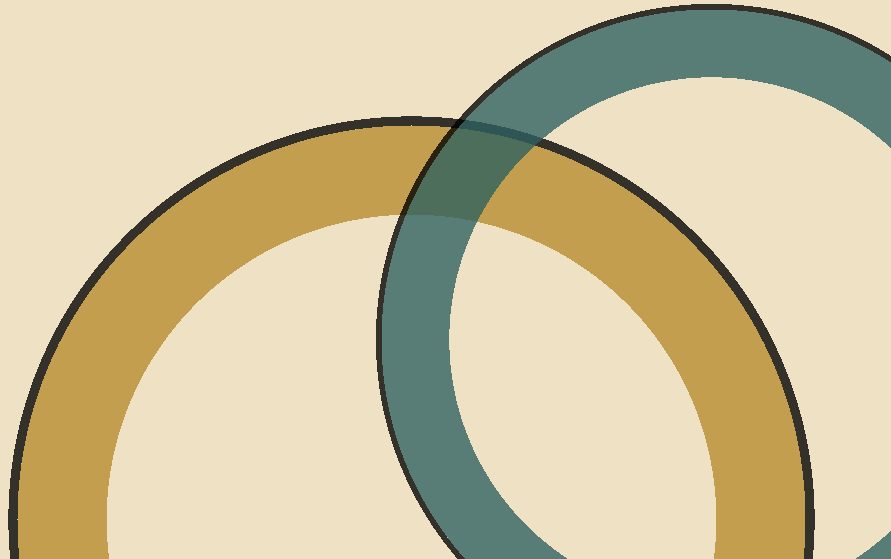
Coding agents and classifiers

Why code went first · the working day · the classifier as a document · measuring it honestly



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

Friday, 5 p.m. Two tasks run unattended all weekend, same model, same harness:

A. “The checkout-total test fails since the currency migration; find the cause and fix it.”
B. “Write an assessment of whether we should expand into a new national market.”

Which result do you open on Monday with more confidence?
Write down why, in one line.

Chapter 21, “Why Coding Led” · the chapter’s own exercise, the market left unnamed

By the end of today you can

Four objectives

1. Explain why coding led, and carry the **ground-truth loop** into another field
2. Run **brainstorm** → **spec** → **plan** → **execute** with tests first, TCR and the **compound step**
3. Build a classifier from **definitions**, a rubric, a flag, and an **ordinal** scorer
4. Develop it with **three sets**, and report **kappa** and the confusion pairs

Part 1 · Chapter 21 (full book)

Why coding went first

The compiler, the suite and the linter were there decades before the agent

The domain grades its own work

Check	Its verdict on
Compiler, type checker	form
Test suite	behavior, with a trace
Linter	hygiene
Build	assembly

- **Cheap**: seconds, no salary
- **Fast**: back while the context is still warm
- **Incorruptible**: unmoved by confident prose
- Green covers a **slice** of correctness, never all of it

Chapter 21 · “Code compiles or it does not. The tests pass or they fail.” · “the river only pushes on what touches the wheel.”

The loop, and what it covers

`find the code` → `edit` → `run the checks` → `read failures` → `green?`

Task	What the check does in it
Questions about a codebase	nothing written, nothing risked: often the first win
Bug fix	the failing test is the goal and the stop condition
Refactor	the suite is the invariant that must survive
Tests, **review**	the agent extends the checker, or reads a diff fresh

Chapter 21, “The Coding-Agent Loop and the Range of Tasks” · two caveats: a weak suite, victory declared early

Predict before the next slide · one minute in pairs

Ana has a demo on Monday. She describes the idea, runs what comes back, pastes the errors in, and ships without ever opening the diff.

Ben builds a service his team will run for years. The agent writes most of the code; he reads every line, rejects what fails his standards, and shapes the architecture himself.

Which of them is using the tool correctly?

Chapter 21, “Augmented Coding versus Vibe Coding” · the chapter’s two engineers, renamed

The same tool at both ends; what differs is what you decided to own

A spectrum of intent

- **Vibe**: judge the behavior, leave the code unread
- **Augmented**: hold the hand-coding bar while a machine types
- Set the point **per artifact**, before the first prompt, by how long it lives
- Warning signs: loops, unasked features, **tests disabled or deleted**

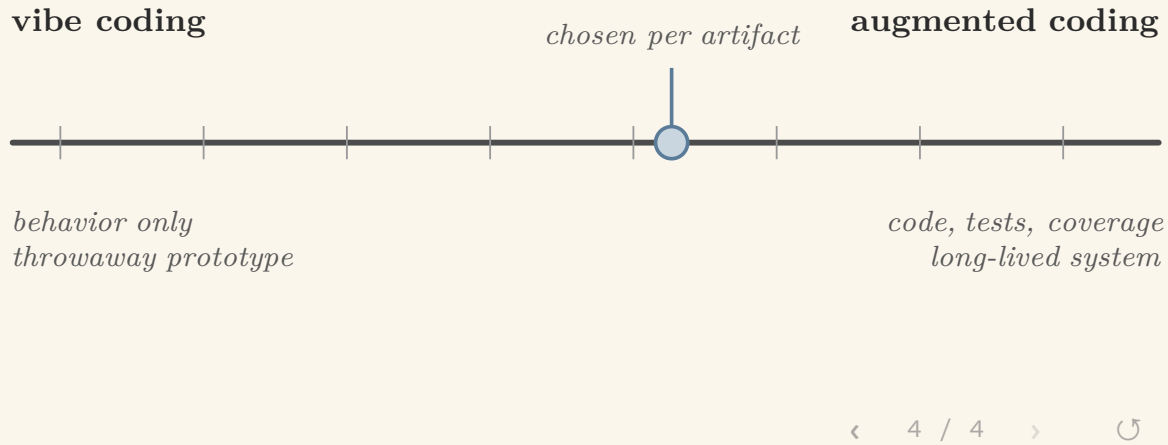


Figure from Chapter 21

Chapter 21, quoting Kent Beck: “In vibe coding you don't care about the code, just the behavior of the system.” · “Same action; opposite acts.”

The reverse adaptation: shape the codebase for a reader that sees a few files at a time

Agent-legible code

- **Local reasoning**: a change understood from a handful of reads
- **Greppable names**; no barrels, aliases, wildcard re-exports
- Consequential checks **where the action is**
- Errors as **values** the call site can see
- Edits that stay **local** in the diff

The toolchain

Fast: it runs inside the loop.
Truthful: no green while the types fail.
Uniform: one way to run things.

Chapter 21, “Making Code Agent-Legible”, quoting two practitioners: “agents really like local reasoning.” · “When an agent struggles, so does a human.”

Strip the story to a template with four slots

The ground-truth loop

1. A **goal** a machine can check
2. Actions that **leave evidence**
3. A **verifier** cheap enough for every turn, independent of the work
4. A **person** reviewing intent, above the slice

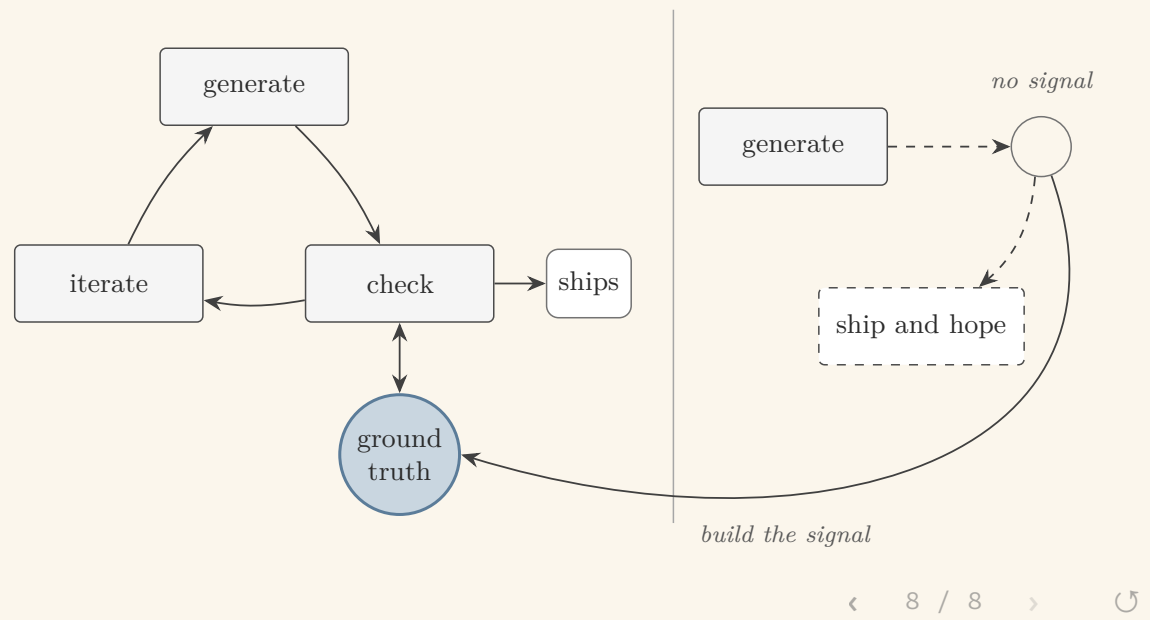


Figure from Chapter 21

Chapter 21, “The General Lesson” · if the answer is “there isn’t one,” “then building one is the project”

Part 2 · Chapter 22 (full book)

Briefs down, receipts up

The workflow the field copied: its first prompt asks the model to ask you

Brainstorm, spec, plan, execute

- **Interview**: one question at a time, until the spec exists
- **spec.md** in the repository, not in a chat
- **Plan** in chunks small enough to test, each one wired in
- A **checklist** ticked on disk: state outside any context
- The **compound step**: what should the agent have known? One line in the standing file

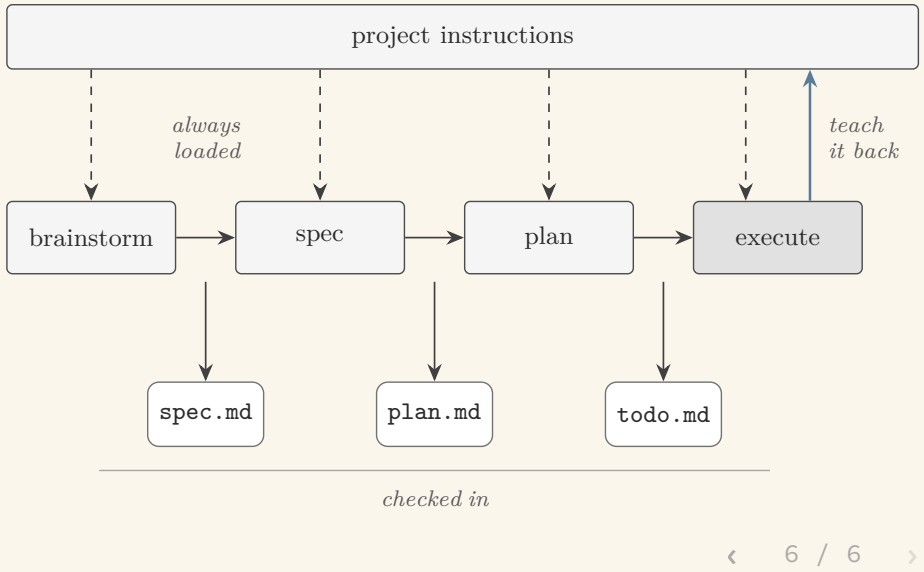


Figure from Chapter 22

Chapter 22, quoting a practitioner's workflow write-up: "Brainstorm spec, then plan a plan, then execute using LLM codegen. Discrete loops. Then magic." · the chapter: "A run is disposable. The lesson it taught you is not"

Fix the spec and regenerate; never patch the output and forget the spec

Spec-driven development

The practice

Specify → plan → tasks → implement → verify, with a **human gate** between phases. Acceptance criteria a machine can check, each wired to a **test**.

How it dies	What it looks like
Stale spec	an agent building, at machine speed, from a document that lies
False precision	a tidy summary that dropped the detail that mattered
Frozen handoff	waterfall again, with better fonts

Chapter 22, “Spec-Driven Development” · “You iterate on intent, never on its consequences.” · a practitioner’s aim: make spec drift “visible the same way a failing test is visible.”

The highest ratio of discipline to typed characters in the course

Tests first, one at a time

“First run the tests”

The agent learns there is a suite and how to run it, gets a sense of the project’s size, and treats the suite as part of the job.

Red, then green

Write the test, **watch it fail**, then make it pass. The fail-first step is the one to defend.

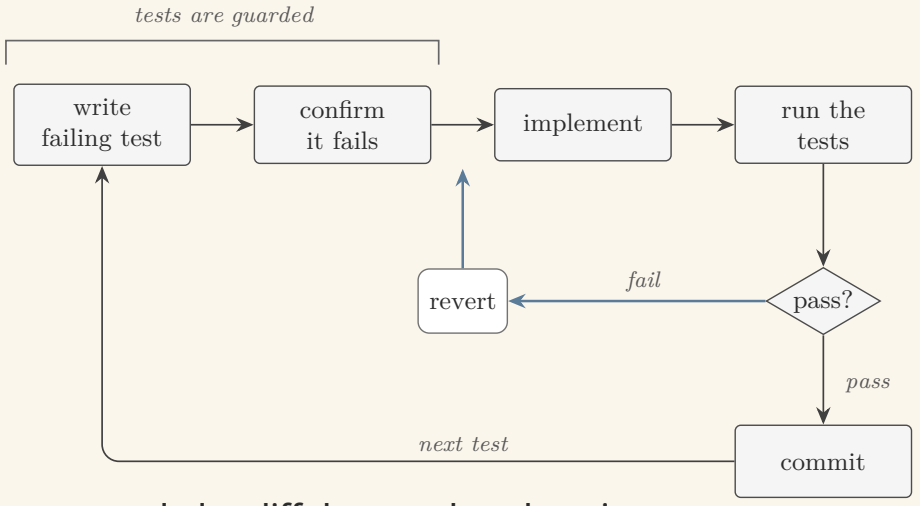
- The test proves the code works and that it was **wanted**
- One small test per step paces an eager generator: **the leash**

Chapter 22, “Test-Driven Development with Agents” · “a test you never saw fail has proven nothing” · “Red is the moment the verifier itself gets verified.”

An old discipline that turned out to be an agent guardrail

test && commit || revert

- Pass: **commit** the green state. Fail: **revert** the tree
- A bad try costs one discarded attempt, no archaeology
- Small steps, enforced by machinery
- Guard the checker: tests written or reviewed **apart** from the code; a diff touching a test is a **person's call**



The tests are guarded: a diff that touches them is a human's call.

‹ 5 / 5 ›

Figure from Chapter 22

Chapter 22, quoting the original TCR essay: "If you don't want a bunch of code wiped out then don't write a bunch of code between greens."

Savepoints and recorded proof

- A commit is a **savepoint**: every green state free to return to
- History is an **authored story**: squash the fumbles
- Size tasks as revertible units; a run far past its guess is a **tripwire**
- Make the agent **use** what it built: a one-liner, a demo, the real browser
- Every defect found becomes a **failing test**
- **Recorded execution**: output captured by the tool, never retyped by the model

Chapter 22 · “you can afford exactly as much boldness as your undo is boring.” · quoting a practitioner: “Never assume that code generated by an LLM works until that code has been executed.”

Every gate passed honestly, and the understanding is still gone

The debt nobody bills

Cognitive debt

The gap between what the system is and what anyone still understands. Accrues whenever work is accepted faster than it is understood, and **feels like competence** while it grows.

1. Walkthroughs with excerpts **pulled from source**
2. An **explainer packet** read before the diff: background, intuition, literate diff
3. A **quiz** you pass before anyone else sees the change
4. Tools you **operate** to learn the system

Chapter 22, “Understanding Code You Didn't Write” · a practitioner’s five questions, “difficult enough that you actually need to understand the substance of the PR to answer them, but not gotchas”

Part 3 · Chapter 26 (full book)

A classifier you can read

An apprentice, or an examiner

Train an apprentice

Label thousands of examples, train a model, retrain when the world drifts. What you get cannot explain itself.

Hire an examiner

A model already educated, who knows nothing of **your** categories. You write the **marking guide**. The finished classifier is a document.

- The examiner remembers nothing between sessions: everything local lives **in the guide**
- Architecturally, one call: no loop, no tools, until the task forces them

Chapter 26, “The Classifier Without a Training Set” · “the definitions are the model” · the chapter’s illustration of the two routes is in the book

Predict before the next slide · thirty seconds alone, then compare

“I was charged twice after the app crashed during checkout.”

billing or bug? Commit to one, then argue for the other.

Chapter 26, “The Classifier Without a Training Set” · the chapter’s own ticket

A class is defined at its edges

Anatomy of a definition

```
billing: the problem concerns money:
  charges, invoices, refunds, plans,
  payment methods.
  Includes: double charges, surprise
    renewals, failed payments, tax and
    receipt questions.
  Excludes: a payment-screen defect
    that charged nobody (bug); a plan
    that does not exist (feature-request).
  Boundary: payment error and defect
    together: decide by the remedy asked.
```

- **Essence**, **inclusions**, **exclusions**, **boundary rules**
- A fix runs through a **text editor**, not a retraining
- Output: a closed **enum**, enforced by the schema
- Give the menu an **exit**: other, and cannot-classify

Chapter 26 · after the chapter’s sketch of the billing class · “Every borderline you decline to legislate, the model will settle on your behalf”

In-context learning: the examples teach the examiner afresh on every call

A textbook, not a database

Role	What it does	Example
Prototype	the unmistakable center of a class	“You charged me twice this month” → billing
Edge	near a boundary, with the deciding feature named	paid for a year, app says the trial ended → billing
Contrastive pair	two near-identical inputs on opposite sides of one boundary	clicked Upgrade: charged twice → billing; froze, no charge → bug

- Each example with a **comment** saying why it is there
- Pairs where **measurement** shows the confusion, not where intuition guesses

Chapter 26, “Examples as a Textbook, Not a Database” · “Examples are a stencil.”

One schema change: the rubric questions come before the label

The worksheet before the verdict

Each answered in a sentence, from the ticket's text:

1. Does money move **wrongly**?
2. Is the product behaving **contrary to its design**?
3. What **remedy** does the customer ask for?

Then the label, then a short rationale citing the answers, and a **flag** when they pull apart.

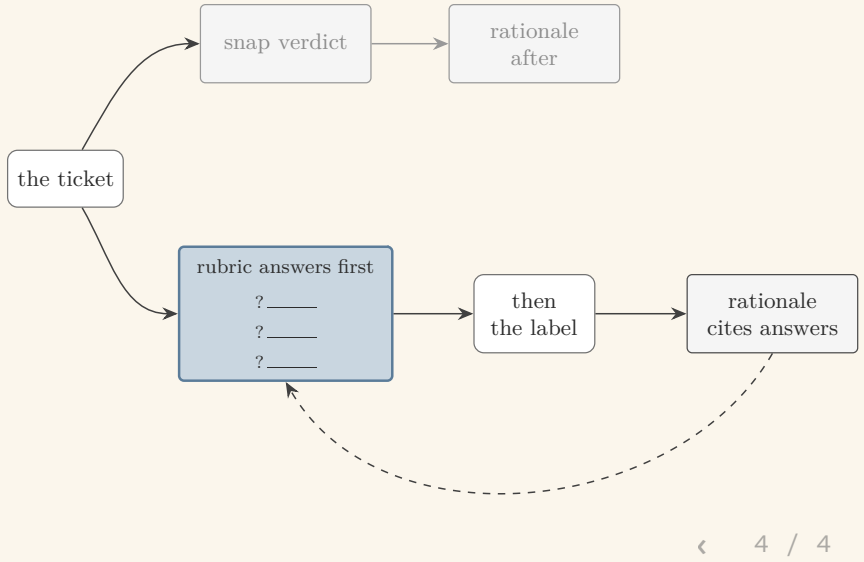


Figure from Chapter 26

Chapter 26, "Reason First, Then Explain" · "text written before the label can inform the label, while text written after it can only defend it"

The worksheet localizes a wrong label the way
a stack trace localizes a crash

Where a wrong label went wrong

The worksheet shows	Failure	Look at
an answer that contradicts the ticket	**evidence**	the input, or how the question is worded
right answers, wrong label	**definitions**	a boundary rule missing or misstated
an answer no rule anticipated	**rubric gap**	the questions themselves

- The **flag** routes the two-headed tickets to a person; the flagged stream is next month's revision
- The rationale is an **account**, not a printout of what happened inside

Chapter 26, “Reason First, Then Explain” · “you debug a document, with a map”

“Can it also score severity? Zero to one, two decimals.”

From labels to scores

- A bare number cannot be **wrong**, so it cannot be trusted
- A small **ordinal** scale: each level defined like a class
- Pairs between **adjacent** levels; rubric first; the flag
- No sentence between two levels? **Merge them**
- Resolution from **repeated draws**: 3, 3, 3, 2 → 2.75

```
4  service down or data lost for
   many; no workaround
3  a core workflow broken for some,
   or any confirmed mis-charge
2  a non-core feature misbehaving;
   a cheap workaround exists
1  cosmetic; no functional impact
Boundary: money handled wrongly
   is never below 3.
```

Chapter 26, “From Labels to Scores” · after the chapter’s severity scale · “a score you can trust is a classification wearing numbers”

Part 4 · Chapter 26 (full book)

Measurement you cannot fool

Predict before the next slide · one minute in pairs

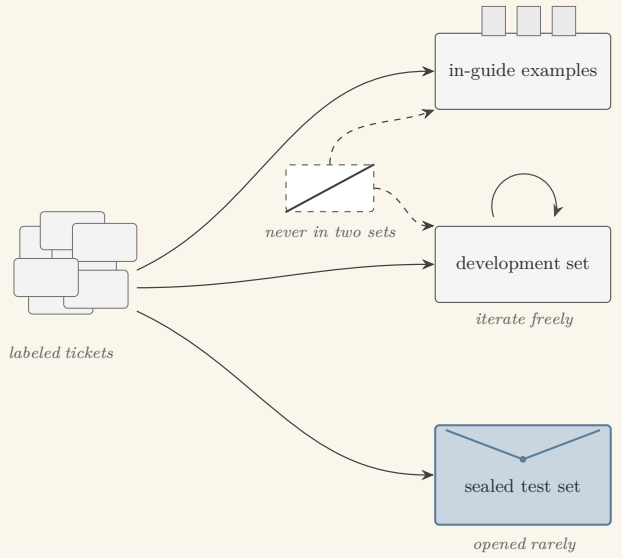
You write the guide with twenty real tickets open beside it. Run on those twenty, it gets nineteen right. You add one boundary clause for the twentieth: twenty out of twenty.

What does that number measure?
What would you do instead?

Chapter 26, “Developing It Without Fooling Yourself” · the chapter’s own story

Three sets, one envelope

- **In the guide**: the few demonstrations
- **Development**: re-run after every revision, read freely
- **Sealed test**: labeled once, opened at milestones, the only number reported
- No training set to feed: a **near-even** split
- Rotating folds show **stability**; they cannot make you forget
- Consulted too often? **Retire** it into development



‹ 5 / 5 ›

Figure from Chapter 26

Chapter 26 · “you have measured the textbook on its own worked examples” · “It is exactly as immune as you are.”

Week 8’s instruments, pointed at a classifier you are still writing

Counting it honestly

- **Kappa**, not accuracy, on a skewed mix
- Confusions read **in pairs**: a to-do list
- The **flag rate** beside the score
- Re-run development after **every** edit

The changelog

version · what changed · the failure that motivated it · dev score before · dev score after

Plus a line each time the envelope is opened.

Chapter 26 · the guide is read whole, so “no edit is local” · an instrument “can be made to look arbitrarily accurate by teaching it to flag everything hard”

The instrument is instructions and files:
package it as a skill

Ship it as a folder

```
ticket-triage/
  GUIDE.md       classes, boundaries,
                 rubric questions
  examples.md    prototypes, edges,
                 contrastive pairs
  schema.json    rubric answers, label,
                 flag, rationale
  CHANGELOG.md   revisions and scores
```

A fresh desk per item

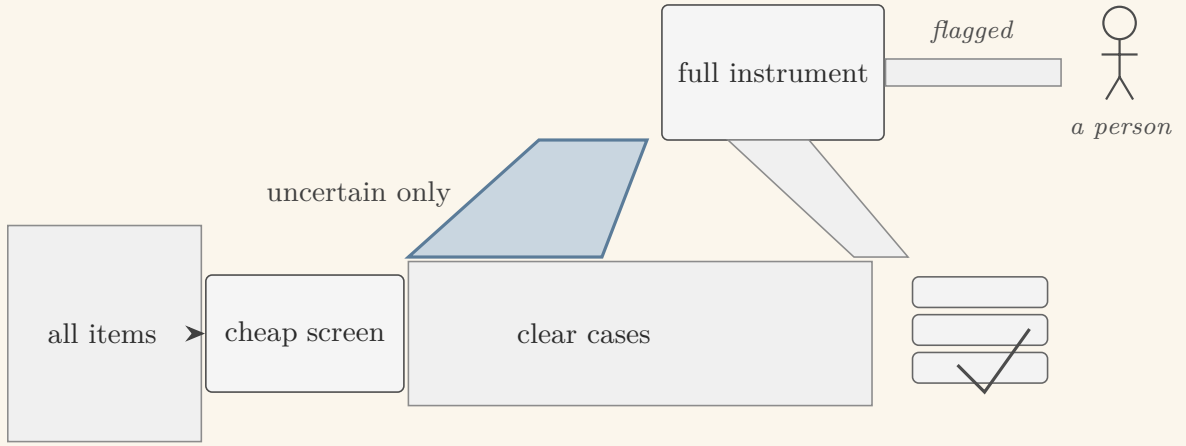
The call for ticket forty holds the guide and ticket forty. **None** of the thirty-nine verdicts before it.

Chapter 26, “Packaging It, and When to Reach for It” · a sketch, not a standard · “amnesia between items is what makes their verdicts comparable.”

Volume changes the economics before it changes anything else

Screen, then detail

- A **cheap screen** clears the obvious bulk
- It must know what it **may not decide**: the flag is the escalation trigger
- **Trained model** instead when categories are stable for years, volumes vast, cost floors low
- Or **distill**: audited verdicts become the training set



Only a thin flagged trickle reaches a person, and most items never touch the expensive call.

‹ 5 / 5 ›

Figure from Chapter 26

Chapter 26 · “The guide remains the source; the small model is a compiled build.”

In class · teams of two or three · 26 minutes · three browser tools

Split it, score it, price it

1. **A · the split (7 min).** The 200 tickets in the three-set splitter: counts, repeats, the envelope
2. **B · the revision (10 min).** The starter guide's development scores in the judge agreement calculator, before and after one revision
3. **C · the volume (6 min).** The cascade estimator at 50 and at 50,000 tickets a day

The tickets and the prefilled links are on the lecture page. Debrief, 3 minutes

Tools after Chapters 16 and 26 · each tool's readings are its own, labeled on the tool · the tickets and the starter guide are invented for the exercise

The optional lab, and reading

Optional: the triage classifier

- A **skill folder**: guide, examples, schema, changelog
- Every revision logged with **dev before and after**
- The sealed set opened **once**
- Replaces your lowest lab grade

Reading for week 13

- Ch. 23, Research and Business Agents `full book`
- Ch. 24, Agent UX and Human Trust `full book`
- Ch. 25, Transversal Recipes `full book`
- Ch. 27, The Frontier `full book`

This week’s reading: Ch. 21, Ch. 22 and Ch. 26 (full book) · glossary: App. B (free) · no paid API needed

Five things to keep

- Agents grow where a **verifier** already stands; elsewhere, build it first
- Own what the artifact's **lifespan** demands, decided before the first prompt
- Briefs down, **receipts** up: a spec, a red test, recorded output, a quiz
- The **definitions** are the model; the worksheet says which one failed
- Three sets, one **envelope**, every revision in the changelog

Engineering AI Agents · Week 12

The instrument is small. The habits are the whole craft.

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/

