the data for other schemas (TBD on xxxx)
26 - don't create the airflow operator (TBD on yyyy)
27
28 Acceptance Criteria
29 =====
30
31 List of the things the person doing the acceptance test must see. e.g.:
32 - querying foo on xyz should show bar field
33 - records without bar should trigger an error
```

<https://gist.github.com/danicat/854de24dd88d57c34281df7a9cc1b215>

A Good Prompt Template

1. Task: description of what you want to achieve, including context, constraints, etc. For example: use SDK x, library y.
2. References: links to documentation, reference implementations, etc.
3. Testing: how to test the successful implementation; example commands with output.

Let's try something practical!

(what could go wrong?)
(again?)



The Prompt

Your task is to create a Model Context Protocol (MCP) server to expose a “hello world” tool. For the MCP implementation, you should use the official Go SDK for MCP and use the stdio transport.

Read these references to gather information about the technology and project structure before writing any code:

- <https://github.com/modelcontextprotocol/go-sdk/blob/main/README.md>
- <https://modelcontextprotocol.io/specification/2025-06-18/basic/lifecycle>
- <https://go.dev/doc/modules/layout>

To test the server, use shell commands like these:

```
(
  echo
  ' {"jsonrpc": "2.0", "id": 1, "method": "initialize", "params": {"protocolVersion": "2025-06-18"}} '
;
  echo ' {"jsonrpc": "2.0", "method": "notifications/initialized", "params": {}} ' ;
  echo ' {"jsonrpc": "2.0", "id": 2, "method": "tools/list", "params": {}} ' ;
) | ./bin/godoctor
```

Typical Error Modes



Fully Recoverable

Typos, compilation errors (e.g. syntax), edit errors (e.g. duplicate lines)

Model usually recovers on its own given enough time



Intervention

Wrong reasoning (e.g. tests vs code), placeholder code, hallucinations

Model **might** recover, but you will be better off giving it a nudge



Full Stop!

Refactor loops; insisting on not following directions

Model **will not** recover and potentially spend lots of \$\$\$\$. Take over and course correct

Tips for a Successful Experience

- **Don't be afraid to interrupt the AI.** The AI will sometimes propose actions or code that you don't agree with. Cancel and course correct.
- **Encourage tool use.** If the AI seems lost or is making up information, encourage it to use its available tools to do web searches, look for examples, inspect files, etc.
- **Have you tried turning it off and on again?** In the (not so) rare case that the AI is misbehaving, clearing the context and giving it a clean start with a better prompt is often the best option.

A Peek at the Future of Go

Go SDK for MCP: <https://github.com/modelcontextprotocol/go-sdk>

MCP Support for gopls: <https://tip.golang.org/gopls/features/mcp>

Google Gen AI SDK: <https://github.com/googleapis/go-genai>

OpenAI Go API Library: <https://github.com/openai/openai-go>

Gemini OpenAI Compatibility:

<https://ai.google.dev/gemini-api/docs/openai>

Useful MCP Servers

Playwright: navigates web pages, takes screenshots, ...

<https://github.com/microsoft/playwright-mcp>

Context7: retrieves documentation from a crowdsourced repo

<https://context7.com/>

GoDoctor: Go coding assistant

<https://github.com/danicat/godoctor>

Speedgrapher: prompts and tools for technical writing

<https://github.com/danicat/speedgrapher>

What About Building Your Own?

[How to Build a Coding Assistant
with Gemini CLI, MCP and Go |
Google Codelabs](#)



In Summary

For successful vibe-coding sessions:

- Develop your prompting (and story writing) skills
- Prioritise your tasks and use asynchronous coding for the less critical ones
- Actively manage your context window
- Equip your toolbox with MCP servers: find what is useful or build your own (it is easy and fun to do)

Thank you!

<https://danicat.dev>

[linkedin.com/in/petruzalek](https://www.linkedin.com/in/petruzalek)

@danicat83

Google Cloud

