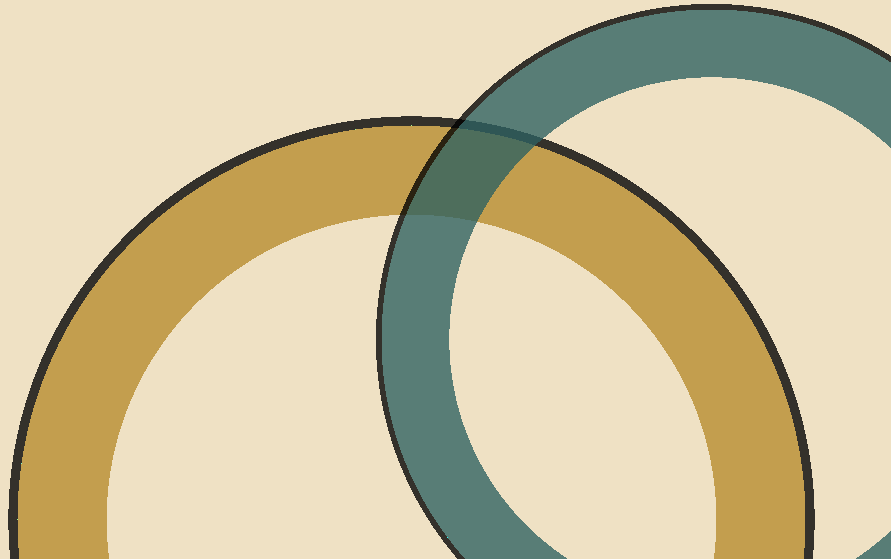
Engineering AI Agents · Week 7

Observability and debugging

Recording one run · watching all of them · making a failure happen twice · the six bugs you will meet



Slides CC BY 4.0 · from the teaching kit of *AI Agents, Engineered* by Enrique Gutiérrez · aiagentsengineered.com/teach/

Your agent has been in production for a month. A user reports a wrong answer, politely, with a screenshot.

You have a stack trace, the logs, and the user willing to try again. Which one finds the bug?

Chapter 15, opening · the scene follows a trace-first debugging guide

By the end of today you can

Four objectives

1. Make every model call and tool call a **span** that carries what Chapter 15 asks
2. Record a run at the **seam** and **replay** a failure with nothing live
3. Read a trace to the **first wrong span** and name its **bug shape**
4. Turn a fixed failure into an **eval case** and a **golden trace**

Part 1 · Chapter 15 (full book)

Traces and spans

Instrument as it runs, capture everything that mattered, rely on nobody’s memory

The trace is a flight recorder

- Investigators do not ask the pilot to remember, or crash a second aircraft
- A **trace**: the complete, structured record of one run, from the request to the final answer
- One limit: an aircraft would fly the same route twice; **your agent would not**

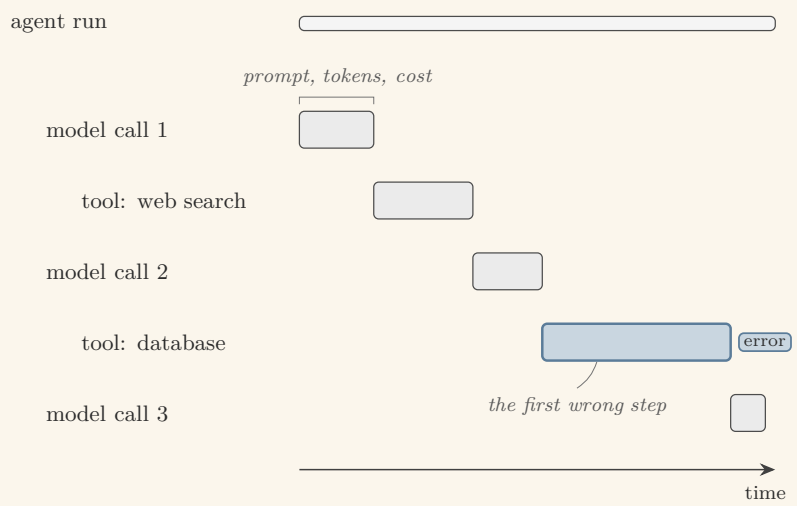
Name	The same record, read by
Transcript	the conversation, in order
Trajectory	the evaluation seat: one sampled path
Trace	engineering: queryable, with its measurements

Chapter 15, “Traces and Spans” · “One object, three trades looking at it.”

Borrowed from distributed tracing, which had the same problem first

A trace is a tree of spans

- A **span**: one unit of work, a start, an end, structured attributes
- Root: the request. Under it, each model call; under each, the tools it asked for
- A subagent's whole run: a subtree under the handoff



The bug is the first wrong span, here the database call that erred, not the final answer.

‹ 5 / 5 ›

Figure from Chapter 15 · timings and ordering illustrative

Chapter 15, "Traces and Spans" · "Every LLM call is a span. Every tool call is a span. Every sub-agent is a sub-tree."

The part that decides whether your traces are worth having

What every span carries

Span	Attributes
Model call	the prompt as sent · tokens in and out · latency · cost · the model *requested* and the model that *answered* · why generation stopped
Tool call	name · full arguments · full result and its **size** · latency · on whose authority · a structured error
Session summary	steps taken · cost · final status

Chapter 15, “Traces and Spans” · “The span schema you set up upfront determines whether the next debug session takes 5 minutes or 5 hours.” (a trace-first debugging guide)

Predict before the next slide

Your log stores, for each model call, the prompt template's id and the variables passed to it.

Can it answer "what exactly did the model see at the moment it made this call?" Name three bugs it would hide.

Chapter 15, "Traces and Spans" · the chapter's own exercise

Store the final assembled text that went to the model

The rendered prompt

> *The rendered prompt is what the model actually saw. The template plus variables is what you intended. Bugs live in the difference.*
>
> Chapter 15, “Traces and Spans” (a trace-first debugging guide)

- Store a **hash** of it too: one comparison says whether the model’s input changed
- A **session summary** span per run: sorting runs by cost becomes one query

What the narration is worth, how to name things, what not to keep freely

Testimony, a standard, and care

Testimony

The model's interleaved reasoning says what it believed it was doing. Read it; check it against the calls and results.

A standard

An open telemetry standard names the AI attributes. Still changing at the time of writing: pin its version; copy what alerts read.

Care

Prompts and results hold users' data. Content capture: deliberate, redactable, off by default. Add the principal.

Chapter 15, "Traces and Spans" · "In a trace, the tool calls and results are ground truth about what happened; the narration is testimony about why."

Part 2 · Chapter 15 (full book)

Production monitoring

Predict before the next slide

Dashboards green: CPU flat, error rate zero, latency in budget. For six days the agent has answered a growing share of billing questions with last quarter's refund policy.

What would have told you, and on which day?

Chapter 15, “Production Monitoring” · the chapter's scene

Signals a user would recognize as the product working, plus the ones that predict the invoice

Three questions, and the tiles

Layer	Asks
Monitoring	Is the system healthy?
Observability	What is happening inside the system?
Debugging	Why did this specific request fail, and how do I fix it?

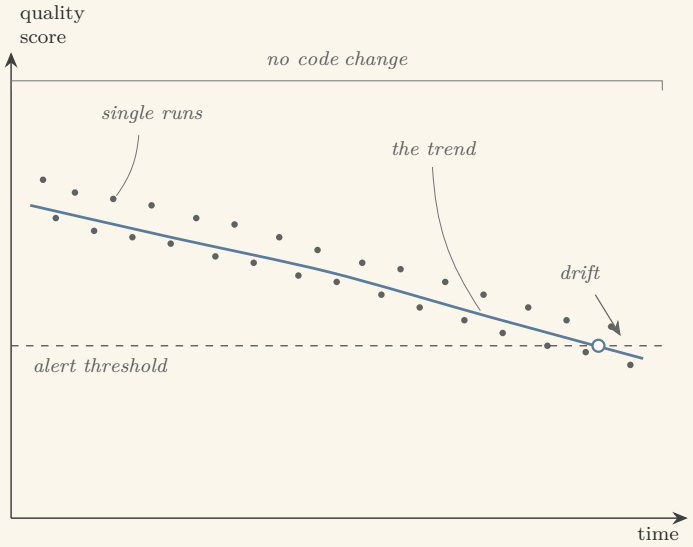
- **Quality**: task success, by a check or a calibrated judge
- **Cost** per run and per day
- **Latency** at the percentiles, not the mean
- **Step counts**: the loop detector
- Errors and retries by tool; **finish reasons**

Chapter 15, “Production Monitoring” · the three questions follow one practical taxonomy cited in the chapter

The slow divergence of behavior, or inputs, from what you validated

Drift is a trend; alert on trends

- Your **users** change: month four is not week one
- Your **model** changes: aliases roll forward
- Your **knowledge** changes: someone else's documents, reorganized
- A **week-over-week** rule catches what no fixed threshold will



‹ 4 / 4 ›

Figure from Chapter 15

Chapter 15, “Production Monitoring”

Two economies and one habit that does not automate

Sample, distrust totals, read traces

Sampling

Capture everything while you learn the bug shapes; then sample successes, and keep every errored run.

No vanity

The tile you glance at every morning should be capable of ruining your morning.

Read, weekly

Sort the week’s runs by cost, steps or a low score. Read the worst ten.

Chapter 15, “Production Monitoring” · first 90 days, then 10–20% of successes, 100% of errors: one guide’s figures, illustrative

Part 3 · Chapter 15 (full book)

Reproducing nondeterministic failures

Predict before the next slide · one minute on paper

Tuesday. A customer was quoted the wrong contract terms. You have the trace: the summary at step six contradicts the source. You rerun the task, and it answers correctly.

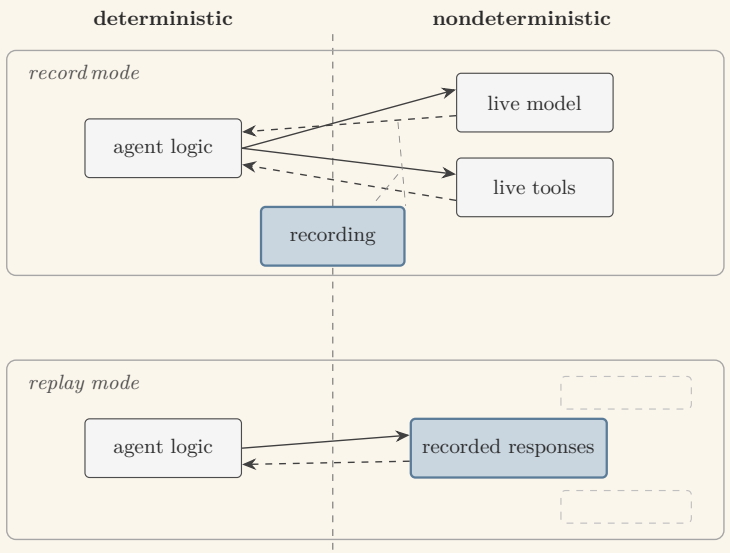
List everything that changed between the two runs.

Chapter 15, “Reproducing Nondeterministic Failures” · the chapter’s scene

Stop trying to recreate the run; record it

Record and replay

- **Record**: run live; journal every crossing of the seam
- **Replay**: your code runs again; every crossing is served from the journal
- No model, no tools, no cost, no side effects
- Anything not in the recording: **fail loudly**



‹ 5 / 5 ›

Figure from Chapter 15

Chapter 15, “Reproducing Nondeterministic Failures” · the engine “fails loudly rather than silently falling through to a live system.”

The first move, this week, before any apparatus

Replay one call

`the failing step’s span` → `its rendered prompt` → `the pinned model, once` → `change one thing`

- **Reproduces**: a tight loop; each attempt costs seconds
- **Does not**: the failure depended on something you did not capture. Add that attribute.
- A recorded run: about **100 KB**, store large payloads once, by hash

Chapter 15, “Reproducing Nondeterministic Failures” · ~100 KB per run of moderate length: the chapter’s figure, illustrative

Recording draws a line through your system
that tells you how to test each side

The deterministic–nondeterministic seam

The model's side

- The output and everything derived from it: a distribution
- Assert **properties**, not strings
- **Pass rates** over many runs (week 8)

Your side

- Tools, parsers, the loop's control logic
- **Exact** assertions, hostile inputs, half-JSON
- The loop against a **scripted model**

Chapter 15, "Reproducing Nondeterministic Failures" · "It passes on Tuesday. It fails on Wednesday because the model reworded one sentence." (an essay on testing agents)

Golden traces, and the limits

Replay correct runs against every new version. The check is structural:

> *Did the agent call the same tools in the same order? Did it extract the same key information? Did it reach the same conclusion?*

- It catches the **prompt regression**
- It **can’t replay what it didn’t record**
- It shows **what** happened, never **why**
- A recording is user data: govern it

Chapter 15, “Reproducing Nondeterministic Failures” · quotations from an article on deterministic replay

Part 4 · Chapter 15 (full book)

A taxonomy of common agent bugs

Six specimens

Specimen	Where the trace shows it	The fix lives in
The stuck loop	the same tool, many times in a row	error surfaces, a backstop, budgets
Hallucinated tool arguments	an argument value with no source	tool types and descriptions
Lost and poisoned context	the failing step's rendered prompt	context assembly, compaction
The swallowed error, silent truncation	a clean call, a clean reply; sizes	structured errors; cut and say so
The wrong stop condition	steps pinned at the cap, or early exits	stop logic, tested with a stub
Silent degradation across deploys	a good trace and a bad one, diffed	version pins, on every span

Chapter 15, “A Taxonomy of Common Agent Bugs”

Commit to one span before the next slide

“No room for six is free tomorrow at 10.”

#	Span	Latency	Result size	Status
1	agent run: “Book a room for six people tomorrow at 10.”	31.6 s	–	completed
2	· model call 1	0.9 s	–	–
3	· · tool: rooms_search {capacity: 6, start: tomorrow 10:00}	30.0 s	13 B	–
4	· model call 2	0.7 s	–	–
5	· session summary	–	–	completed

rooms_search’s client times out at 30 s. Its results this month: median 1.4 kB. The office had four free rooms.

A trace invented for this check · timings and sizes illustrative

The first wrong span

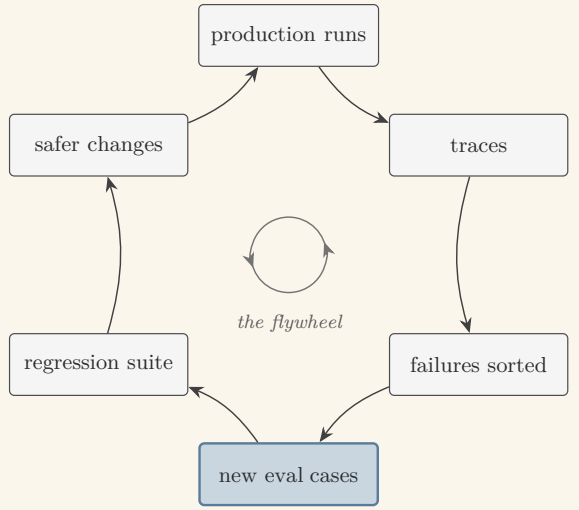
> *read the trace forward to the first step where something is wrong—not the last step, where the wrongness became visible—because that first step is the bug, and everything after it is consequence.*
>
> Chapter 15, “A Taxonomy of Common Agent Bugs”

- With a good trace and a bad one, the reading becomes a **diff**
- The fixes live in descriptions, error surfaces, assembly, stop logic, pins: **the parts you own**

The failing trace in your hand is two assets wearing one file format

The trace-to-dataset flywheel

- The **reproduction**: replay it against the fix; put a run of the fix in the golden library
- The seed of an **eval case**: the input, the wrong behavior, what right looks like
- A quarter of this: an instrument built from real failures



‹ 4 / 4 ›

Figure from Chapter 15

Chapter 15, “A Taxonomy of Common Agent Bugs” · “a measuring instrument built from your agent’s actual failures rather than your feared ones.”

In class · teams of two or three · 24 minutes · two browser tools

Four traces, the first wrong span

1. **A · B · C (12 min).** Trace A: the viewer's stuck-loop sample. Trace B: its cut-result sample. Trace C: paste the June run and the May run from the lecture page; clear the hide box; read the comparison; copy as Markdown
2. **D (8 min).** Paste trace D; clear the hide box again. Then the agent bug bestiary: answer the wizard, pick a card, fill the eval case, copy as Markdown, keep the address

For every trace: the **first** wrong span, by number and name, and what is wrong there. Debrief, 4 minutes

Tools after Chapter 15 · the flag rules and the suspect ranking are the tools' own · traces C and D invented for the exercise

Lab 6, and reading

Lab 6 · a tracer and a replay

- **Spans** with parents, sizes, a summary; on disk as they end
- **Record and replay** at the seam; loud on a miss
- Five runs; a **changed parser** and its structural diff
- **Golden traces** for every tool and the fix; the failure as an **eval case**

Reading for week 8

- Ch. 16, Evaluating Agents `full book`

This week’s reading: Ch. 15 (full book) · glossary: App. B (free) · no paid API needed

Five things to keep

- A trace is a **tree of spans**; the summary span makes it searchable
- Store the **rendered prompt**, both **models**, and every result's **size**
- Drift is a **trend**; alert on trends and read the worst ten
- **Record and replay** at the seam; a miss **fails loudly**
- The **first** wrong span is the bug; the fixed trace becomes an **eval case**

Engineering AI Agents · Week 7

You cannot debug what you cannot see.

Slides from the teaching kit of *AI Agents, Engineered* by Enrique Gutiérrez, CC BY 4.0 · aiagentsengineered.com/teach/

Reuse, adapt and translate with attribution · creativecommons.org/licenses/by/4.0/

