

TRAINING

Hands-on AI-Assisted Development

IT track Day 2

Build tests with AI



Do not redistribute these materials

Redistribution of these materials is prohibited.

These materials are copyrighted by Givery, Inc. Use them only for your own learning and work reference. Do not share them outside your company, on social media or with third parties.

During the course, do not take screenshots or recordings that show these materials.

Today's goals

Today you test the Day 1 features **against the spec**

Get a fix from symptoms alone

Describe a one-line AI bug by its symptoms, not its cause. Have it fixed and check the screen.

Build a test case table with sources

List test viewpoints from the spec and build a test case table that traces expected results to requirements.

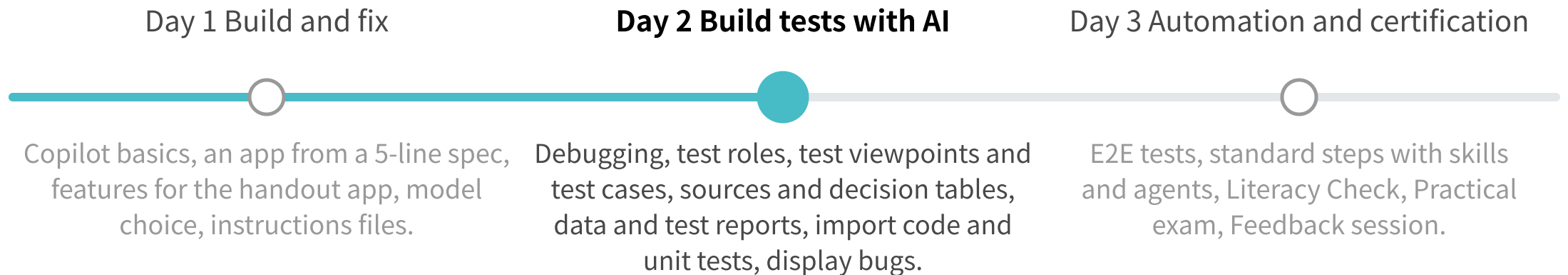
Judge tests written by AI

Read unit tests written by AI and explain in your own words what a pass means.

Today's subject is index.html, with the filtering and totals you added on Day 1.

Where today fits in the 3 days

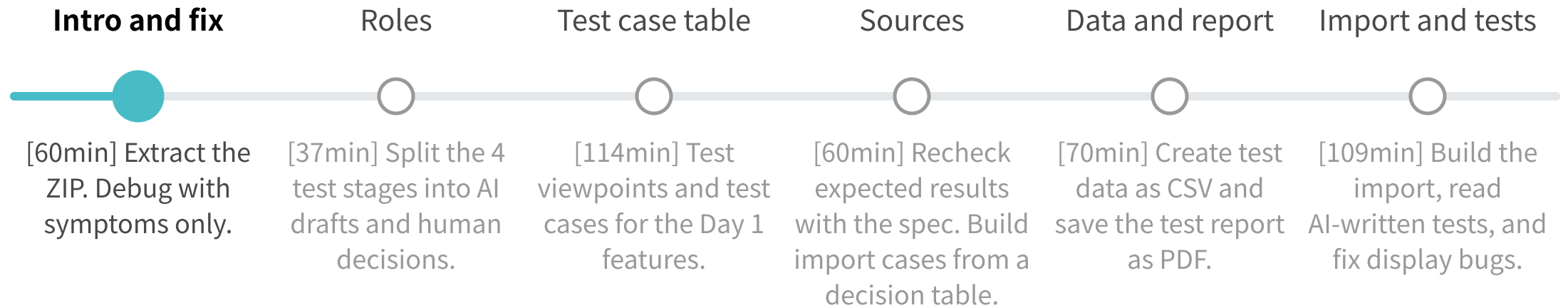
Today, reread the app you built on Day 1 **from test viewpoints**



Day 3 is 1 week away. You use today's files again on Day 3.

Today's schedule

Test cases this morning; **the specification** after lunch



Lunch break comes after Test case table. There are 3 [10min] breaks between the other blocks.

Exercise 1: Handout ZIP and where index.html goes

From today, use index.html **directly in tpro-work**

Scenario

On Day 1, you added features to index.html in the worklog-app folder. Today's handouts and the Day 3 materials expect index.html directly in tpro-work.

Steps

- 1 Extract the handout ZIP into the tpro-work folder and check that a day2 folder appears in tpro-work.
- 2 Copy index.html from the worklog-app folder and paste it directly into tpro-work. Keep the original file.
- 3 Reopen the tpro-work folder in VS Code. Open its index.html in the browser and check that the filter and the totals table appear.
- 4 If you have no Day 1 index.html, or filtering and totals do not work, copy day2/fallback/index_day1.html into tpro-work and rename it index.html.

Result

tpro-work now contains index.html, the day2 folder and the .github folder.



01

Debugging with AI

Have AI fix a one-line bug without telling it the cause

One comma in the initial value **changes the total's type**

In the line that computes the total, delete only the trailing , 0. The code differs from person to person, so follow the example on the instructor's screen.

```
// Original code (in tpro-work/index.html, the line that computes the total with reduce)
.reduce(function (sum, record) { return sum + parseFloat(record.hours); }, 0);
// Broken code: delete only the trailing , 0
.reduce(function (sum, record) { return sum + parseFloat(record.hours); });
// Without reduce, change the initial total 0 to '' to get the same symptom
```

What the broken screen looks like

The screen still shows values, so bugs are easy to miss

After deleting , 0

The totals table shows text and numbers joined, like [object Object]2.5. Filtering by category no longer changes the list, and a TypeError appears in the F12 Console.

With , 0

With the first 8 records, the totals are Design 5.5, Implementation 10.5, Review 2.5, Testing 5.5. The Console shows nothing.

When you can notice it

Only when you add up the totals yourself or open the Console. No error screen appears.

This is why you cross-checked the totals by hand in the Day 1 exercise.

Exercise 2: Break one line

Break **only the given line** in your index.html

Scenario

If you accept AI code without reading it, one-character errors like this slip in. See once in your own file how the screen looks when that happens.

Steps

- 1 Copy index.html and keep the copy as index_ok2.html.
- 2 Open index.html in VS Code, find reduce with Ctrl+F, and delete the , 0 at the end of the total line. No reduce? Change the initial total 0 to ''.
- 3 Save with Ctrl+S, reload the browser with Ctrl+Shift+R, and check the totals.
- 4 Change the category filter. Press F12 and look for red lines in the Console tab.

Result

No deliverable. You are done when the totals show a value starting with [object Object] or numbers joined together.

Completion criteria for the break

See **both symptoms** before the fix exercise

Done when

- A total starts with [object Object] or shows numbers joined together.
- The filter does not change the list, or the Console shows a TypeError.
- index_ok2.html is in tpro-work.

If you're stuck

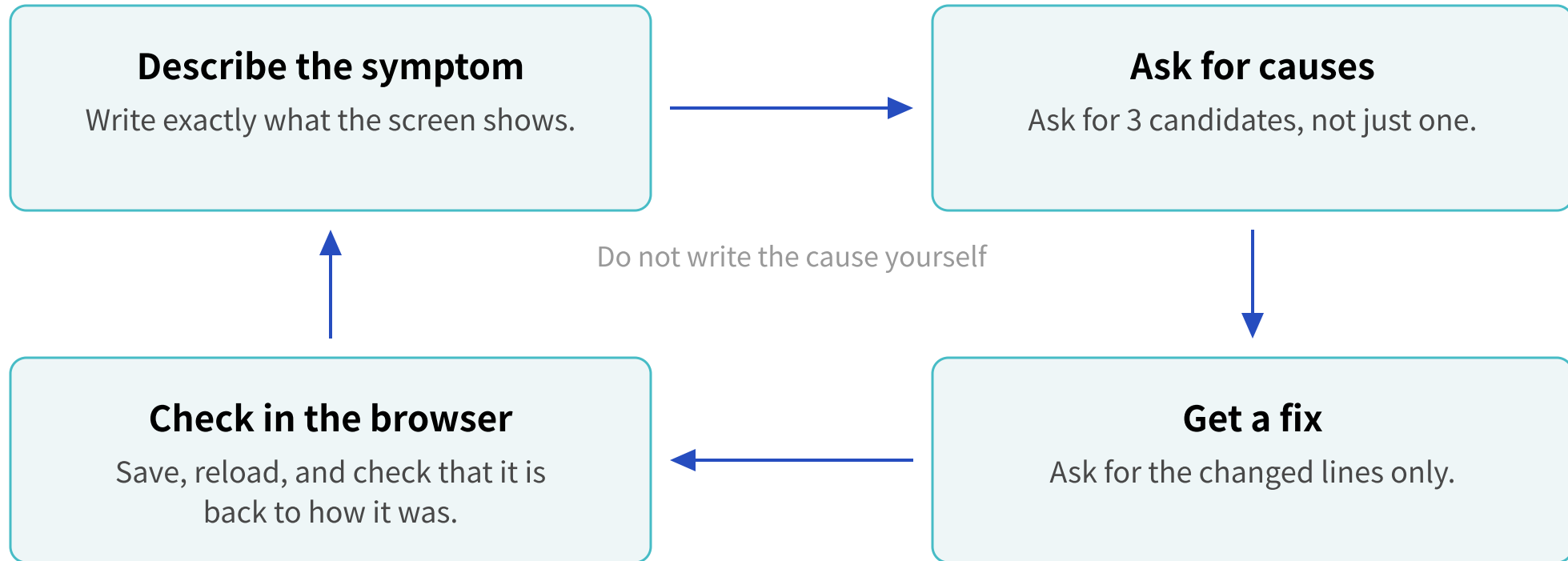
- Nothing changes. Save with Ctrl+S, then press Ctrl+Shift+R in the browser.
- The screen goes blank. You deleted a bracket too. Copy index_ok2.html back to index.html and delete only , 0.

If time allows

- Copy the TypeError text from the Console into prompt.txt as is. You decide whether to paste it in the next exercise.

4 steps of debugging

Skip step 4 and you move on thinking it is fixed



If you know the cause, stating it is faster. But if your guess is wrong, the AI goes wrong too.

Write **only what happens**, not a word about the cause

With the code still broken, send these 5 lines in Agent chat. Save them in prompt.txt to reuse.

```
I am running a single HTML file, index.html, by opening it directly in the browser.  
The totals table by category shows values like [object Object]2.5.  
Filtering by category does not change the list, and the Console shows a TypeError.  
List 3 possible causes, then show only the lines you changed.  
Do not add external libraries.
```

Give only the symptoms, get a fix, and read why it works

Scenario

Bug reports from clients rarely state the cause. Starting from only what the screen shows, you do the first round of listing and ruling out causes with AI.

Steps

- 1 With the code still broken, send the 5 lines from the previous page in Agent chat.
- 2 Read the 3 possible causes and pick the one you think is right.
- 3 Read the diff, click Keep, save with Ctrl+S and reload the browser.
- 4 Check the totals against your hand sums. Change the filter and see no Console errors.
- 5 Add the 5 lines you sent and the cause explanation to prompt.txt and save.

Result

index.html is back to how it was, and prompt.txt has the 5 lines you sent and the cause explanation.

Totals are numbers again and **Day 1 features still work**

Done when

- The totals match your hand sums.
- Changing the filter changes the list and totals, and the Console shows nothing.
- prompt.txt has the 5 lines you sent and the cause explanation.

If you're stuck

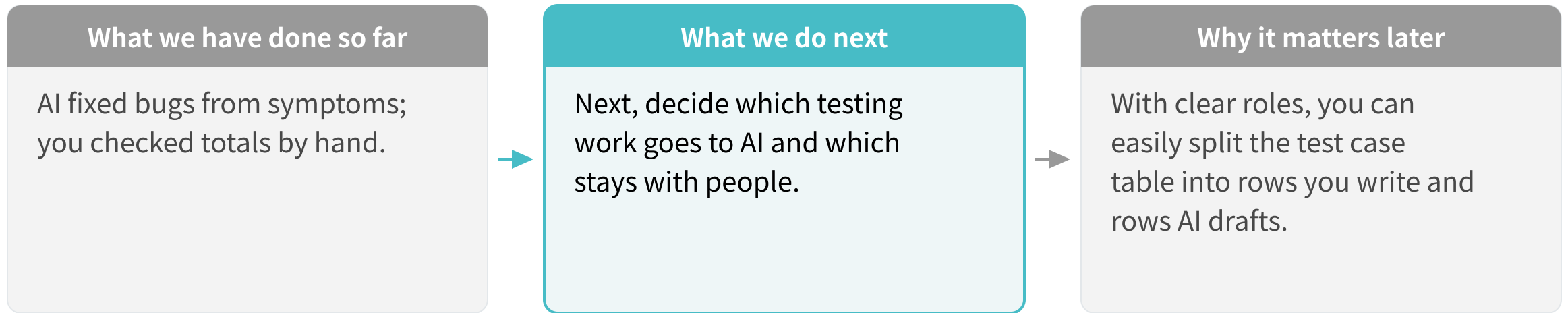
- A total is blank. Send one sentence: "Show 0 for categories with no records."
- It still fails after several tries. Copy index_ok2.html back as index.html and try once more.

If time allows

- State the cause up front and compare how the answer changes.
- Clean up prompt.txt to keep only 3 prompts that worked on Day 1 and today.

From debugging to testing

Checking every fix by hand **has limits**

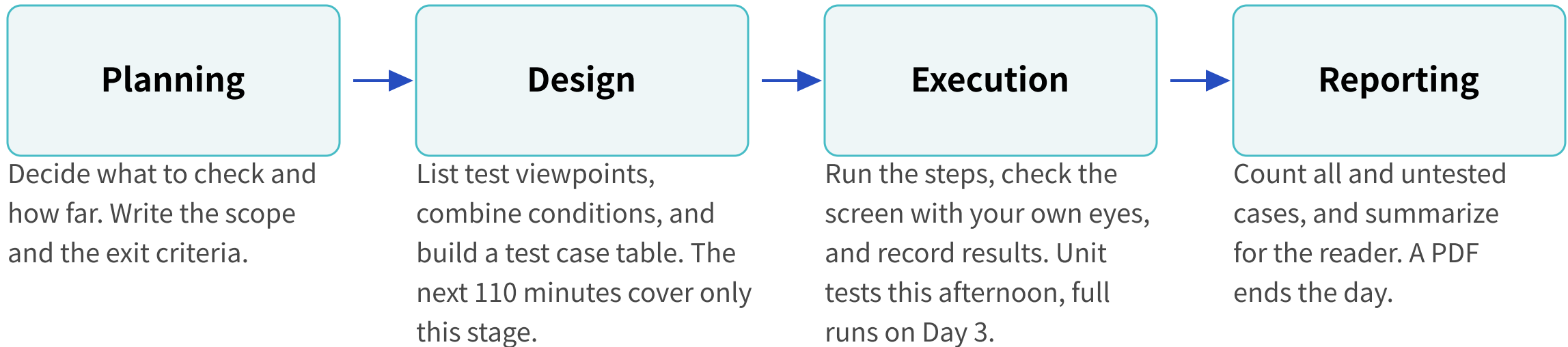


02

Sharing test work with AI

Learn what AI drafts and what you decide in each of the 4 stages

Of the 4 stages, the next part covers **only design**



Unit tests and integration tests both follow these 4 stages in order. Just remember the stage names.

In every stage, AI only gives **candidates**

Planning

Lists scope candidates from past bug trends. It can also point out missing test viewpoints.

Design

Test viewpoints, condition combinations, draft test case tables. 30 drafts in 2 minutes.

Execution

Writing steps, formatting results, writing repro steps. Turns tester notes into readable text.

Reporting

Counts, tables, wording. Tallies results by priority in a table.

All of these look the same on screen even when they are wrong.

Same 4 stages. **Decisions stay with people**

Planning

Where to draw the line for release. Deciding to narrow the scope, and recording why.

Design

Reading the spec, priorities, which cases to drop. Decide what to cut from the 30 drafts.

Execution

Running it and checking the screen yourself, and judging if the environment is valid. People do this today and on Day 3.

Reporting

Bug severity, go or no-go for release, who to tell and how. Only people who know the business impact can decide.

Side by side, every stage splits into 2 parts: "draft candidates" and "decide".

Delegate only **what you can sign with your name**

**"AI guaranteed the test coverage."
No test report can say that.**

When you explain missing cases, "the AI wrote them" is no excuse. Delegate to AI only what you can put your name on. Every judgment you hand over is one you cannot explain.

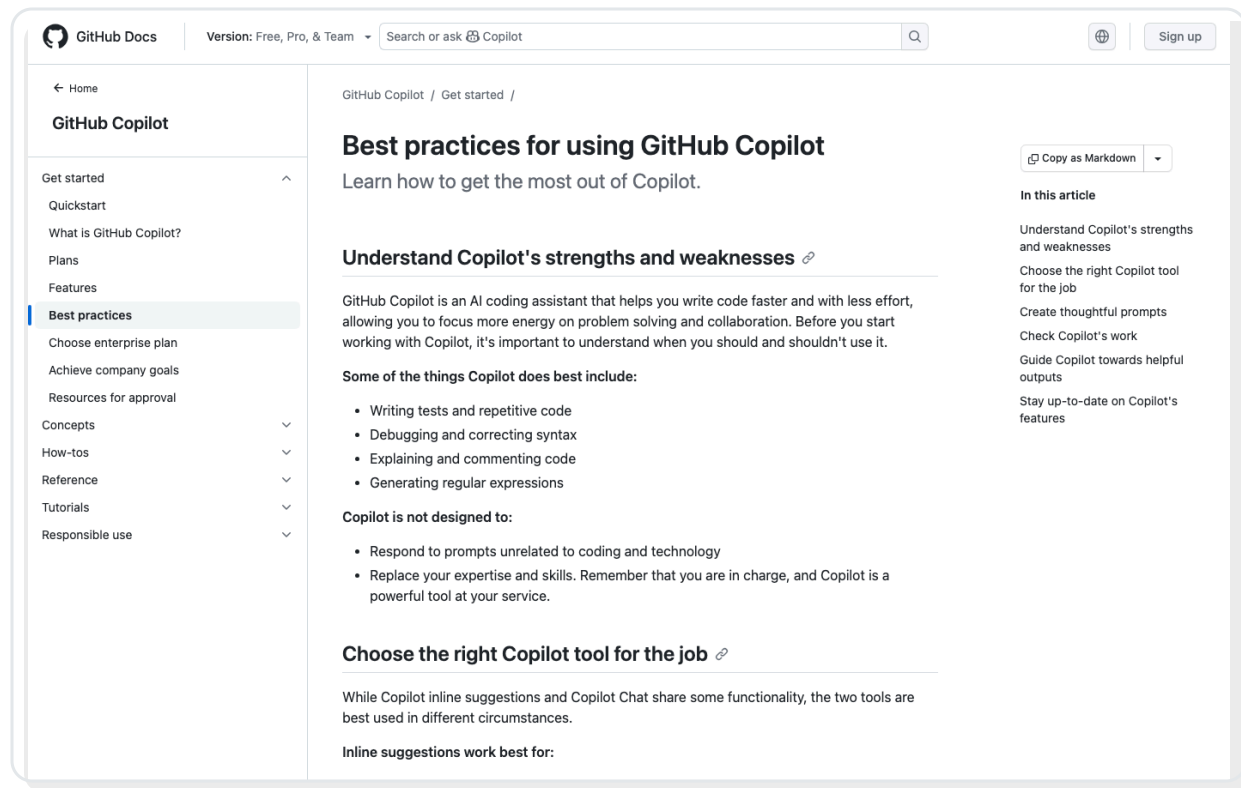
One line banning extras controls the output size

If AI outputs 100 cases at once, you will not review them one by one. The first step to splitting the output is a one-line ban on what not to write. Only line 2 does the work.

```
List test viewpoints for the hours input check in the Worklog app.  
Do not write test cases, expected results or operation steps.
```

After the break, you get the full prompt: these 2 lines plus target, count and format.

Follow these 2 points in today's exercises



GitHub's official best practices page lists: review output before you use it, and split requests into small tasks. It is updated, so check the current page before sharing it at your company.

Break

After the break, build a **test case table** for Day 1

After the break

Keep tpro-work open in VS Code.

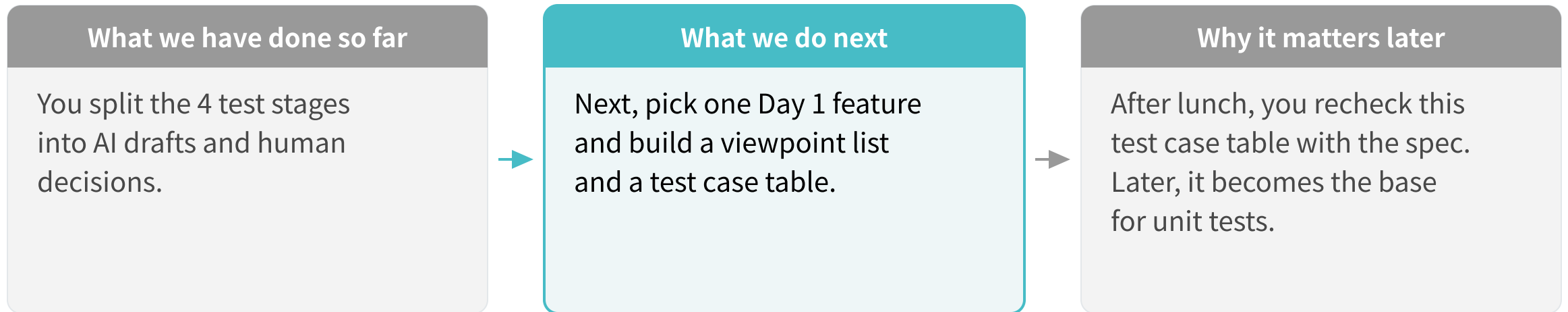
Next section

Build a viewpoint list and a test case table. After the talk, you have lots of time to work on them.

We resume in 10 minutes.

From roles to design

Now you work through **the design stage** by hand

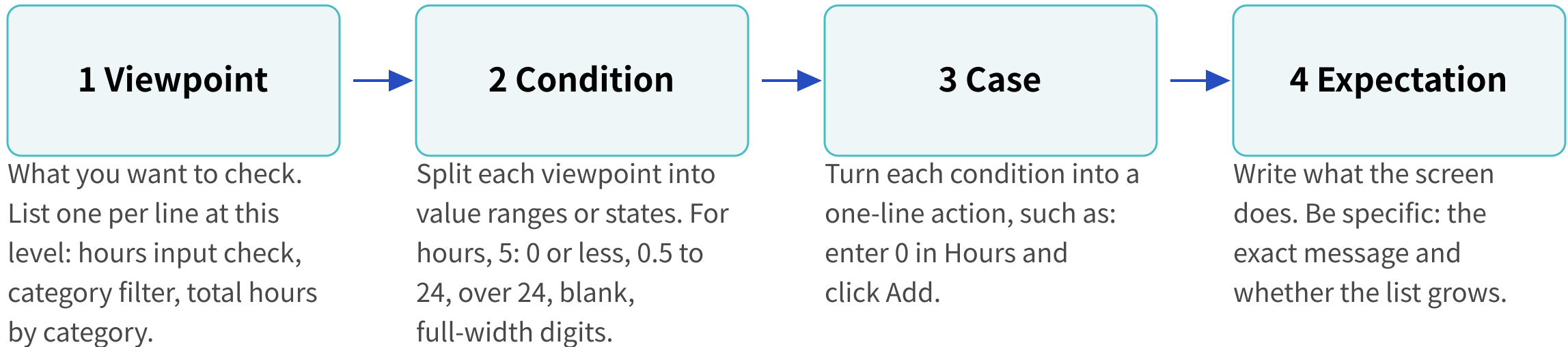


03

Test viewpoints and test cases

Learn to go from test viewpoints to expected results in 4 stages and run the gap-filling cycle twice

Go from viewpoints to test cases in 4 stages, adding detail



If you skip stages and ask at once, the table mixes whole-screen and single-button viewpoints.

Take Hours in the Worklog app through stages 1 to 4

1 Viewpoint

Hours input check

2 Condition

5 classes: 0 or less / 0.5 to 24 / over 24 / blank / full-width digits

3 Case

Enter 0 in Hours and click Add

4 Expectation

Under Hours: "Enter a value from 0.5 to 24". The list does not grow

Do not target the whole screen. Narrow it to one input field or one dropdown.

Viewpoints from the spec alone miss actions on the screen

From the spec only

Paste the spec into the chat and get test viewpoints as a list. It is fast, but fields missing from the spec and buttons that can still be clicked do not appear. Spec gaps become test gaps.

From both the spec and the screen

Paste the spec memo and the HTML of the running screen, then ask for every control you can operate. Actions not in the spec appear here. Merge duplicate viewpoints later.

Unless you write "no table", you get a test case table at once

This is the prohibition line shown before the break, plus target, count and format. Limit the target to one item and cap the count. Without both, you get 30 viewpoints for the whole screen.

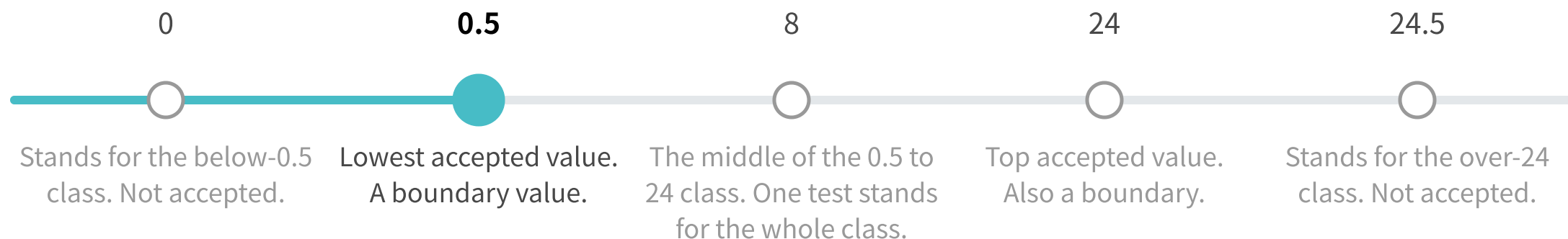
List only the test viewpoints for this part of the web app below, as bullets.

- The target is only the Hours input field and the Add button
- Do not make a table. Write only the viewpoint names, one per line
- List 10 or fewer
- Write each viewpoint name as a short noun phrase, like "XX input check"

(Paste the HTML of the target part below this line)

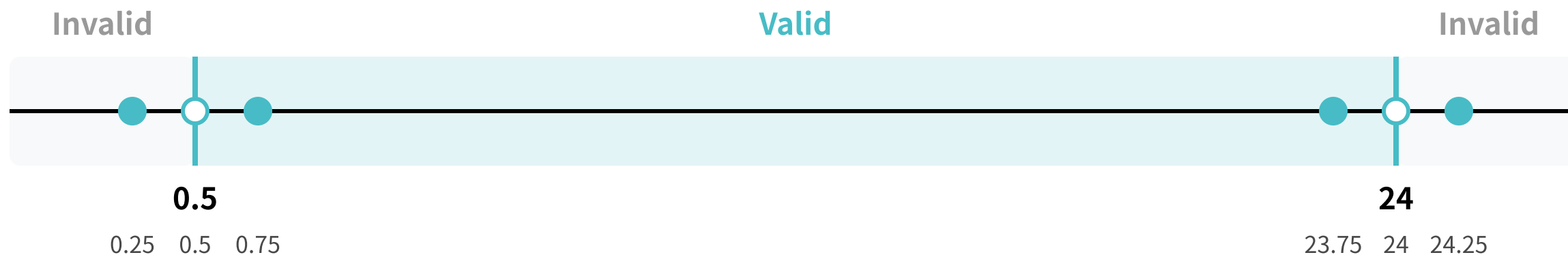
Do not use the viewpoints as they are. Add 2 of your own before the next stage.

Equivalence class: a **group** with one result. Boundary: its edge



Values off the line are equivalence classes too: blank, full-width digits and negative numbers.

Testing **one value on each side of the edge** is enough



The valid range of Hours is 0.5 to 24

Every value in this range is treated the same. Testing both 10 and 15 tells you nothing more

Test 6 values

There are 2 edges: 0.5 and 24. Test each edge and 15 minutes outside and inside it: 0.25 / 0.5 / 0.75 and 23.75 / 24 / 24.25

Split by field to see the number of cases in advance

Field	Equivalence classes	Boundary values to test
Hours (0.5 to 24)	In range / below range / above range / not a number	0, 0.5, 24, 24.5
Description (max length set by spec)	Has text / empty / over the limit	Empty, 1 char, limit, limit + 1
Date	Today or earlier / future date / wrong format	Yesterday/today/tomorrow
Category dropdown	One category / All	Switching back to All

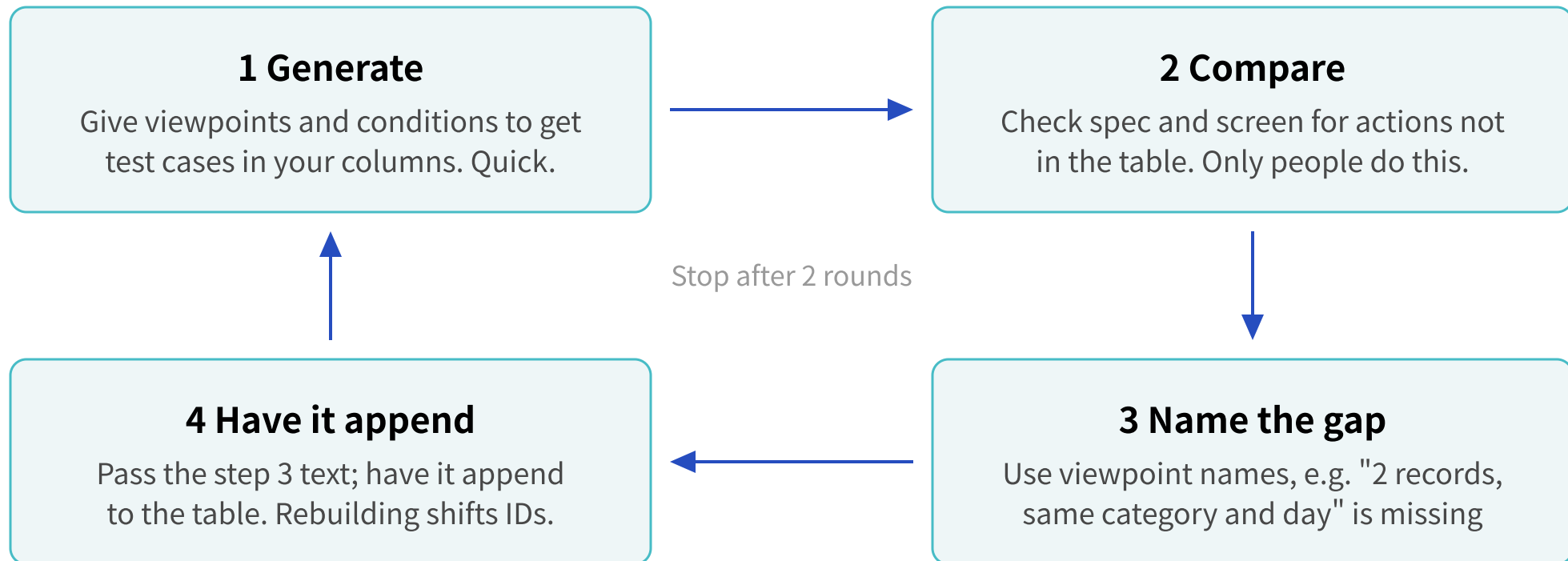
Do not make every combination. Change one field at a time and combine only related fields.

Fix the columns first, or the AI changes the format each time

Copy this header row as is into the chat. The exercise uses the same row.

```
| ID | Test viewpoint | Precondition | Action | Expected result | Priority |  
|---|---|---|---|---|---|  
| C-01 | Hours input check | Initial data shows in the list | Enter 0 in Hours, click Add |  
"Enter a value from 0.5 to 24" appears. The list does not grow | High |  
| C-02 | Hours input check | Same as above | Enter 0.5 in Hours, click Add | The list grows by 1  
and the input fields clear | High |  
| C-03 | Hours input check | Same as above | Leave Hours empty, click Add | "Please enter a  
value" appears and the cursor moves to Hours | Medium |  
| C-04 | Category filter | Initial data shows in the list | Set Category to "Implementation" |  
Only Implementation rows remain. Back to All restores the count | High |
```

The first output is a draft. Fill the gaps over **2 rounds**



When you go from 4 back to 1, do not say "redo it". Write "append to the existing table".

Exercise 4: Viewpoint list and test case table

Pick one Day 1 feature and write its viewpoints and test cases

Scenario

Your first test design job at a client site is usually part of a screen someone else built. The spec is incomplete and you only have the running screen. From there, find test viewpoints and write a test case table others can run. Do it once with the Worklog app.

Steps

- 1 Pick one target. From the handout app you extended on Day 1, choose one: category filter, total hours by category, or hours input check. Not the whole screen.
- 2 Paste the HTML of the target into the chat and use the prompt from the "Prompt for test viewpoints" page to get viewpoints only. Add 2 of your own.
- 3 For each viewpoint, ask for 3 columns: field, equivalence class, boundary value.

Put only the handout app's content into the chat. No client names, product names or real data.

Exercise 4: Viewpoint list and test case table (continued)

Steps

- 1 Copy the header row from the "Test case table columns" page as is and get a Markdown table. Save it as testcases.md in the tpro-work folder on your desktop.
- 2 Run the cycle twice: find a missing action, write one sentence, have it appended.
- 3 In prompt-memo.txt in the tpro-work folder on your desktop, write one line on which sentence you think worked.

Result

testcases.md (test case table) and prompt-memo.txt (one-line memo) in tpro-work on your desktop. You do not run the cases yet; that comes in later chapters and on Day 3.

Completion criteria for the test case table

Your added rows and checkable expected results, not count

Done when

- testcases.md has 10+ cases, and 2+ of them are rows you added after spotting a gap
- Every row has a precondition, action and expected result, and the expected result says what the screen shows ("works correctly" is not OK)
- Not every priority is "High". Only cases that stop the work if they fail are High

If you're stuck

- 30 viewpoints came back: the target is too broad. Narrow to one field or dropdown.
- Your Day 1 feature is unfinished. You can target it up to where the filter field shows. Base expected results on the spec you decide.
- You asked for viewpoints only but got a test case table. Add "viewpoints only, as bullets, no table".
- The Markdown table looks broken. Press Ctrl+Shift+V to open the preview and see it formatted.
- Only 5 cases came out. Go back to the conditions and use the boundary value column.

Completion criteria for the test case table (continued)

If time allows

- Add 3 cases that add a record, delete it, then add the same record again.
- Make a separate table: a run order list of the High priority rows only.
- Add 2 cases that combine the category filter and the hours input check.

Gaps the AI misses are in **order and prior state**

What the AI gives in one pass

Independent cases, one action each. The HTML you pass is one screen frozen in time, so it holds no passage of time or order of actions. That is why they do not appear.

What it gives when you supply the angle

Add then delete; open the next with old input still there; use 2 tabs in turn. Describe order and state in words, and the AI can write the rows.

Takeaway from these 110 minutes

Good test cases: **checkable** expected results, not rows

The AI offers candidates You write the pass criteria

In 110 minutes you went back and forth: get candidates fast, then rework them so others can run them.
It was not about adding rows.

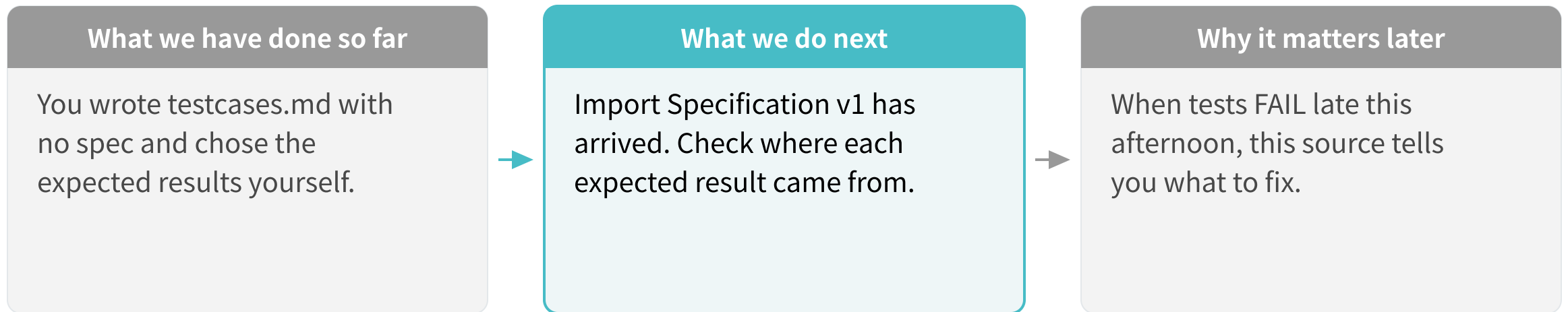
04

Test design basics

Review today's test case table against the new spec, and derive import cases from a decision table

A test case table without a spec

Your expected results so far came from **your own guesses**



Passing and **meeting the spec** are different things

Expected values copied from the code

Tests written from the code use the function's output as the expected value. The broken total from the debug exercise would pass too.

Expected values set from the spec

Expected values come from the Import Specification. If the code differs, the test FAILs, and then you know what to fix.

4 sources of expected results

Only **the spec and acceptance criteria** count as evidence

Sources	Counts as evidence for the spec	Value in the Source column
Specification	Yes	Spec (R-number)
Acceptance criteria	Yes, once the client approves	Acceptance criteria (AC-number)
Implementation	No. Wait until the spec confirms it	HOLD
Guess	No	Guess

Keep Guess and HOLD rows and make them tests too. Triage them once they show as FAIL.

Rows you can check with one function [go to unit tests](#)

Layer	How to check	In your cases
Unit	Pass values to a function, compare the result	Reading hours, totals by category
E2E	Open the screen, use it, compare the display	Switching filters, the list after import
Manual	A person judges wording and appearance	Whether warning reasons read clearly

In the Layer column of testcases.md, enter Unit, E2E or Manual

Before opening the spec, note one expected result's source

1

Create decision-log.md

Create it directly under tpro-work. Write all of today's decisions here.

2

Line 1

Pick one expected result from testcases.md and write where that value came from.

3

Line 2

Write whose word would change that value.

Do not open the spec yet.

First, mark the requirements your test case table covers

Requirement	What it defines
R-01	How import accepts files, encoding, header columns
R-02 to R-05	Per-row checks: date, hours, category, description
R-06 to R-08	Relations between rows: duplicates, daily total, holidays
R-09	How warnings appear, and the warning code table
R-10	How totals by category are calculated
R-11	Export format
R-12	Existing behavior and manual entry

Exercise 5: Map expected results to requirement IDs

The AI drafts the mapping; **you check it yourself**

Scenario

Make a table of which of the 12 requirements each expected result in testcases.md comes from. Do not edit files at this stage.

Steps

- 1 Give Ask #testcases.md and #import-spec_v1.md, and get the mapping in 3 columns: ID, requirement ID, source sentence [3min]
- 2 Pick 6 rows from the mapping, read the matching spec sentences yourself and compare. Mark rows the AI got wrong [4min]

Result

Results for the 6 checked rows in decision-log.md, used as columns on the next page

Prompt pattern to add Source and Layer columns

Give **the results you checked yourself** as the values

A prompt pattern for Agent. Fill in the <> parts yourself.

```
Add <2 column names to add> to testcases.md.  
Source is one of <4 values>, and Layer is one of <3 values>.  
Fill in the values as in this list: <your checked results>  
Do not change <columns to keep>.
```

If the diff changes the expected result column, do not click Keep.

Exercise 5: Three rows that conflict with the spec

Do not fix the expected results of conflicting rows yet

Scenario

Whatever target you chose, today's test case table conflicts with the spec somewhere. Whether to fix the code or the test is decided only after you run it against the code.

Steps

- 1 Find 3 rows that conflict with the spec. In decision-log.md, write one line each: case ID, requirement ID, which is right. Set Source to Guess [3min]
- 2 Have Ask list only what import-spec_v1.md leaves out and what can be read two ways [3min]

Result

3 conflict lines in decision-log.md, and the list of gaps in the spec (Ask's answer)

Ask so you can implement **whichever way it is decided**

Wording that leads to more questions

You write "Please make the spec more detailed."
The client does not know what to decide, and work stops until they reply.

Wording with two options

Add the requirement ID and ask them to choose A or B. You can implement either way, so you keep working while you wait.

Done when: rechecking sources

Done when

- Every row in testcases.md has a Source and a Layer, and Spec rows have a requirement ID
- The expected result column has not changed by a single character since the recheck
- decision-log.md has 3 lines on rows that conflict with the spec, with requirement IDs
- open-questions.md has 3 questions for the client, each with two options
- decision-log.md gives the reasons for the 2 rows you left out

If you're stuck

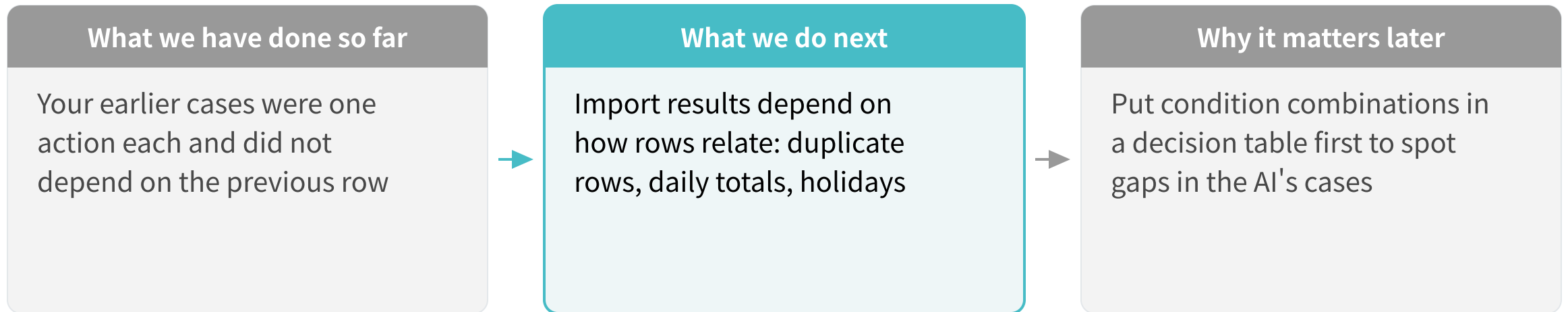
- Ask labeled every row Spec → Compare the source sentence column with the expected results. Rows that do not match were mapped wrong
- Expected results changed too → Undo the Keep and resend, saying only to add columns
- A question says "please give more detail" → Rewrite it so the client picks A or B

If time allows

- Turn open-questions.md into a draft email of questions to the client. Keep the addressee and company fictional: 5 lines with requirement IDs and two options

From single-action cases to cases across rows

In import, **the surrounding rows** change the result



Exercise 6: Six conditions for import

Before you ask the AI, **list the conditions and values yourself**

Scenario

Six conditions change the import result: date format; hours format, range and step; category name; empty description; duplicate rows; holidays and the approval marker.

Steps

- 1 Add a Conditions heading in decision-log.md; write the 6 conditions, one per line [1min]
- 2 On each line, list the possible values from the specification [4min]

Result

The 6 condition lines in decision-log.md, with possible values

You do not need **every combination**

A prompt pattern for Ask. Fill in the < > parts yourself.

```
Make a decision table for CSV import with the following <number of conditions> conditions.  
Each result must be one of <3 results>. Put the supporting <requirement ID> in a column.  
Holidays are weekends and the dates in <holiday file>.  
Set combinations that the spec does not decide to <value when undecided>.  
<the 6 condition lines you wrote>
```

Vary one condition at a time. Combine only two: duplicate rows and the daily total

Exercise 6: Cases from the decision table

Expected results are **backed by requirement IDs**

Scenario

Add decision table cases to Import in testcases.md. Check expected results in the spec.

Steps

- 1 Have Agent add the decision table to the Import section of testcases.md from I-01, read the diff and click Keep [3min]
- 2 Pick 5 expected results and check the requirement ID and spec sentence yourself [4min]

Result

The Import section of tpro-work/testcases.md, from I-01

Exercise 6: Three cases for order and state

You decide which drop and in what state to test

Scenario

The decision table covers only one import. It cannot produce these 3: dropping the same CSV twice, dropping after adding a record by hand, and dropping while filtered.

Steps

- 1 For each of the 3, write the precondition and action yourself, one line each [4min]
- 2 Have Agent add them to the end of the Import section, with expected results from the spec requirements, then click Keep [2min]

Result

The 3 rows you wrote, in the Import section of testcases.md

Acceptance criteria are not spec **until the client approves them**

Acceptance criteria you cannot judge

You write "imports correctly." With no count to accept, it is neither pass nor fail

Acceptance criteria judged by count

Write whether you judge by records imported, records not imported or warnings. You need not know the numbers yet; write how to count

Done when: import cases

Done when

- The decision table result column has requirement IDs; undecided combinations are PENDING
- The Import section has cases from I-01, including the 3 rows you wrote
- Expected results are written as counts or as on-screen text
- open-questions.md has 3 acceptance criteria under Awaiting approval, each judged by count
- Rows from acceptance criteria have Source set to Acceptance criteria (AC-n)

If you're stuck

- The decision table got too big → Vary one condition at a time; combine only 2 related ones
- Days over 24 hours in total are marked Not imported → Have the AI reread R-07
- Holidays are judged by weekends only → Resend with #holidays_2026.csv attached
- Acceptance criteria say "imports correctly" → Rewrite them in terms of what to count

If time allows

- Write cases for a name missing from category-map.csv in 2 ways: not imported as the spec says, and added to the map later by the client

Break

After the break, make **data and a test report** for the test cases

After the break

Check that testcases.md and decision-log.md are directly in tpro-work.

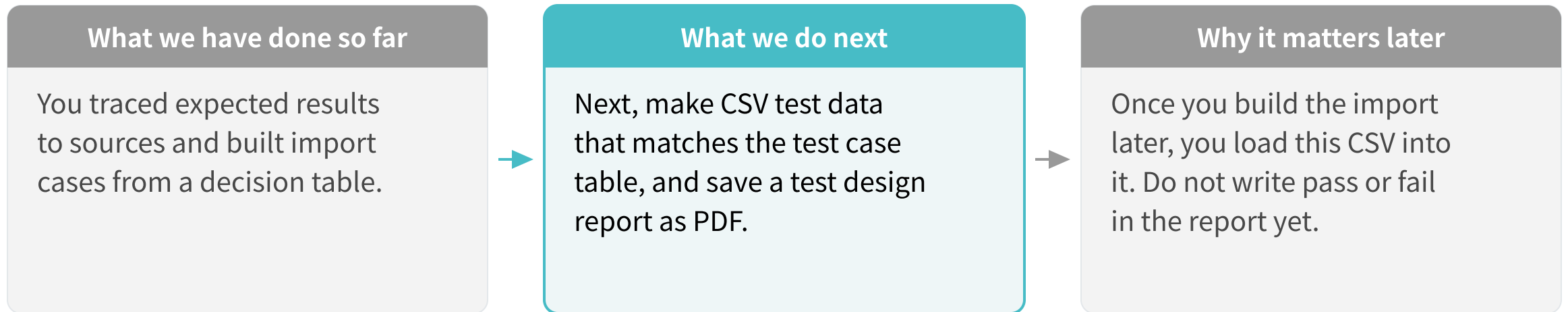
Next section

Create test data as CSV and save the test report as PDF.

We resume in 10 minutes.

From sources to data and test reports

Before running test cases, prepare **the data and report format**



05

Test data and test reports

Create CSV with set counts and distribution, and print the test report to PDF from the browser

Without **counts and distribution**, you get look-alike rows

Give the test case table first, then ask. Match the columns to the handout app's internal field names.

```
Create 30 rows of CSV input data for the following test case table.
```

```
Conditions:
```

- Use 4 columns: date,kind,memo,hours
- Split the rows into 20 normal cases, 5 boundary values and 5 error cases
- Include one row each with hours 0, 0.5, 8 and 24
- Use only these 4 values for kind: Design / Implementation / Review / Testing
- Make memo a fictional task description of 20 characters or fewer
- Use YYYY-MM-DD for date
- Do not use real company or project names
- Make the first row a header, and output only a code block

The 3 terms split equivalence classes into valid and invalid

Normal case

A value near the middle of a valid equivalence class. For hours, 8

Boundary value

A value at the edge of a class. For hours, 0.5 and 24

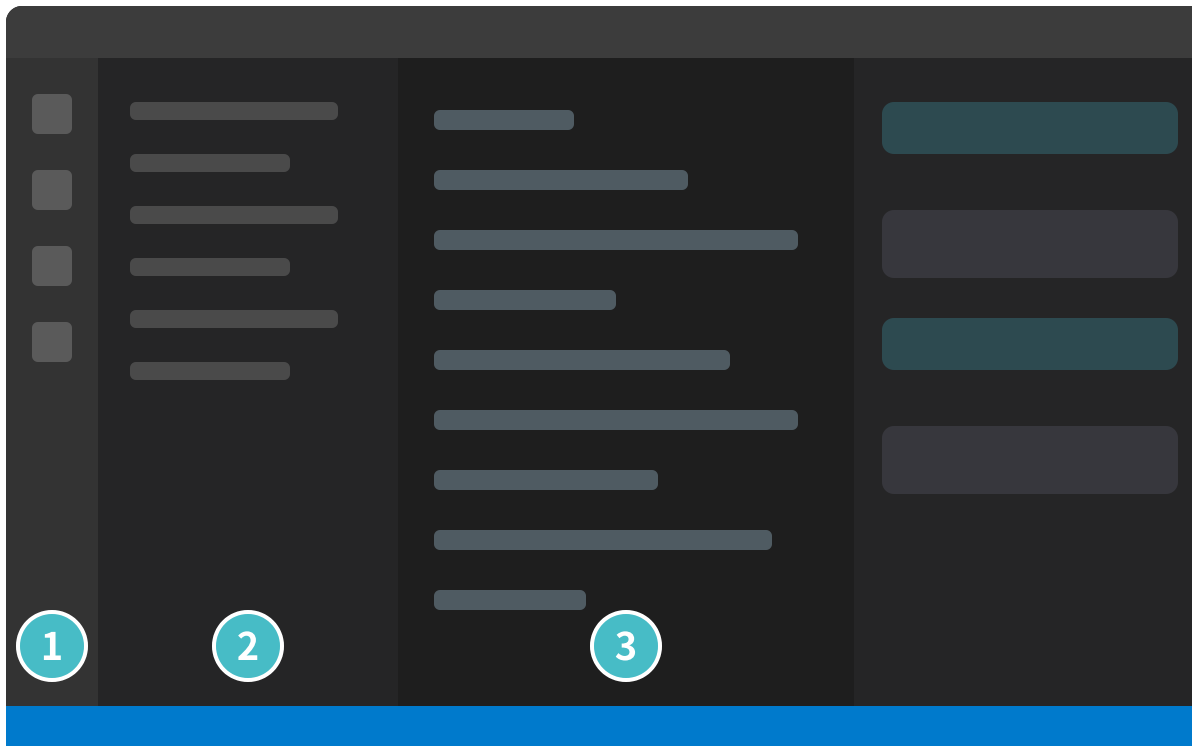
Error case

A value in an invalid class. For hours, 0, 24.5, blank or full-width digits

Without a breakdown, you get bland data with only normal cases.

Saving the CSV and counting rows

Put the CSV in **the tpro-work folder** and count rows in VS Code



1 Activity Bar

Switch between Explorer and Chat here. You go back and forth while you work

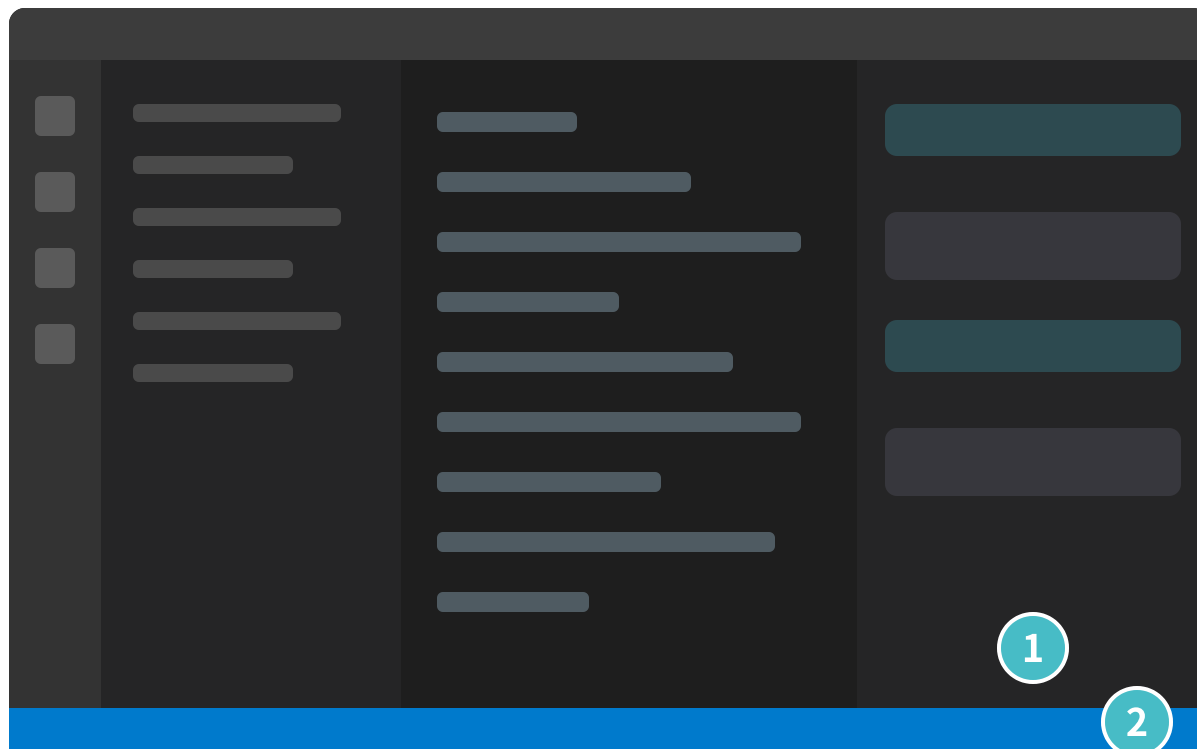
2 Explorer

data.csv appears in the tpro-work folder. If it is not here, you saved it to another folder

3 Editor

Open data.csv to see line numbers at left. Check for 31: 1 header + 30 data rows

Saving the CSV and counting rows (continued)



1 Chat view

If rows are missing, continue in this box: "3 error case rows are missing. Please add them."

2 Status Bar

The cursor's line number shows at right. Put the cursor on the last line to read the count

Exercise 7: Generate test data

Make 30 CSV rows for test cases; **count the breakdown yourself**

Scenario

When people make test data by hand, all 30 rows end up as normal cases. Boundary rows are tedious, get put off and are missed in the end. Try once, with the handout app columns, a prompt that sets counts and distribution first.

Steps

- 1 Copy the prompt from the "Prompt for test data" slide into the chat. If your earlier thread still has the test case table, continue in it
- 2 Adjust the hours condition to your test case table. If any of 0, 0.5, 8 or 24 does not appear in your table, add it
- 3 Copy the returned CSV, create a new file in VS Code, and save it as data.csv in the tpro-work folder on your desktop

Exercise 7: Generate test data (continued)

Steps

- 1 Count the lines. If not 31, name the missing part of the breakdown and ask for more
- 2 Check by eye that the hours column has 0 and 24, and kind has only Design / Implementation / Review / Testing
- 3 In data-memo.txt in the tpro-work folder, write one line on what differed from your request: how many rows were missing or which condition was ignored

Result

data.csv (31 lines) and data-memo.txt (1 line) in the tpro-work folder on your desktop.

Completion criteria for test data

Done when there are 31 lines and hours includes 0 and 24

Done when

- data.csv has 31 lines (1 header + 30 data rows)
- The hours column has 0, 0.5, 8 and 24 at least once each
- memo holds fictional task descriptions, with no real company names
- kind uses only Design / Implementation / Review / Testing

If you're stuck

- It stops at 15 of 30 rows. Say "Continue with the rest". Count the rows every time
- Text appears before or after the code block. You can delete it by hand
- hours has values like 1.25 that are not in 0.5 steps. Add "Use 0.5 steps only" to the prompt
- Saved, but not in Explorer. The save dialog was in another folder. Choose tpro-work on the desktop again

If time allows

- Ask for 3 more rows with the same category twice on the same day, to add duplicates
- Have the AI total hours by category, then cross-check one category yourself
- Ask for 3 rows dated next month to add future dates

Today's report counts **the cases you designed**, not results

1

1 Overview

Two lines on what the test design covers: the target app and when you designed it

2

2 Target feature

Only one: filter by category, total hours by category, or the hours input check

3

3 Number of designed cases

The total and a breakdown by priority. Use the counts from testcases.md

4

4 Not run and why

No cases are run today. As the reason, write "To run after the import is built"

5

5 Findings

One line is enough. Example: 2 order-related viewpoints were missing, so a person added them

Do not add a pass/fail section. Nothing has run, so there are no numbers to write.

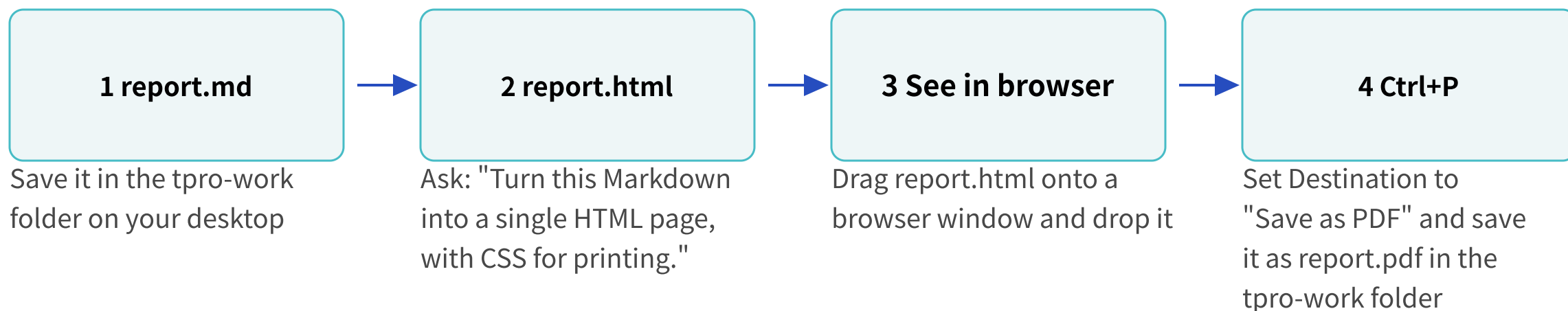
In the prompt, set the heading order and ban pass/fail sections

Attach testcases.md to the chat before you ask. Always recount the numbers yourself afterward.

```
Write a test design report in Markdown based on the attached testcases.md.  
Headings, in order: Overview/Target feature/Number of designed cases/Not run and why/Findings  
Conditions:  
- In Number of designed cases, add a table: the total and counts by priority (High, Medium, Low)  
- In Not run and why, write "None run. To run after the import is built."  
- Keep Findings to 3 lines or fewer  
- Do not add sections for pass/fail or bug counts
```

Do not trust the counts as given. Count the rows in testcases.md and compare.

From report.md to PDF in 4 steps, **with nothing to install**



Ask for print CSS that keeps table rows from splitting across pages and sets margins.

Go all the way from test case table to test report to PDF

Scenario

Work that ended at design is the hardest to explain when you report to your manager. One page with case counts and the reason they were not run makes a short verbal update enough. Go once from the test case table to a test report and a PDF.

Steps

- 1 Use the prompt on the "Prompt for the test report" slide to get report.md, and save it in the tpro-work folder on your desktop
- 2 Open testcases.md, count the rows, and check that the report total and the counts by priority match. Fix any difference by hand
- 3 Ask "Turn this Markdown into a single HTML page, with CSS for printing." and save it as report.html in the tpro-work folder

Before you save, check yourself that the report has no words that identify the client.

Exercise 8: Test report as PDF (continued)

Steps

- 1 Drag report.html onto a browser window to open it, and press Ctrl+P. Set Destination to "Save as PDF" and save it as report.pdf in the tpro-work folder
- 2 In the preview, check the right edge of tables and where headings fall. If something is cut off, ask to switch to landscape or for CSS that makes the text smaller

Result

report.md, report.html and report.pdf in the tpro-work folder on your desktop.

Completion criteria for the test report

Done when the counts match yours and no table is cut off

Done when

- The PDF is 1 or 2 A4 pages, and no table is cut off on the right
- No heading is left alone on the last line of a page
- The report total and counts by priority match your count of testcases.md
- Not run and why says "None run. To run after the import is built."

If you're stuck

- The print preview is blank. Under More settings, check "Background graphics"
- Double-clicking report.html opens another browser. Drag it onto the browser window to open it
- The AI's counts do not match the real ones. This happens often. Overwrite them with your own counts; you may also note the mismatch in Findings.
- The table is cut off. First set the paper to landscape; if still cut off, remove columns.

If time allows

- Add one more table to the test report with only the High-priority rows.
- Start the test report with a 3-line summary: feature, cases designed, cases added by hand.
- Have AI write in-house and client versions of report.md in different tones, and compare them.

Two ways to make a PDF

Today **browser printing covers it**. The extension is for later

Method	Today	Common pitfalls
Browser printing	Today's method. No setup needed	Adjust page breaks with CSS. The right edge of tables is often cut off.
Markdown PDF	For later reference. Follow your workplace rules on installing it.	Without the setting, it tries to download a browser for conversion and hangs on a company network.

If you use the extension, set `markdown-pdf.executablePath` in Settings to the path of the Edge executable on your PC.

Break

After the break, **implement import** and write unit tests

After the break

Check that data.csv and report.pdf are directly under tpro-work.

Next section

Implement the import from the Import Specification, and read the tests AI writes.

We resume in 10 minutes.

06

Implementing and testing import

Split the logic into files, add import, and triage FAILs with your own tests.html

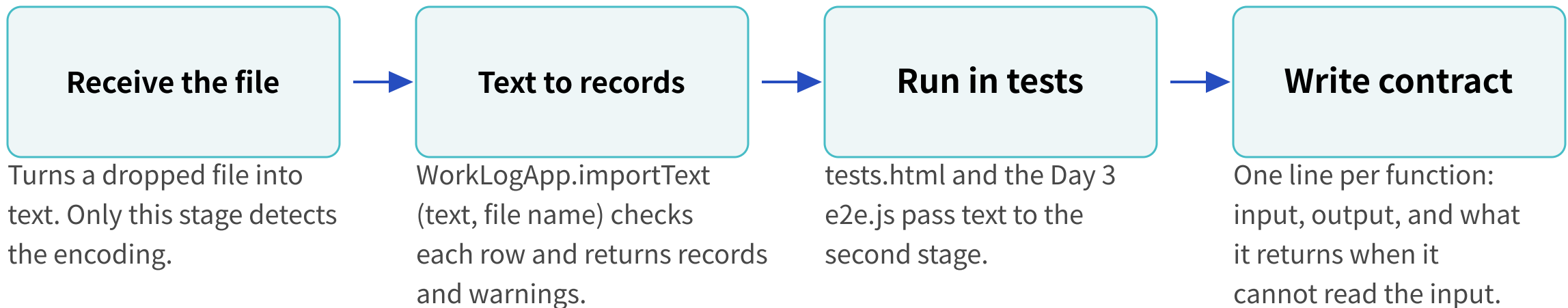
Opened directly in the browser, the page **cannot fetch files**

Mechanism	Opened via file://	How we use it today
fetch	Cannot fetch files	Receive files by drag and drop
import with type="module"	Blocked by the browser	Pass via script tag load order
Global names	Work as is	Put functions on window.WorkLogCore
Category map and holiday CSVs	Cannot be read	Copy as constants into worklog-config.js

The load order is lib/papaparse.min.js, worklog-config.js, worklog-core.js, then the page script.

The 2 import stages and the contract

Detect the encoding **only in the first stage**



Exercise 9: Rewrite 2 lines of the instructions

Fix the instructions first, or Agent goes back to one file

Scenario

The Day 1 copilot-instructions.md has lines that conflict with today's file split. When the premise changes, update the instructions file too.

Steps

- 1 Find the conflicting lines in copilot-instructions.md, rewrite them yourself as 2 lines (where logic and UI go, and what not to use), and save [2min]
- 2 If you have AGENTS.md, change its structure lines to the same 2 lines and save [1min]

Result

2 lines in tpro-work/.github/copilot-instructions.md (plus AGENTS.md, if any)

In the contract, also write **what to return on unreadable input**

Write one line per function in docs/core-contract.md. You decide the < > parts.

```
<function name> takes <input> and returns <output>.
```

```
If it cannot read the input, it returns <output>.
```

```
6 functions: parseCsvText, normalizeDate, parseHours, normalizeKind, sumByKind, detectIssues
```

```
Warnings use the R-09 warning codes in import-spec_v1.md
```

You decide whether to return null or a result with a warning code

Exercise 9: The plan and stage 1

Check first that the plan **keeps the existing render**

Scenario

Have Plan make a 2-stage plan, and have Agent implement only stage 1: worklog-core.js and worklog-config.js.

Steps

- 1 Give Plan 5 items: the contract, the spec, index.html, the category map and the holidays. Get a 2-stage plan [2min]
- 2 Check that the plan keeps the existing render and the records field names, then hand stage 1 to Agent [1min]
- 3 Read the diff and click Keep. Compare the category map and holiday counts in worklog-config.js with the original CSV row counts [3min]

Result

tpro-work/worklog-core.js (detectIssues empty) and tpro-work/worklog-config.js

Exercise 9: Stage 2 and checking it works

Check not only the new import but also **the Day 1 features**

Scenario

Stage 2 is index.html only. Add an import area and a warning area, and apply the spec's hours check to manual entry too.

Steps

- 1 Ask Agent for stage 2, check that worklog-core.js is not in the diff, and click Keep [4min]
- 2 Reload the browser and check 4 things: the filter, totals by category, and manual entry of 0.25 and 24.25 [3min]

Result

The import and warning areas in index.html, and the manual entry check

Done when: Import implementation

Done when

- copilot-instructions.md no longer has the line "Do not split files"
- All 6 lines in core-contract.md say what to return on unreadable input
- The category map and holiday counts in worklog-config.js match the original CSV row counts
- The Day 1 filter and totals by category work both before and after import
- Manual 0.25 and 24.25 follow R-03 and R-12; the warning area shows R-09 warning codes

If you're stuck

- Everything went into index.html → Save the instructions file, fix the AGENTS.md structure lines, and resend in a new chat
- Manual 0.25 is rejected → The Day 1 check function is still there. Have AI merge the checks into parseHours alone
- Blank page → Read line 1 of the F12 Console. CORS means type="module"; Papa means load order

If time allows

- Copy today's data.csv as data_r01.csv, change only line 1 to Date,Category,Description,Hours, then drop it in. Dropping the original data.csv and getting nothing read is correct per R-01. Note that in decision-log.md in one line too.

Recovery slot

Here, **everyone gets** the code the next tests will call

If import does not work

Write in one chat line which stage failed and the symptom. The instructor replies with only the next step

If it still does not work

Get the instructor's version of the failed stage; update your contract to its function names and return values

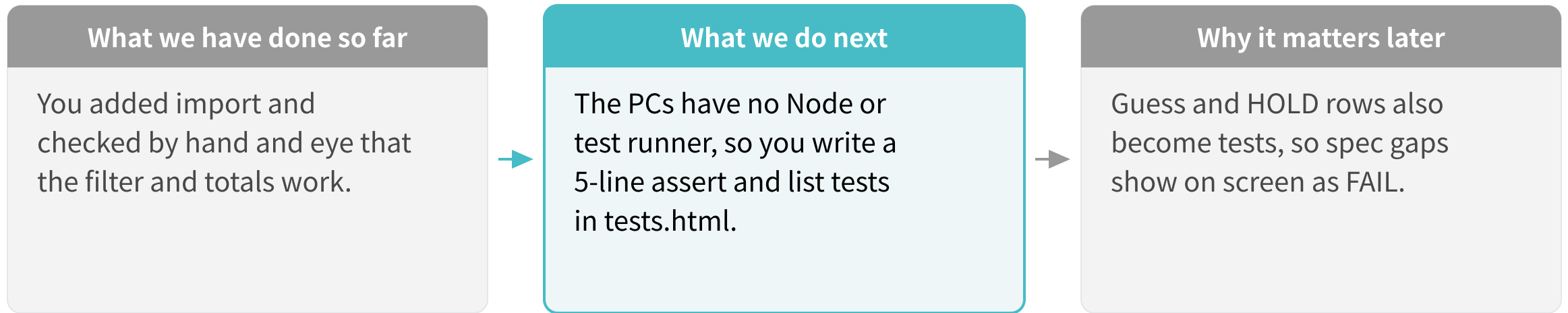
If it works

Copy data.csv with the header set to the spec's 4 columns, drop it in, log the warning count in decision-log.md

If you used the instructor's version, note the file name in decision-log.md in one line

From manual checks to your own tests

No test runner? **Build one**



Exercise 10: Read and check assert.js

Have Ask write assert.js; **you read the branches yourself**

Scenario

Your PC has no Node or test runner. The base is a small function that takes the expected value, actual value and case ID and prints one result line.

Steps

- 1 Create a tests folder in tpro-work and send Ask: "Write an assert that takes expected, actual and case ID, prints one line (PASS if equal, FAIL with both values if not, HOLD if expected is HOLD) and counts them" [2min]
- 2 Read the returned code. Check that all 3 branches (PASS, FAIL, HOLD) exist and that FAIL shows both values [2min]
- 3 If a branch is missing, ask for just that one, then save as tests/assert.js [1min]

Result

tpro-work/tests/assert.js, a base you read and checked yourself

Paste the **SUMMARY** line as is into the Day 3 practical exam

A prompt pattern for Agent. Fill in the <> parts yourself.

```
Create tests.html. Load <the 5 files, in order>.  
Do not change <the files you wrote>.  
At the top, show <the 3 counts> and a SUMMARY line for submission.  
Put <3 counts and 3 ID lists> in the SUMMARY line.  
Do not use <what not to use>.
```

If the diff includes tests/assert.js, do not click Keep

Exercise 10: Test hours given as strings

Write the first test from **the symptom you saw on Day 1**

Scenario

Recall the Day 1 totals exercise where the numbers got joined side by side. In the 8 initial records, hours are still strings like "3" and " 4.5".

Steps

- 1 At the top of tests/core.tests.js, write 3 lines yourself that pass records with string hours to sumByKind [3min]
- 2 Open tests.html in the browser and look at the 3 results [1min]

Result

The first 3 lines of tests/core.tests.js, the tests you wrote

If Agent writes them, it **copies expected values from the code**

1

Close the worklog-core.js tab

Open tabs are attached implicitly, even without #

2

Remove the current file attachment

Remove the open file that is attached in the input box

3

Attach only 2 files

Attach only #testcases.md and #core-contract.md; first get a table of expected results as return values

4

Paste the code block

Paste the returned code block below your 3 lines in tests/core.tests.js and save

Have AI add ? to the case ID of Guess rows and ! to HOLD rows

Exercise 10: Check the generated tests

Check 3 tests to see if expected values came from the code

Scenario

Check that expected values AI restated as contract return values pass or fail like the expected results in testcases.md. If a row differs, send back its case ID for a rewrite.

Steps

- 1 Check that worklog-core.js is not in the answer's References [1min]
- 2 Pick 3 tests. Check that the restated expected values pass or fail like the expected results in testcases.md and the calls follow the contract. If not, send them back [2min]

Result

Results for the 3 checked tests, one line each in decision-log.md

Done when: tests.html

Done when

- tests.html lists PASS and FAIL with case IDs, with counts and the SUMMARY line on top
- assert.js is still the 5 lines you wrote, and counts HOLD cases as HOLD
- There is a test that passes hours as strings
- Every row with Layer = Unit is a test, with ? on Guess rows and ! on HOLD rows
- worklog-core.js does not appear in Ask's References

If you're stuck

- Every test is TypeError → Attach #core-contract.md and have AI regenerate them
- tests.html is blank → Check line 1 of the F12 Console: load order or a typo
- worklog-core.js is in References → Discard the answer, remove the attachment, and resend in a new chat

If time allows

- Add an assert that compares arrays and objects by content. Then you can compare the whole sumByKind result.

Exercise 11: Add detectIssues afterward

With tests first, you see the code take effect on screen

Scenario

detectIssues, which checks relations between rows, has only a contract and is still empty. Fill it in now that the tests exist.

Steps

- 1 Give Agent the detectIssues line of the contract and R-06 to R-08, and have it implement only detectIssues in worklog-core.js [2min]
- 2 Check that no tests/ files are in the diff, then click Keep [1min]
- 3 Reload tests.html and check that the R-06 to R-08 rows changed from FAIL to PASS [2min]

Result

detectIssues in worklog-core.js

Just say "fix it" and AI **changes the tests to match the code**

Where to fix	When it applies	How to ask
Implementation	The expected value is from the spec, but the code differs	Have AI fix the function without changing tests/
Test	The expected value was a guess	Fix testcases.md to match the spec, and have AI fix the test with the same value
Client	The spec decides neither way	Move it to open-questions.md, set expected to HOLD

Pick one remaining FAIL per direction, and write what to fix and why in decision-log.md, one line each

Exercise 11: Triage in 3 directions

Say what to fix first, then ask AI

Scenario

Have AI fix the 3 items in decision-log.md as you decided.

Steps

- 1 Have Agent fix the Implementation one; if tests/ is not in the diff, click Keep [3min]
- 2 For the Test one, have AI fix testcases.md and the test, and remove ?. For the Client one, move it to open-questions.md and set expected to HOLD. Click Keep [3min]

Result

Fixed worklog-core.js, fixed tests/core.tests.js with a HOLD, entry in open-questions.md

Done when: FAIL triage

Done when

- Rows R-06 to R-08 changed to PASS after you added detectIssues.
- decision-log.md has the fix target and reason for all 3: implementation, test, client.
- In the case where you fixed the implementation, the test file is unchanged.
- HOLD is counted apart from FAIL and appears in HOLD_IDS on the SUMMARY line.
- After a reload, the 3 cases show the results you decided.

If you're stuck

- The test was rewritten to match the code → Undo the Keep and resend with "Do not rewrite the tests" at the start
- Code or test? → Check the Source column. Spec means fix the code; Guess means fix the test
- Not enough FAILs to triage → Get a replacement file from the instructor, or copy sample rows from fallback

If time allows

- Copy today's data.csv as data_r01.csv, change only line 1 to Date,Category,Description,Hours, then drop it in. Dropping the original data.csv and getting nothing read is correct per R-01. Note that in decision-log.md in one line too.

07

Display and export gaps

After import, your app shows text written by others. Close the gaps you opened yourself.

Your app shows text written by others **starting today**

Content set with innerHTML

The string is parsed as HTML. If Description holds an img tag with onerror, its script runs. The totals table or warning area the AI added may work this way.

Content set with textContent

Shown as plain text. The render in the Day 1 handout uses this. To show text, this is enough without an extra library.

Same problem: outside values are read as commands

Gap	What happens	Fix and its cost
CSV injection	If a value starts with = + - @, Excel reads it as a formula.	Adding ' at the start stops the formula. Reading it back changes the value.
String-built SQL	Outside values run as part of the SQL statement.	Not in this app. Write how to fix it in the server-side code.

The server-side code is in day2/review/review_target.js. It is for reading only.

Exercise 12: Find the gaps in your app

Ask does the review; Agent and you do the fixes

Scenario

day2/data/worklog_2026-07-08.csv holds July to August data exported from the legacy timesheet system. Some rows have tags or a leading symbol in Description.

Steps

- 1 Drop the CSV on the app; check tagged rows in the list, warnings and totals [2min]
- 2 Write 3 lines yourself: the viewpoints you want Ask to check [3min]
- 3 Give Ask #index.html, #worklog-core.js, #review_target.js and your viewpoints, and get only the findings in a table [2min]

Result

3 viewpoints and Ask's findings table: file, line, viewpoint, why risky, fix direction

Look at the real output and **decide yourself**

Add ' at the start

Excel shows it as text. If you drop the exported CSV back on the app, Description comes back with the leading '.

Only wrap in quotes

The value stays the same on re-import. Open the file in Excel to see how it looks.

Exercise 12: Export and re-import

Drop your export **back into your own app**

Scenario

Check how the exported CSV looks in Excel and what values come back on re-import.

Steps

- 1 Open the exported CSV in Excel and check rows with a leading symbol [1min]
- 2 Re-import the same CSV and check that Description matches the original [1min]

Result

How it looks in Excel vs. the re-imported value (1 line in decision-log.md)

Exercise 12: Two regression tests for toCsv

Do not copy expected values from the code; write them yourself

Scenario

Keep the handling you just chose in tests, so it holds when someone changes toCsv later.

Steps

- 1 Write the expected toCsv output yourself, one line each, for records with Description =SUM(A1) and -0.5h adjustment for the previous day [2min]
- 2 Have Agent add them to tests/core.tests.js, and check that both PASS in tests.html [2min]

Result

2 regression tests in tests/core.tests.js

On Day 3, these 3 viewpoints become **the reviewer agent's text**

Scenario

Keep the 3 viewpoints you first sent to Ask, and write 2 lines on how to fix string-built SQL, which this app does not have.

Steps

- 1 Create security-notes.md in tpro-work and copy in the 3 viewpoints [1min]
- 2 Write 2 lines on how to fix the string-built SQL in review_target.js, and save [2min]

Result

tpro-work/security-notes.md (3 viewpoints and 2 lines on fixes)

Done when: export and gaps

Done when

- `<img src=x onerror=...>` in Description shows as text, and no script runs.
- In the exported CSV opened in Excel, values starting with `= + @` do not run as formulas.
- Valid values starting with `-` show as decided; you can explain the change on re-import.
- Both regression tests PASS.
- `security-notes.md` has 3 viewpoints and 2 lines on fixes.

If you're stuck

- Excel shows 1 column → Check delimiter or BOM; just view the first characters in VS Code
- `'` stays after re-import → That is the cost. Note in `decision-log.md` if import should strip it
- An alert appears → Match the file and line of that innerHTML against Ask's findings table

If time allows

- Make a table of which exported columns need the leading-symbol fix. Check that adding `'` to Hours makes it non-numeric when read back.



08

Review and Day 3 preview

Check the files you made today and carry them into Day 3

Today's cycles

AI produced the drafts; **you compared and decided**

Fix

You gave only the symptoms, had AI fix it, and checked the total by hand.

Design

AI drafted viewpoints and test cases; you traced each expected result back to the spec.

Verify

AI built the import; you checked its tests against the spec and decided where to fix each FAIL.

A passing test only shows that the code matched the expected values you wrote

Day 3 overview

Day 3 uses today's files for **E2E runs and skills from steps**

E2E

Run import through reload end to end in the browser. Pick 4 cases from today's test case table.

Skills and agents

Turn today's steps into a skill and your security viewpoints into a custom agent.

Certification and practical exam

After the Literacy Check, finish an import of the new attendance format in the practical exam.

Day 3 is 1 week away

What to keep for next session

Bring the tpro-work folder to Day 3 **as it is**

tpro-work

Keep index.html, testcases.md, decision-log.md, data.csv, worklog-core.js, the tests folder and security-notes.md. Do not delete them.

Sign in

Stay signed in to Copilot.

index.html in the worklog-app folder is as it was at the end of Day 1. Use it to roll back



Givery, Inc.

Teito Shibuya Bldg. 8F, 15-13 Nampeidai-cho, Shibuya-ku, Tokyo 150-0036, Japan