



HANDS-ON GUIDE

Hands-on AI-Assisted Development, IT track, Day 2 Hands-on Guide

Hands-on AI-Assisted Development, IT track, Day 2

This guide covers the 12 exercises you do hands-on on Day 2, with the steps and how to recover when you get stuck. The slides keep moving, but you keep this guide. If you stop, go back to the page of the exercise you are working on.

Contents

What you do today

What you use

Exercise 1: Where the handout ZIP and index.html go

Exercise 2: Break one line

Exercise 3: Fixing with AI

Sharing test work with AI

Exercise 4: Create a viewpoint list and a test case table

Sources of expected results

Exercise 5: Recheck today's test case table

Exercise 6: Derive cases from a decision table

Exercise 7: Create 30 rows of test data as CSV

Exercise 8: Produce the test report as a PDF

How to split files so they work over file://

Exercise 9: Contract and import implementation

Recovery slot

Exercise 10: From assert.js to tests.html

Exercise 11: Triage of FAILs

The 2 gaps in display and export

Exercise 12: Add export and close the gaps

Where to look when things go wrong

What you take home today

Rules to follow

What you do today

Today you build tests with AI, using the Worklog app to which you added filtering and totals on Day 1. In the morning you make a test case table without a specification. After lunch the Import Specification arrives, and you recheck where the expected results come from. In the second half of the afternoon you have AI implement the import, and check the tests that AI wrote against the specification.

Time	Steps	Minutes
Morning	Extracting the handout ZIP and debugging with AI (Exercises 1 to 3)	[60min]
Morning	Sharing test work with AI	[37min]
Morning	Break	[10min]
Morning	Test viewpoints and test cases (Exercise 4)	[114min]
Afternoon	Test design basics (Exercises 5 and 6)	[60min]
Afternoon	Break	[10min]
Afternoon	Test data and the test report (Exercises 7 and 8)	[70min]
Afternoon	Break	[10min]
Afternoon	Import implementation and tests, display and export gaps, and review (Exercises 9 to 12)	[109min]

The total is 480 minutes. The lunch break comes after "Test viewpoints and test cases". The instructor will tell you its length on the day.

Put the files you make today directly under tpro-work (or the folder you are told to use), not inside the day2 folder.

What you use

The tools are the same as on Day 1: VS Code, GitHub Copilot and the browser. You do not use Python, Node.js or the command line.

Contents of the handout ZIP

The top level of the handout ZIP is the day2 folder. If you choose the tpro-work folder as the destination, day2 is created inside tpro-work.

Contents	Used for
day2/docs/import-spec_v1.md	Material for after lunch. It has requirements R-01 to R-12 and a table of warning codes
day2/docs/category-map.csv , holidays_2026.csv	Used to convert the categories of the legacy timesheet system and to determine holidays
day2/data/worklog_2026-07-08.csv	The July and August hours exported from the legacy timesheet system. You use it from the import implementation onward
day2/review/review_target.js	A file you only read in the exercise on the display and export gaps
day2/fallback/	Replacement files for anyone who is missing files. Use them only when the instructor tells you to

Files you create in this folder today

File	Exercise where you create it
index_ok2.html	Exercise 2. A copy kept from before you break the code
testcases.md , prompt-memo.txt	Exercise 4. The test case table and a note on the one sentence that worked
decision-log.md , open-questions.md	Exercises 5 and 6. The recheck of sources and the questions to ask the client
data.csv , data-memo.txt	Exercise 7. The 31 rows of test data and a note on where the result differed from your instructions
report.md , report.html , report.pdf	Exercise 8. The test report for your test design
docs/core-contract.md , worklog-core.js , worklog-config.js	Exercise 9. The import contract and implementation
tests.html , tests/assert.js , tests/core.tests.js	Exercises 10 and 11. Your own test runner and tests
security-notes.md	Exercise 12. Test viewpoints and the approach for display and export

Exercise 1: Where the handout ZIP and index.html go

About [6min]

Scenario

On Day 1 you added features to `index.html` inside the `worklog-app` folder. Today's handouts and the Day 3 materials assume that `index.html` is directly under `tpro-work`. First, you align the file locations.

Steps

1. Extract the handout ZIP into the `tpro-work` folder and check that a `day2` folder now exists inside `tpro-work`.
2. Copy `index.html` from the `worklog-app` folder and paste it directly under `tpro-work`, at the same level as the `.github` folder. Do not delete the original file.
3. In VS Code, reopen `tpro-work` as a folder, open the `index.html` directly under it in the browser, and check that the filter field and the totals table appear.
4. If you do not have the Day 1 `index.html`, or if the filter and totals do not work, copy `day2/fallback/index_day1.html` directly under `tpro-work` and rename it `index.html`.

Check your result

- `index.html`, the `day2` folder and the `.github` folder are side by side directly under `tpro-work`
- The `index.html` in the `worklog-app` folder is still there

If it does not work

Symptom	Fix
A second <code>tpro-work</code> folder was created inside <code>tpro-work</code>	The one that contains the <code>.github</code> folder is the real one. Choose that one as the destination again.
You do not know where to paste <code>index.html</code>	In the VS Code Explorer, paste it at the same level as <code>.github</code> . If you paste it one level up, it is not read.

Exercise 2: Break one line

About [8min]

Scenario

If you approve code that AI returned without reading it, one-character differences like this can slip in. One comma in the initial value changes the type of the total, and no error screen appears either. You look once, in your own file, at how the screen looks when this happens.

The one line to break

From the line that calculates the total, delete only the `, 0` at the end. The form of the total that Copilot wrote on Day 1 differs from person to person. Below is one example.

```
// Original code (inside tpro-work/index.html)
.reduce(function (sum, record) { return sum + parseFloat(record.hours); }, 0);

// Broken code. Delete only the , 0 at the end
.reduce(function (sum, record) { return sum + parseFloat(record.hours); });

// If your code does not use reduce, changing the initial value 0 of the total to '' gives the same symptom
```

Steps

1. Copy index.html and keep the copy under the name `index_ok2.html` .
2. Open index.html in VS Code, search for reduce with Ctrl+F, and delete the `, 0` at the end of the line that calculates the total. If there is no reduce, change the initial value `0` of the total to `''` .
3. Press Ctrl+S to save, press Ctrl+Shift+R in the browser to reload, and look at the values in the totals table.
4. Change the filter by category, press F12 to open the Console tab, and look at the red lines.

Check your result

- The totals table shows a value that starts with `[object Object]` , or a value where the numbers are joined side by side
- When you filter by category, the list does not change, or a `TypeError` appears in the Console
- `index_ok2.html` is still in `tpro-work`

No deliverable. Before you break it, if you still have only the first 8 records, the totals table shows Design 5.5, Implementation 10.5, Review 2.5 and Testing 5.5. If you added records on Day 1, add up the hours yourself and check.

If it does not work

Symptom	Fix
Nothing changes	Press Ctrl+S to save, then press Ctrl+Shift+R in the browser
The screen went blank	You deleted the parenthesis too. Copy <code>index_ok2.html</code> over <code>index.html</code> to restore it, and delete just the <code>, 0</code> at the end
The totals do not change	You deleted a reduce in a different place. Look inside the function that renders the totals table.
You cannot find the Console	Press F12 and select Console in the tabs at the top

If time allows

- Copy the `TypeError` message shown in the Console into `prompt.txt` as it is. You decide for yourself whether to paste it in the next exercise.

Exercise 3: Fixing with AI

About [14min]

Scenario

Bug reports you receive at a client site usually do not state the cause. Starting from only what the screen shows, you do the first round of listing candidate causes and ruling them out, together with AI.

The 5 lines to send

With the code still broken, send the next 5 lines in the Agent chat. Replace the value on line 2 with the value shown on your own screen. Do not write even a single character of a guess about the cause.

```
I open and run a single HTML file, index.html, directly in the browser.  
The totals table by category shows values like [object Object]2.5.  
When I filter by category, the list does not change and a TypeError appears in the Console.  
List 3 possible causes, then show only the corrected lines.  
Do not add any external libraries.
```

Steps

1. With the code still broken, send the 5 lines above in the Agent chat.
2. Read the 3 candidate causes that came back, and choose for yourself the one that looks most likely to be right.
3. Read the diff, click Keep, press Ctrl+S to save, and reload the page in the browser.
4. Compare the totals table with the numbers you add up by hand, change the filter, and check that nothing appears in the Console.
5. Add the 5 lines you sent and the explanation of the cause that came back to prompt.txt, and save.

Check your result

- The values in the totals table match the numbers you added by hand
- When you change the filter, the list and the totals change, and nothing appears in the Console
- prompt.txt contains the 5 lines you sent and the explanation of the cause that came back

If it does not work

Symptom	Fix
The filter or the totals table is gone	Add "Show only the lines you changed. Do not change existing features." and have it add back only the features that disappeared
The totals come out blank	Send one sentence: "Show 0 for a category with 0 records."
It is not fixed no matter how many times you send it	Copy index_ok2.html back as index.html, and try again just once.

If time allows

- Compare how the reply changes when you write the cause first and send it.

Dividing test work between humans and AI

You do not work hands-on in this chapter. From here on, everything in Day 2 and Day 3 is about testing.

The 4 test stages

Stage	Steps	Coverage today
Planning	Decide what to check and how far to go. Write down the scope and the condition for being done	Not covered
Design	List the test viewpoints, combine the conditions, and make a test case table	The 110 minutes after the break cover only this stage
Execution	Run it as the steps say, look at the screen, and record the result	You run the unit tests later this afternoon, and the full E2E tests on Day 3
Reporting	Count the cases and the ones not run, and summarize to suit the reader	You produce it as a PDF at the end of today

These 4 stages come in the same order in unit tests and in integration tests. If you lose track of what you are working on, come back to this table.

AI produces candidates only

At every stage, what AI produces is a candidate. In planning, it is a candidate for the range you want to check, based on past bug trends. In design, it is a list of viewpoints and a draft of the test case table. In execution, it is a written version of the operation steps and the steps to reproduce a problem. In reporting, it is a count of cases and a neatly formatted table. Everything listed here comes back looking the same on the screen, even when it is wrong.

People decide the following. In planning, it is where to draw the line for release. In design, it is how to read the spec and which cases to drop. In execution, it is the check where you run the app and look at the screen with your own eyes. In reporting, it is the severity of bugs and whether to release. Only someone who knows the impact on the business can decide these.

No report can say "AI guaranteed the test coverage." When you have to explain that cases were missing, "AI produced it" does not hold up. You may leave work to AI only as far as you can put your own name on what it produces.

One line to limit the output

If you ask for 100 cases at once, you lose the energy to check them one by one. The first step toward getting them in parts is to forbid, in one line, what you do not want written.

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

The second line is the one that works. You use the full version, which adds the target, the number of items and the format to these 2 lines, in Exercise 4.

GitHub's official best-practices page also lists two points: do not use generated output without reviewing it, and split a request into small parts. The page is updated from time to time, so check it yourself before you roll this out in your company.

Comprehension check

Which of the following is the most appropriate line for what you may leave to AI in test work?

1. You list the viewpoints yourself and leave only the final write-up of the test case table to AI
2. You give AI past bug trends and leave to it even the decision of how far to check before you can release
3. You leave the work of producing candidates to AI, and you decide which to drop and how severe a bug is
4. You have AI produce 100 cases at once and limit the review to the places where bugs appeared

Choose your answer before you read this.

The correct answer is 3. The same pattern holds at all 4 stages: planning, design, execution and reporting. AI can mass-produce candidates, but which cases to drop and how severe a bug is can be decided only by someone who knows the site and the impact on the business.

Option 1 reverses the roles. Listing viewpoints is quick work for AI. What you should keep is not the final write-up but the decision about which of the listed viewpoints to drop. In option 2, you can use AI to produce candidates for the range from trends, but the decision about when you are done stays with you. In option 4, asking for a large number at once turns the review into a formality, and you cannot notice that cases are missing in the first place.

Exercise 4: Create a viewpoint list and a test case table

Scenario

At a client site, the first test design you are given is usually part of a screen someone else built. The specification is only partly written, and all you have is the screen that runs. From that state, you list viewpoints and turn them into a test case table that other people can run, once through with the provided Worklog app.

This is the longest block today, at 110 minutes. 44 minutes of it are time for you to work hands-on. You do not run the tests this morning. You run them in a later chapter and on Day 3.

The 4-stage structure

Split the path from viewpoints to the test case table into 4 stages and make it more detailed one stage at a time. If you skip a stage and ask for everything at once, you get a table that mixes the whole screen with a single button.

Stage	What you write	Example in the handout app
Stage 1: Viewpoint	The angle for what you want to check	Hours input check
Stage 2: Condition	Breaking down into value ranges and states	0 or less / 0.5 to 24 / over 24 / blank / full-width digits
Stage 3: Case	One operation, written in one line	Enter 0 in Hours and click the Add button
Stage 4: Expected result	What the screen shows then	The message "Enter a value from 0.5 to 24" appears under the Hours field. The list does not grow

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

Equivalence classes and boundary values

An equivalence class is a group of values that give the same result. A boundary value is a value at the edge of such a group. If the range for Hours is 0.5 to 24, then 0 represents the values that are not accepted, 0.5 is the lower edge of the accepted values, 8 represents the middle of the group, 24 is the upper edge, and 24.5 represents the values that are not accepted.

Values that do not sit on the number line are also picked up as equivalence classes: blank, full-width digits and negative numbers. Do not make every combination. Vary one field at a time and combine only the fields that seem related.

Field	Equivalence classes	Boundary values to test
Hours	In range / below range / above range / not a number	0, 0.5, 24, 24.5
Description	Has text / empty / over the limit	Empty, 1 character, limit, limit + 1 character
Date	Today or earlier / future date / wrong format	Yesterday, today, tomorrow
Category dropdown	One category selected / All	When you switch back to All

Steps

1. Narrow the target to one thing. From the handout app you extended on Day 1, choose one of these: the category filter, the total hours by category, or the hours input check. Do not target the whole screen.
2. Paste the HTML of the target part into the chat and use the next prompt to get viewpoints only. Add 2 viewpoints of your own to what comes back.

List only the test viewpoints for the following part of a web app, as bullets.

- The target is only the Hours input field and the Add button

- Do not make a table. Write only the name of each viewpoint, one per line
 - List 10 or fewer
 - Write each viewpoint name as a short noun phrase, such as "XX input check"
- (Paste the HTML of the target part below this line)

3. Get the equivalence classes and boundary values for each viewpoint. Ask for 3 columns: field, equivalence classes and boundary values.
4. Copy the next header row as it is, give it to the chat, and get the table in Markdown. If you do not specify the columns first, AI returns a different format every time.

```
| ID | Test viewpoint | Precondition | Action | Expected result | Priority |
|---|---|---|---|---|---|
| C-01 | Hours input check | Initial data shows in the list | Enter 0 in Hours and click Add | "Enter a value from 0.5 to 24" | High |
| C-02 | Hours input check | Same as above | Enter 0.5 in Hours and click Add | The list grows by 1 and the input fields clear | High |
| C-03 | Hours input check | Same as above | Leave Hours empty and click Add | "Please enter a value" appears and the cursor is in the field | High |
| C-04 | Category filter | Initial data shows in the list | Switch Category to "Implementation" | Only Implementation rows are shown | High |
```

5. Save the table as testcases.md in the tpro-work folder on your desktop.
6. Run the improvement cycle twice. Look at your spec notes and the screen, and find operations that are not in the table. Write each one as a single sentence that names the viewpoint, such as "Registering 2 records for the same category on the same day is missing", and have it added to the existing table. Do not say "Rebuild it". If you have the table rebuilt, the IDs shift.
7. Write one line about which sentence you think worked in prompt-memo.txt in the tpro-work folder.

In one pass, AI produces independent cases, one operation each. The HTML you gave it is one screen frozen in a single state, so the passage of time and the order of operations are not part of its material. That is why they do not appear. Register and then delete. Open the next screen with the previous input still there. Open two tabs and operate them alternately. If you give AI this kind of angle in words, AI can write the rows.

Put only the contents of the handout app into the chat. Do not enter client names, product names or real data.

Check your result

- testcases.md has 10 or more cases, and at least 2 of them are rows you added after noticing a gap
- Every row has a precondition, an action and an expected result
- Expected results say what appears on the screen. "Works normally" and "An error occurs" are not acceptable
- Not every priority is "High". Only cases that stop the work if they fail are High

If you find a row whose expected result cannot be decided from the spec, finding it is itself a result. It is a gap in the spec, so write "PENDING" in the test case table and leave the row there. The same applies in real work.

If it does not work

Symptom	Fix
The target was not narrowed and 30 viewpoints came back	Narrow the target again to one input field or one dropdown
Your Day 1 feature is only partly added	You may limit the target to what is already on the screen, up to the filter field. Write the expected results from the spec you decided on yourself
You asked for viewpoints only but a test case table came back	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 case rows come back	Go back to the table of conditions and pick the values again from the "Boundary values to test" column
You asked to add rows but got more rows that look alike	"Add more" only adds more rows. Name the operation that is missing and send that
testcases.md cannot be found	It was saved in a different folder. Save everything in the tpro-work folder on your desktop

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.

Sources of expected results

Passing tests and meeting the spec are different things. If you have AI write tests, it copies the expected values from inside the implementation. Tests made that way always pass, but they are no evidence that the spec is met. You could tell that the total in the debugging exercise was broken by looking at the screen. A test that copied its expected value from the implementation would have passed `[object Object]2.5` as the correct answer.

Source	Contents	Counts as evidence for the spec
Specification	A requirement in Import Specification v1. In the column, write the requirement ID, like <code>Spec (R-03)</code>	Yes
Acceptance criteria	States, in numbers of items, what must be done before the work is accepted. In the column, write <code>Acceptance criteria (AC-1)</code>	Yes, once the client has approved it

Source	Contents	Counts as evidence for the spec
Implementation	The fact that the current code behaves that way	No. Mark it HOLD until the specification confirms it
Guess	A value you decided yourself because it seemed right. The range for Hours that you wrote this morning is usually this	No

The values you enter in the Source column of `testcases.md` are these 4: Spec, Acceptance criteria, Guess and HOLD. Do not throw away the Guess and HOLD rows; make them tests too. A HOLD row keeps PENDING as its expected value, so it fails with any implementation. What the specification does not decide shows up on the screen as a FAIL row.

Test layers

Layer	How to check	Example in the Worklog app
Unit	Pass a value to a function and check the value it returns	Convert the hours string to a number
E2E	Go through the screen and check in the order of the operations	Export the CSV while a filter is applied
Manual	A person looks at the result and decides, with no automatic judgment	How the exported CSV looks when you open it in Excel

Sort them this way. If you pass a value to a function and a value comes back, it is a unit test. If it does not happen unless you operate the screen, it is an E2E test. Every row of today's test case table fits one of these 3 layers.

The first 2 lines of decision-log.md

Pick one expected result from your own `testcases.md` and write 2 lines at the top of `decision-log.md` [2min]. Line 1: where you took that value from. Line 2: whose word would change that value, if it changed. Write them without opening the specification yet.

If you feel that writing expected values by looking at the implementation is normal: it can be used to confirm that existing behavior does not change, that is, for regression. But treat that as a different thing from checking the spec. If you know the term test oracle, today's Source is another name for it.

Exercise 5: Recheck today's test case table

The working time is [24min]. The breakdown: read the specification [3min], have the requirement ID mapping produced [4min], check 6 rows yourself [5min], add columns [3min], write down the mismatches [3min], have the gaps in the spec listed [3min], and write the open questions [3min].

Purpose

`day2/docs/import-spec_v1.md` is made of 12 requirements. You wrote today's test case table without a specification, so whichever target you chose, it will conflict with the specification somewhere. Where it conflicts depends on the target you chose.

Find the requirements related to your chosen target in the requirements column of the specification.

Separate the rows that come from the specification from the rows that are guesses. Do not rewrite the expected result of a conflicting row here. In Exercise 10 you turn it into a test as it is, and in Exercise 11 you decide what to fix after you see the FAIL. Whether to fix the implementation or the test is not decided until you run the test against the implementation. What the specification does not say, you leave as a question for the client.

What you bring from the earlier exercises

The `testcases.md` from this morning, and the first 2 lines of `decision-log.md` that you wrote at the start of the chapter on sources of expected results. The columns of `testcases.md` should be these 6: ID, Viewpoint, Precondition, Action, Expected result and Priority.

Files you use

- `day2/docs/import-spec_v1.md`
- `tpro-work/testcases.md`
- `tpro-work/open-questions.md` You create this new file in this exercise
- `tpro-work/decision-log.md`

Steps

1. Open `import-spec_v1.md`, read R-01 to R-12, and mark the requirements related to the target of your test case table [3min]
2. In Ask, attach `testcases.md` and `import-spec_v1.md` with #, and have it list in a table which requirement ID each row's expected result comes from. Do not let it change any files [4min]
3. Pick 6 rows from the mapping that came back and compare them yourself with the matching sentences in the specification. Mark the rows where the requirement ID is wrong [5min]
4. Have Agent add a Source column and a Layer column to `testcases.md`. Have it fill in the values with the results you matched yourself in Step 3. Read the diff and Keep [3min]
5. Find 3 rows that conflict with the specification and write the case ID, the conflicting requirement ID and which side you think is right, one line each, in `decision-log.md`. Do not change the Expected result column; set the Source to Guess [3min]
6. Have Ask list only two kinds of places in the specification: places where it says nothing, so the result changes depending on who implements it, and places that can be read in two ways. Do not have it propose fixes [3min]
7. Directly under `tpro-work`, create `open-questions.md` and write 3 questions to ask the client. Choose 2 rows to drop from today's cases, and write the reasons in `decision-log.md` [3min]

Include these 4 elements in the prompt for Step 2.

Element	What you write
Attachments	<code>#testcases.md</code> and <code>#import-spec_v1.md</code>
Output format	A table with 3 columns: ID, requirement ID, and the specification sentence used as the basis
If there is no basis	Set the requirement ID to None
Off limits	Do not rewrite the file

It helps to have the AI show, in a column, the specification sentence it used as the basis. A row where the sentence and the expected result do not match is a mix-up by the AI. The prompt for Step 4 includes 3 things: the names of the columns to add and their possible values, the list you checked yourself, and an instruction to leave the other columns, expected results included, unchanged. The possible values are as follows. For Source: Spec (requirement ID), Acceptance criteria (AC-number), Guess and HOLD. For Layer: Unit, E2E and Manual.

What to think through yourself

- 3 rows that conflict with the specification. After you read the specification, write in your own words which side you think is right
- 3 questions for the client. Write each as a two-option question, so that you can implement it whichever option the client chooses
- The 2 rows to drop from today's cases, and the reasons

If you write "Please make the specification more detailed", you will get no reply. Ask in a way that settles the implementation whichever way the answer goes, for example: "In this case, do you not import the row, or import it and show a warning?"

Resulting files and where to save them

- `tpro-work/testcases.md` gains 2 columns, Source and Layer. The expected result column stays as it was in the morning
- `tpro-work/open-questions.md` gets 3 questions for the client. In Exercise 11, it becomes one of the places a FAIL can go
- `tpro-work/decision-log.md` gains 3 lines of conflicts and 2 lines of reasons for dropping cases. The 3 conflict lines are the draft you use when you triage FAILs in Exercise 11

Check your result

- Every row of `testcases.md` has a Source and a Layer
- Rows whose Source is Spec have a requirement ID
- The expected result column is unchanged from the morning
- `decision-log.md` has 3 rows that conflict with the specification, each with a requirement ID
- `open-questions.md` has 3 questions for the client, each written with two options

- The reasons for the 2 dropped rows are in `decision-log.md`

If it does not work

Symptom	Fix
Ask labels every row Spec	Have it show the specification sentence it used as the basis in a column. A row where the sentence and the expected result do not match is a mix-up
You cannot find rows that conflict	Reread only the requirements related to the target you chose. If you still do not reach 3 rows, read C-06, C-08 and C-11 in <code>day2/fallback/testcases_sample.md</code> . All 3 rows conflict with R-03
You want to fix the expected result of a conflicting row	Do not fix it now. Fix it after you see the FAIL
Agent also rewrote the expected results	If the diff shows that the expected result column changed, do not click Keep. Send the request again and say to only add the columns
The open questions turn into requests	Rewrite each as two options, so that the implementation is settled whichever answer comes back

If time allows

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

Exercise 6: Derive cases from a decision table

The working time is [26min]. The breakdown is: write the conditions [6min], get the decision table [2min], add rows to the test case table [3min], check the expected results [5min], write the order and state cases [6min], and write the acceptance criteria [4min].

Purpose

In the morning, each case was independent, one operation at a time. The import is different. The result changes with the relation between rows: the same row appears twice, the total for one day exceeds 24 hours, or a row on a holiday has the Holiday work approved marker. You first write the combinations of conditions in a decision table, and then have the AI derive the rows. You check the expected results against the requirement IDs in the specification.

You write the acceptance criteria yourself. The specification does not say what must work before the import is accepted. However, acceptance criteria you wrote yourself are not part of the specification until the client approves them. Put them in `open-questions.md` as awaiting approval, and mark expected results that come from them as Acceptance criteria (AC-n), so that you keep them apart from the spec.

What you bring from the earlier exercises

These are `testcases.md` , to which you added Source and Layer in Exercise 5, and `open-questions.md` .

Files you use

- `day2/docs/import-spec_v1.md`
- `day2/docs/category-map.csv`
- `day2/docs/holidays_2026.csv`
- `tpro-work/testcases.md`
- `tpro-work/open-questions.md`

Steps

1. Write 6 lines of import conditions yourself: the date format, the hours format with its range and step, the category names, a blank description, the same row twice, and a holiday with the Holiday work approved marker. Put one condition on each line, and list the possible values too [6min]
2. Give Ask the 6 condition lines, `#import-spec_v1.md` and `#holidays_2026.csv` , and have it produce a decision table [2min]
3. Have it derive cases from the decision table and add them from I-01 under the heading Import in `testcases.md` . Read the diff and click Keep [3min]
4. Choose 5 expected results, and check the requirement IDs and the specification sentences yourself [5min]
5. Write 3 order and state cases yourself: drop the same CSV in twice, drop it after adding one record by hand, and drop it while a filter is applied. Have Agent add them, and click Keep [6min]
6. Write 3 acceptance criteria for the import as AC-1 to AC-3 in the section Awaiting approval of `open-questions.md` , and save. In `testcases.md` , set Source to Acceptance criteria (AC-n) for rows that come from acceptance criteria [4min]

Include these elements in the prompts for Step 2 and Step 3.

Element	What you write
Conditions	Paste the 6 lines you wrote, as they are
Result values	3 values: Imported, Not imported, Imported with warning
Basis	Have it put the requirement ID for each result in a column. Have it judge holidays by the holiday CSV and by weekends
When the result cannot be decided	Have it mark combinations whose result the specification does not decide as PENDING
Format of the added rows	8 columns: ID, Viewpoint, Precondition, Action, Expected result, Priority, Source and Layer. Have it number the IDs from I-01, and write each expected result as what appears on the screen or how many rows are imported

Element	What you write
Handling of HOLD rows	For rows that are PENDING in the decision table, have it set Source to HOLD and leave the expected result as PENDING
Off limits	Do not change existing rows

What to think through yourself

- 6 condition lines. List even the possible values yourself
- 3 order and state cases. They do not come out of the AI's decision table. As covered in the morning, the gaps that AI has trouble finding gather around order and state
- 3 acceptance criteria. Write, as counts, what has to happen for you to accept the import. You do not need to know the numbers yet. Write how to count them

Resulting files and where to save them

- `tpro-work/testcases.md` gets cases from I-01 in the Import section. They become the basis for the implementation in Exercise 9 and turn into tests in Exercise 10. The 3 order and state rows become E2E scenarios on Day 3
- `tpro-work/open-questions.md` gets the acceptance criteria AC-1 to AC-3 in the Awaiting approval section

The flow you went through here has 4 steps: you write the conditions, produce the decision table, add rows to the test case table, and check against the requirement IDs. In the Day 3 skill exercise, you turn these 4 steps into a skill as they are.

Check your result

- The result column of the decision table has requirement IDs, and combinations that cannot be decided are PENDING
- The Import section has cases from I-01 and includes the 3 rows you wrote
- Expected results are written as a count or as text shown on the screen
- `open-questions.md` has 3 acceptance criteria under Awaiting approval, and each can be judged by a count
- Rows that come from acceptance criteria have Source set to Acceptance criteria (AC-n), not Spec

If it does not work

Symptom	Fix
The decision table grows to 60 rows	You do not need every combination. Change one condition at a time, and combine only the 2 related ones, the same row twice and the total for one day
Days with a total over 24 become Not imported	Have it read the R-07 sentence again. It says whether to import the row and where to show a warning

Symptom	Fix
Holidays are judged by weekends only	You did not give it <code>#holidays_2026.csv</code> . Attach it and have the AI produce the table again
The acceptance criteria end up as "imports correctly"	Rewrite it as how many rows are imported and how many rows appear in the warnings

If time allows

- Write cases for how to handle a name that is not in category-map.csv, in 2 ways: not imported, as the specification says, and added to the map later by the client

Exercise 7: Create 30 rows of test data as CSV

Scenario

When people make test data by hand, all 30 rows end up as normal cases. Boundary value rows are tedious, so they get put off, and in the end they are the ones that are missed. Once, using the columns of the handout app, you write a prompt that sets the number of rows and their distribution first and then has the AI create the data.

CSV is a table-style text file in which values are separated by commas. The first line holds the column names.

Normal cases, boundary values and error cases are another way to say equivalence classes and boundary values. A normal case is a value that fits plainly into any equivalence class. For hours, it is a middle value such as 8. A boundary value sits at the edge of a group. For hours, these are 0.5 and 24. An error case is a value outside the equivalence classes. For hours, these are 0 and 24.5, plus a blank and full-width digits. If you do not specify the breakdown, you get bland data that is all normal cases.

Steps

1. Copy the following text into the chat. If the test case table is still in the earlier thread, continue in the same thread.

```
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
```

2. Adjust the hours condition to your own test case table. If any of 0, 0.5, 8 and 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 the desktop.
4. Count the rows. When you open data.csv, line numbers appear at the left edge. If there are not 31 lines, name the missing breakdown and ask for more.
5. Check by eye that the hours column has 0 and 24, and that the kind column has only the 4 values Design / Implementation / Review / Testing.
6. Write one line about an item that differed from your instructions in data-memo.txt, in the tpro-work folder. Write how many rows were missing, or which condition was ignored.

Do not trust the returned CSV by eye. Count the rows. The main point of this exercise is cross-checking, not generating.

Check your result

- data.csv has 31 lines (1 header + 30 data rows)
- The hours column has at least one row each of 0, 0.5, 8 and 24
- kind has only the 4 values Design / Implementation / Review / Testing
- memo contains fictional task descriptions (no real company names)
- data-memo.txt has one line about where the result differed from your instructions

If it does not work

Symptom	Fix
You ask for 30 rows and it stops at 15	Send "Continue with the rest" and the rest comes out. Count the rows each time
Explanatory text appears before and after the code block	You can delete it by hand. The last line of the prompt reduces it, but it does not disappear every time
Values such as 1.25 appear in hours	The cause is that the conditions did not say 0.5 steps. Add "Use values in 0.5 steps only"
Research or Meeting appears in kind	The instruction is sometimes ignored even when you specify it. Name the problem, have it fixed, and write the fact in data-memo.txt
You saved the file, but it does not appear in File Explorer	The save dialog was pointing to a different folder. Choose tpro-work on the desktop again
It is hard to count the lines	When you put the cursor on the last line, the line number appears in the Status Bar at the bottom right

Most people do not get 31 lines. Not even reaching half is normal, so just name the missing breakdown and ask for more.

If time allows

- Ask it to add 3 rows where 2 records are registered for the same category on the same day, and widen the data to include duplicates
- Have the AI calculate the total hours for each category, then cross-check one category yourself
- Ask it to mix 3 rows with dates from next month into date, and add future-date data

Exercise 8: Produce the test report as a PDF

Scenario

Work that ended at the design stage is the hardest to explain when you report to your manager. If you have one page that gives the number of cases and the reason they have not been run, you can explain progress orally. Once, you take the test case table through to a test report saved as a PDF.

Today's report has no pass/fail column. You have not run anything, so there are no numbers to write. You write 5 items: overview, target feature, number of designed cases, not run and why, and findings.

Steps

1. Attach testcases.md to the chat, have the AI write report.md with the following text, and save it in the tpro-work folder on the desktop.

```
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
```

2. Open testcases.md, count the rows, and check that the total and the counts by priority in the report match. If they differ, fix them by hand. When you count, exclude the header row and the separator row.
3. Ask "Turn this Markdown into one HTML page and include CSS for printing", and save it as report.html in the tpro-work folder. Have the print CSS include a rule that keeps table rows from splitting across pages, and a margin setting.
4. Open report.html by dragging it into a browser window, and press Ctrl+P. Set the destination to "Save as PDF", and save it as report.pdf inside the tpro-work folder.
5. In the preview, check the right edge of the table and the position of the headings. If the table is cut off, ask Copilot to set the paper to landscape or to add CSS that makes the text smaller.

Do not skip Step 2. If you skip it, this exercise becomes just practice in converting files.

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

Check your result

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

If it does not work

Symptom	Fix
The print preview is blank	Check "Background graphics" under "More settings"
report.html opens in a different browser when you double-click it	Open it by dragging it into the window
The counts the AI wrote do not match the actual counts	This happens often. Overwrite them with the counts you made. You may also write in the Findings that they did not match
The table is cut off at the right edge	First, set the paper to landscape. If the table is still cut off, reduce the columns. You do not need to put every column in the PDF
A heading is left at the bottom of a page	Ask Copilot to add CSS that prevents a page break right after a heading

Today, you make the PDF entirely with the print function of the browser. You can also use the Markdown PDF extension, but follow your workplace rules on whether you may install it. If you do not configure the setting, the extension tries to download a browser for the conversion, and on a company network it stays stuck. If you use it, enter the path of the Edge executable on your PC in the markdown-pdf.executablePath setting.

If time allows

- Add one more table to the report with only the rows whose priority is High
- Add a 3-line summary at the top of the report (target feature, number of designed cases, number of cases you added by hand)
- Have the AI make 2 versions from the same report.md, one for internal readers and one for the client, with a different level of formality, and compare them

How to split files so they work over file://

If you add the import logic directly to the script in `index.html`, you have to open the page and drop a file onto it by hand every time you check. If you extract the logic into functions and split them into files, `tests.html` can call the same functions. However, two things stop working on a page opened by double-click, that is, over file://.

What stops working	Reason and workaround
<code>fetch</code>	The browser cannot go and read files on your PC. This is why the Day 1 instructions say to receive files by drag and drop
A <code>type="module"</code> import	The browser blocks it because of CORS. Instead, share the code through the load order of the script tags, and put the functions together under one name, <code>window.WorkLogCore</code>

The category map and the holidays cannot be read from CSV files either, so copy them into `worklog-config.js` as constants. The load order is as follows.

```
lib/papaparse.min.js
worklog-config.js
worklog-core.js
Screen script (inside index.html)
```

The 2 import stages and the contract

Split the import into 2 stages: a stage that turns the file into a string, and a stage that turns the string into records. Name the second stage `WorkLogApp.importText(text, fileName)`. Put the character encoding detection only in the first stage. The tests and the E2E test run from the second stage, so if you do not split here, you run into trouble with `e2e.js` on Day 3 and with `Shift_JIS` in the practical exam.

The other is the contract. For each function, write in one line what it takes, what it returns, and what it returns when it cannot read the input. If you leave out the unreadable case and let the AI decide, you end up with a mix of functions that throw an exception and functions that return null.

If you run a local server, you can use both `fetch` and `modules`. However, today's PCs have no administrator rights and no Node. Today's condition is to build as if you cannot run a server. In your Day 1 `copilot-instructions.md`, there is one line that contradicts this way of splitting files. Find it before Exercise 9.

Exercise 9: Contract and import implementation

The working time is [24min]. The breakdown is: rewrite the instructions file [3min], write the contracts [7min], have Plan make a plan [2min], have Agent implement round 1 [4min], have Agent implement round 2 [4min], check on the screen [4min].

Purpose

Your Day 1 `copilot-instructions.md` says to write everything in a single `index.html` and not to split the file. This directly contradicts today's way of splitting files, so if you do not fix the instructions first, Agent pushes everything back into one file. If you added `AGENTS.md`, the line about the structure carries the same assumption. Instructions are not finished once you put them in place. You fix them when the assumptions change. Here you get one real example of that.

After you fix the instructions, write the contracts for the 6 functions yourself, have Plan make a plan, and then have Agent implement in 2 rounds. Round 1 is `worklog-core.js` and `worklog-config.js`, up to the check of each row. Round 2 is the import area and the warning area in `index.html`. If you read the diff one round at a time, you read half as much at each Keep. For `detectIssues` (R-06 to R-08), which checks the relation between rows, write only the contract and leave the body empty. You build the tests first and fill it in during Exercise 11.

You check more than the new import. You also check 2 more things: whether the filter and the total you added on Day 1 still work as before, and whether the hours check now applies to manual input as well. This is what R-12 requires.

What you bring from the earlier exercises

`testcases.md` from Exercise 6 (the import section), the Day 1 `copilot-instructions.md`, your own `index.html`, and whether `addRecord` in `index.html` has an input check.

Files you use

- `tpro-work/.github/copilot-instructions.md`, and `tpro-work/AGENTS.md` if you added one
- `tpro-work/docs/core-contract.md` You create this file.
- `tpro-work/worklog-core.js` and `tpro-work/worklog-config.js` Agent creates these.
- `tpro-work/index.html`
- `day2/docs/import-spec_v1.md`, `day2/docs/category-map.csv`, `day2/docs/holidays_2026.csv`

Steps

1. Rewrite the "Do not split the file" line in `copilot-instructions.md` into 2 lines yourself, and save. Logic goes in `worklog-core.js`, and the screen goes in `index.html`. Do not use `type="module"` or `fetch`. If you added `AGENTS.md`, rewrite the line about the structure into the same 2 lines [3min]
2. Create `docs/core-contract.md`, and write the contracts for the 6 functions, one line per function. They are `parseCsvText`, `normalizeDate`, `parseHours`, `normalizeKind`, `sumByKind` and `detectIssues`. Add to the contract that warnings are returned with the warning codes of R-09 in the Import Specification [7min]
3. Set the mode to Plan, and have it present the plan in 2 rounds. Check that the plan keeps the existing `render` and the field names of records [2min]
4. Use the hand-off button that appears below the plan to pass it to Agent, and have it implement round 1 only. Read the diff and Keep it, then compare the number of entries in the category map and the holidays in `worklog-config.js` with the number of rows in the original CSV files [4min]
5. Ask Agent for round 2: the import area, the warning area and the check on manual input in `index.html`. Read the diff and Keep it [4min]
6. Reload the browser and check 4 things: the filter, the total by category, entering 0.25 by hand, and entering 24.25 by hand [4min]

Write each line of the contract in the following form. Fill in the parentheses yourself.

```
functionName(input) → return value. If the input cannot be read, it returns (_____)
```

Include the following elements in the request you send to Plan.

Element	What you write
Attachments	<code>#core-contract.md</code> , <code>#import-spec_v1.md</code> , <code>#index.html</code> , <code>#category-map.csv</code> , <code>#holidays_2026.csv</code>
Requirements to include	R-01 to R-05, R-09, R-10 and R-12. Do not include the export of R-11 yet
How to split the rounds	Round 1 changes only <code>worklog-core.js</code> and <code>worklog-config.js</code> , and round 2 changes only <code>index.html</code>
What to leave for later	Place <code>detectIssues</code> so that it returns an empty array, as the contract says
What you want back	For each round: the files to change, the functions to add and where to put them, what disappears from <code>index.html</code> , and the effect on the existing <code>render</code> and on records

The request for round 2 includes the following conditions. Split the logic into a function that turns the file into a string and `WorkLogApp.importText(text, fileName)` , and detect the character encoding only in the first function. Give the load order. In the warning area, show one warning per line, with the row number, column, warning code and reason. Do not delete the existing `render` , and do not change the field names of records: date, kind, memo and hours. Apply the check to adding a record by hand as well, and put the check in one place only, `parseHours` .

What to think through yourself

- How you rewrite the conflicting line in `copilot-instructions.md`
- The contracts for the 6 functions. In particular, whether a function returns null when it cannot read the input, or returns a result with a reason and a warning code
- If you have a check function from Day 1, which one you keep and where you consolidate the checks

Resulting files and where to save them

- `tpro-work/docs/core-contract.md` The contracts for the 6 functions. Pass it with # for the plan with Plan, for test generation in Exercise 10, and for the plan in the practical exam. In Exercise 12, you add one line for `toCsv`
- `tpro-work/worklog-core.js` `detectIssues` stays empty. `tests.html` in Exercise 10 reads this file
- `tpro-work/worklog-config.js` The names of the original files you copied from go at the top
- `tpro-work/index.html` An import area and a warning area are added. `e2e.js` on Day 3 calls `WorkLogApp.importText`
- `tpro-work/.github/copilot-instructions.md` The 2 lines you rewrote

Check your result

- `copilot-instructions.md` no longer has the "Do not split the file" line. If you added `AGENTS.md` , the line about the structure there no longer has it either
- All 6 lines in `core-contract.md` say what to return when the input cannot be read

- The number of entries in the category map and the holidays in `worklog-config.js` matches the number of rows in the original CSV files
- The Day 1 filter and total by category work both before and after an import
- The results of entering 0.25 and 24.25 by hand are as written in R-03 and R-12
- Each row in the warning area shows a warning code from R-09

If it does not work

Symptom	Fix
Agent writes everything in the script of <code>index.html</code>	First, you may have forgotten to save <code>copilot-instructions.md</code> . Look at the dot on the tab. If it is saved, the line about the structure in <code>AGENTS.md</code> still assumes that everything fits in one file. Fix both, and then send the request again in a new chat
0.25 entered by hand is rejected	The Day 1 check function is still there. Ask Agent to consolidate the check into one place, <code>parseHours</code> , and describe only the symptom
The screen is blank, and the Console shows a line about CORS	<code>worklog-core.js</code> is loaded with <code>type="module"</code> . Have Agent read line 2 of the instructions file and fix it
Papa is not defined	The load order is reversed. Put <code>lib/papaparse.min.js</code> first
After an import, the numbers in the total are joined side by side	The hours field is being added as strings. This is the same symptom as in the Day 1 exercise on the total. Describe only the symptom and have Agent fix it
In round 2, Agent rewrites <code>worklog-core.js</code> as well	If the diff includes <code>worklog-core.js</code> , do not Keep it. Put "only <code>index.html</code> " at the start of the request, and send it again

If time allows

- Duplicate the `data.csv` from Exercise 7 as `data_r01.csv`, rewrite only the first line to Date,Category,Description,Hours, and then drop it onto the page. If you drop the original `data.csv` as it is and the whole file cannot be read, that is the correct behavior according to R-01. Write one line about that in `decision-log.md` too

Recovery slot

The slot is [5min]. If `worklog-core.js` is not working, `tests.html` in the next step has nothing to call. Use this time to bring everyone to a state where the import works.

If your import is not working, write in the Zoom chat whether you stopped at stage 1 or stage 2, and describe the symptom in one line. The instructor replies with just one next step. If it still does not work, the instructor sends you the instructor version of the stage where you stopped in a private chat. For stage 1, it is `worklog-core_stage1.js` and `worklog-config.js`. For stage 2, it is the script you paste into `index.html`. Rename `worklog-core_stage1.js` to

`worklog-core.js` and put it in place. When you receive it, rewrite `core-contract.md` to match the function names and return values in the instructor's file, and leave one line about the change in `decision-log.md`. The tests are written against the contract, so do not leave it out of sync.

If you used the instructor's file, leave one line in `decision-log.md` with the name of the file you used.

Exercise 10: From `assert.js` to `tests.html`

The working time is [21min]. The breakdown: have Ask produce `assert` and read it [5min], have the skeleton of `tests.html` built [3min], write the test for hours as a string [4min], have Ask write the remaining tests [4min], and compare [5min].

Purpose

Participants' computers have neither Node nor a test runner. If there is none, you build one. Have Ask produce `assert`, then read its branches yourself to check them. A function that takes the expected value, the actual value and the case ID and prints the result in one line is enough. Put into these 5 lines a line that counts it as HOLD when HOLD is passed as the expected value. `tests.html` is a page that lists them and counts PASS, FAIL and HOLD.

Write the first test yourself. Do you remember the case in the Day 1 total exercise where the total came out joined side by side like `1.52.5`? It is a test that passes records whose hours are still strings to `sumByKind`. Have Ask write the rest. Give it the rows whose Layer is Unit, regardless of Source. Turn the Guess rows and the HOLD rows into tests too, and mark their case IDs.

If you have Agent write them, it goes and reads `worklog-core.js` on its own and copies the expected values from the implementation. Even with Ask, the open tabs and the current file shown in the input box are attached silently. Remove them first, then send.

What you bring from the earlier exercises

These are `worklog-core.js`, `worklog-config.js` and `docs/core-contract.md` from Exercise 9, and `testcases.md`, to which you added Source and Layer in Exercises 5 and 6.

Files you use

- `tpro-work/tests/assert.js` You create it from scratch.
- `tpro-work/tests.html` Agent creates it.
- `tpro-work/tests/core.tests.js` You create it from scratch.
- `tpro-work/worklog-core.js`, `tpro-work/worklog-config.js`
- `tpro-work/testcases.md`, `tpro-work/docs/core-contract.md`

Steps

1. Have Ask produce assert. It is a function that takes the expected value, the actual value and the case ID, prints PASS if they match or FAIL with both values in one line if they differ, and counts it as HOLD if the expected value is HOLD. Read it to check that all 3 branches, PASS, FAIL and HOLD, are there, and save it to `tests/assert.js` [5min]
2. Have Agent build the skeleton of `tests.html`. Have it show the PASS, FAIL and HOLD counts at the top, along with the SUMMARY line used for submission. Read the diff and click Keep [3min]
3. Pass records whose hours are still strings, such as "3" or "4.5", to `sumByKind` in a test. Write the 3 lines yourself in `tests/core.tests.js`. Open `tests.html` in the browser and check the result [4min]
4. Close the tab of `worklog-core.js` and remove the attachment of the current file shown in the Ask input box. Give it only `#testcases.md` and `#core-contract.md`, and have it turn every row whose Layer is Unit into a test with a case ID. First have it output a table that restates the expected results as return values of the contract, then have it output a code block. Paste the code block it gives you into `tests/core.tests.js` and save it [4min]
5. Check that `worklog-core.js` is not in References of the reply. Pick 3 of the generated tests and compare: does the restated expected value give the same pass or fail as the expected result in `testcases.md`, and does the call match the contract? Also check that the Guess and HOLD rows are marked. If any value is different, send back the case ID and have it rewrite the test [5min]

Put the following in the prompt for `tests.html`: the load order, that `assert.js` must not be rewritten, how to show the counts and the SUMMARY line, that no external libraries are added, and that `type="module"` is not used. The load order is as follows.

```
lib/papaparse.min.js
worklog-config.js
worklog-core.js
tests/assert.js
tests/core.tests.js
```

The SUMMARY line is pasted as it is into the practical exam submission, so fix its format here. IDs are separated by commas. Have it shown at the top of the page in a form you can select and copy.

```
SUMMARY PASS=n FAIL=n HOLD=n IDS=executed_case_IDS FAIL_IDS=FAIL_case_IDS HOLD_IDS=HOLD_case_IDS
```

Put the following elements in the prompt for Step 4.

Element	What you write
Attachments	Only <code>#testcases.md</code> and <code>#core-contract.md</code>
Target	All rows whose Layer is Unit. Source does not matter
Call format	<code>assert(expected, actual, caseId)</code> . One call per case, one code block
Expected value	The expected results in <code>testcases.md</code> are on-screen sentences and counts, so restate them as the contract's return value (one warning code or one value). Do not change the pass or fail outcome or the range. Have it output the table of restatements first

Element	What you write
Mark	For a Guess row, add <code>?</code> to the end of the case ID. For a HOLD row, add <code>!</code> . For a row whose expected result is PENDING, make the expected value the string PENDING
What not to show	Do not refer to the contents of <code>worklog-core.js</code>

In a HOLD row the expected value is PENDING, so it FAILs with any implementation. That is correct. It is the undecided part of the specification showing up on screen as a FAIL.

What to think through yourself

- When assert fails, what should it output so that the person fixing it is not stuck? Read the generated function to see whether it outputs that
- When a Guess row FAILs, do you expect the fix to go to the implementation or to the test, as things stand now? Compare with the 3 conflict lines in `decision-log.md`

Resulting files and where to save them

- `tpro-work/tests/assert.js` The function you had Ask produce and checked by reading it yourself
- `tpro-work/tests.html` A test runner you built yourself that runs without Node. Paste the SUMMARY line into the practical exam submission
- `tpro-work/tests/core.tests.js` Tests with `?` on the Guess rows and `!` on the HOLD rows. In Exercise 11 you triage the FAILs, in Exercise 12 you add regression tests, and in the practical exam you add tests in the new format

Check your result

- When you open `tests.html` in the browser, PASS and FAIL are listed with case IDs, and the counts and the SUMMARY line appear at the top. The SUMMARY line has FAIL_IDS and HOLD_IDS
- `assert.js` is still the 5 lines you wrote yourself, and it counts an expected value of HOLD as HOLD
- There is a test that passes hours as a string
- Every row whose Layer is Unit is a test, with `?` on the Guess rows and `!` on the HOLD rows
- Every restated expected value gives the same pass or fail as the expected result in `testcases.md`
- `worklog-core.js` does not appear in the References of Ask

If it does not work

Symptom	Fix
All of them throw <code>TypeError</code>	The calls do not match the contract. You probably did not give it <code>#core-contract.md</code> . Give it and have it generate the tests again

Symptom	Fix
<code>tests.html</code> is blank	The cause is the load order. Press F12 and look at the first line in the Console tab
<code>worklog-core.js</code> appears in References	It read the implementation and copied the expected values. Discard that reply, close the tab, remove the attachment, and send the prompt again in a new chat
All the R-06 to R-08 rows are FAIL	Because <code>detectIssues</code> is still empty. You add it at the start of Exercise 11, so this is correct for now
It corrects the expected values of the Guess rows to match the specification and gives them back	Put "Do not fix them" at the start of the prompt and send it again. You fix them in Exercise 11

If time allows

- Add a version to assert that compares arrays and objects by their contents. This lets you compare the whole return value of `sumByKind`.

Exercise 11: Triage of FAILs

The working time is [18min]. The breakdown: add `detectIssues` [5min], decide where to fix [3min], have the implementation fixed [3min], handle the test item and the client item [3min], and compare [4min].

Purpose

With the tests already in place, add `detectIssues` (R-06 to R-08), which looks at relationships between rows. Watch the R-06 to R-08 rows change from FAIL to PASS, then triage the remaining FAILs.

There are 3 places a fix can go. If the implementation differs from the specification, fix the implementation. If the test's expected value was a guess, fix the test. If the specification does not decide either way, ask the client. The remaining FAILs come from 3 sources: implementation bugs, Guess rows (?) and HOLD rows (!).

If you give a FAIL to the AI and ask it to fix it, it usually changes the test to match the implementation. Then an implementation that differs from the specification is locked in as PASS. Pick one item from each of the 3 destinations, decide where to fix it yourself, and then ask.

What you bring from the earlier exercises

These are `tests.html` and `tests/core.tests.js` from Exercise 10, the 3 conflict lines you wrote in `decision-log.md` in Exercise 5, and `open-questions.md`.

Files you use

- `tpro-work/tests.html`, `tpro-work/tests/core.tests.js`, `tpro-work/tests/assert.js`

- `tpro-work/worklog-core.js` , `tpro-work/docs/core-contract.md`
- `tpro-work/testcases.md` , `tpro-work/open-questions.md` , `tpro-work/decision-log.md`

Steps

1. Give Agent the `detectIssues` line in `#core-contract.md` and R-06 to R-08 in `#import-spec_v1.md` , and have it implement only `detectIssues` in `worklog-core.js` . Have it return warnings with the warning codes of R-09, and do not let it rewrite anything under `tests/` or `index.html` . Click Keep, reload `tests.html` , and check that the R-06 to R-08 rows changed to PASS [5min]
2. From the remaining FAILs, pick one each that belongs to the implementation, the test and the client. Write where to fix it and why, one line each, in `decision-log.md` . Compare them with the 3 conflict lines from Exercise 5 [3min]
3. Have Agent fix the one you decided belongs to the implementation. Write which case ID is FAIL, which requirement ID the expected value comes from, and which function to fix, and do not let it rewrite anything under `tests/` . Read the diff and click Keep [3min]
4. For the one you decided belongs to the test, have it correct the expected result in `testcases.md` from the requirement in the specification, fix the test with the same value, and remove the `?` . Move the one you decided belongs to the client to `open-questions.md` , and have it set the test's expected value to HOLD. Do not let it change `worklog-core.js` . Read the diff and click Keep [3min]
5. Reload `tests.html` and compare the 3 items one by one to check that each gives the result you decided. Verify with FAIL_IDS and HOLD_IDS in the SUMMARY line [4min]

What to think through yourself

- Where each of the 3 FAILs should be fixed: the implementation, the test or the client
- For the item you decided to fix in the test, can you say in one line why it is not the implementation?

Resulting files and where to save them

- `tpro-work/worklog-core.js` It gets the body of `detectIssues` and the fix for the item routed to the implementation
- `tpro-work/tests/core.tests.js` It gets the fix for the item routed to the test, and the row you set to HOLD
- `tpro-work/decision-log.md` 3 lines are added about where to fix and why
- `tpro-work/open-questions.md` 1 item that you sent to the client is added. Write it the same way as the rationale in the judgment table of the practical exam

Check your result

- The R-06 to R-08 rows change to PASS after you add `detectIssues`
- `decision-log.md` has where to fix and why for each of the 3 items: implementation, test and client
- For the case where you fixed the implementation, the test file has not changed
- HOLD is counted separately from FAIL and appears in HOLD_IDS in the SUMMARY line
- After the reload, the 3 items show the results you decided on

If it does not work

Symptom	Fix
Agent rewrote the test to match the implementation	If the diff includes files in <code>tests/</code> , do not Keep it. Put "Do not rewrite the tests" at the start of the prompt and send it again
You cannot decide whether to fix the implementation or the test	Look at the Source column of that row. If it says Spec, fix the implementation. If it says Guess, fix the test
You want to route a HOLD row to the implementation or the test	Look at the Source column and <code>open-questions.md</code> , and decide for yourself. The instructor will not give you the answer
There is no FAIL routed to the implementation	This happens when the AI's implementation matched the spec. Raise your hand. The instructor sends you a few lines by private chat that temporarily replace <code>parseHours</code> . You only add 1 line at the end of the loading section of <code>tests.html</code> , and you do not touch <code>worklog-core.js</code> . When you finish the triage, delete that 1 line
There is no FAIL routed to the test	Copy the 3 rows C-06, C-08 and C-11 from <code>day2/fallback/testcases_sample.md</code> into <code>testcases.md</code> , renumber their IDs to F-06, F-08 and F-11, set Source to Guess, turn them into tests, and run the tests again. This keeps the IDs from overlapping with your own C-rows

If time allows

- Duplicate the `data.csv` from Exercise 7 as `data_r01.csv` , rewrite only the first line to Date,Category,Description,Hours, and then drop it onto the page. If you drop the original `data.csv` as it is and the whole file cannot be read, that is the correct behavior according to R-01. Write one line about that in `decision-log.md` too

The 2 gaps in display and export

Once you added the import, this app began to show strings written by other people on the screen. In the September participant survey, participants asked for security to be taught through concrete mechanisms. You will see the gaps as they happen in your own app, not in someone else's sample.

Gap	What happens	How to stop it and the cost
Display	A string put into <code>innerHTML</code> is interpreted as HTML. If the Description field contains <code></code> , the script in the attribute runs	With <code>textContent</code> , the string is shown as plain text. The <code>render</code> in the Day 1 handout uses <code>textContent</code> , but the totals table the AI added and the warning area you added today may not

Gap	What happens	How to stop it and the cost
Export	If a CSV value starts with <code>=</code> , <code>+</code> , <code>-</code> or <code>@</code> , Excel reads it as a formula. This is called CSV injection	If you add <code>'</code> at the start, the formula stops working. However, the value changes when you read the exported CSV back

String concatenation in SQL has the same root: it lets a value from outside be interpreted as a command. The Worklog app has no SQL, so you look at it in the server-side code `review_target.js`.

Participants often say, "Just add DOMPurify." The handouts include only 3 libraries, and the Day 1 instructions say not to add external libraries. If you only need to show text, `textContent` is enough. If you add Content-Security-Policy with a meta tag, it can block inline event attributes even over `file://`, but today we only mention the name.

Exercise 12: Add export and close the gaps

The working time is [23min]. The breakdown is: drop the file and look [2min], write review points and show them to Ask [5min], decide how to handle them and have them fixed [5min], check with Excel and by re-importing [4min], add regression tests [4min], write `security-notes.md` [3min].

Purpose

`day2/data/worklog_2026-07-08.csv` is data for July and August exported from the legacy timesheet system. The Description column contains rows with tags, rows that start with `=`, `+` or `@`, and valid adjustment rows that start with `-`. One example is an adjustment for the previous day. First, drop the file onto your app and see what happens in it.

Leave the reviewing to Ask. Ask does not rewrite files, so findings and fixes do not get mixed. Agent and you do the fixing. The valid rows that start with `-` need your judgment. If you add `'`, Excel shows them as text, but the values change when you re-import the exported CSV. You decide which to choose after you look at the real result.

What you bring from the earlier exercises

`worklog-core.js`, `index.html`, `docs/core-contract.md` and `tests/core.tests.js` as they stand after Exercise 11.

Files you use

- `day2/data/worklog_2026-07-08.csv`
- `day2/review/review_target.js`
- `tpro-work/index.html`, `tpro-work/worklog-core.js`, `tpro-work/docs/core-contract.md`
- `tpro-work/tests/core.tests.js`
- `tpro-work/security-notes.md` A new file that you create

Steps

1. Drop `worklog_2026-07-08.csv` onto the app and see how the rows with tags in Description appear in the list, the warning area and the totals table [2min]
2. Write 3 lines of review points you want checked. Give Ask `#index.html` , `#worklog-core.js` and `#review_target.js` , and have it output findings only [5min]
3. Decide for yourself, in 2 lines, how to handle valid descriptions that start with `-` . Ask Agent to implement the R-11 export `toCsv` and the export button, and to fix the display spots that Ask flagged. Also have it add 1 line to `core-contract.md` for `toCsv` . Read the diff and Keep it [5min]
4. Open the exported CSV in Excel and see how the rows that start with a symbol look. Drop the same CSV onto the app again and check whether the Description values are the same as the original [4min]
5. Write the expected values for the 2 regression tests yourself, and have Agent add them to `tests/core.tests.js` . The inputs are a record whose Description is `=SUM(A1)` and a record whose Description is `-0.5h adjustment for the previous day` [4min]
6. Create `security-notes.md` and copy in the 3 lines of review points that you sent in Step 2. Write 2 lines on how you plan to fix the SQL string concatenation in `review_target.js` [3min]

Include the following elements in the prompt for Step 2.

Element	What you write
Attachments	<code>#index.html</code> , <code>#worklog-core.js</code> , <code>#review_target.js</code>
Review points	The 3 lines you wrote
Output format	A table with 5 columns: file, line, review point, why it is dangerous, direction for the fix
What to stop	Do not output code

In the prompt for Step 5, always include this: use only the expected values you wrote yourself, and do not copy them from `worklog-core.js` .

What to think through yourself

- The 3 lines of review points you show Ask. What to show and what not to show
- Whether to protect valid descriptions that start with `-` with `'` , or only wrap them in quotation marks. Whether to accept the change in the value when you re-import
- 2 lines on the approach to fixing the SQL string concatenation

Resulting files and where to save them

- `tpro-work/worklog-core.js` gets `toCsv` . `docs/core-contract.md` also gets 1 more line
- `tpro-work/index.html` gets the fixes to the display spots and the export button. In the Day 3 E2E test, you run the whole flow from import through export to reload
- `tpro-work/tests/core.tests.js` gets 2 more regression tests

- `tpro-work/security-notes.md` holds the 3 lines of review points and the 2 lines on the fix approach. In the Day 3 skill exercise, the 3 lines of review points become the body of `security-reviewer.agent.md`

Check your result

- A Description of `<img src=x onerror=...>` is shown as plain text and the script does not run
- Even when you open the exported CSV in Excel, descriptions that start with `=`, `+` or `@` do not run as formulas
- Valid descriptions that start with `-` look in Excel the way you decided to handle them, and you can explain how the values change when you re-import
- The 2 regression tests PASS
- `security-notes.md` has 3 lines of review points and 2 lines on the fix approach

If it does not work

Symptom	Fix
Everything lands in 1 column when you open the file in Excel	The cause is the delimiter or the character encoding. Check whether the export is UTF-8 with a BOM. Do not dig deeper. Switch to looking at the first characters in VS Code instead
Excel is not installed	Check how it looks on the instructor's screen share, and make only the decision on how to handle it yourself
After you add <code>'</code> , the <code>'</code> remains in the re-imported value	That is the cost. Decide whether to strip it on the import side or to use only quotation marks, and write it in <code>decision-log.md</code>
Ask outputs long blocks of the code it fixed	Put "Do not output code" at the end of the prompt and send it again
A dialog appeared on a row with tags	Finding it is the result in itself. Compare which file and which line has the <code>innerHTML</code> with the table of Ask's findings
The number of items in the list is higher than you expected	Check whether rows identical to records already in the list came in again. Check whether or not you wrote it in <code>open-questions.md</code> , and mark it HOLD for now. In the reload of the Day 3 E2E test, you will look at it again as a count

If time allows

- For each column you export, make a table showing whether it needs protection against a leading symbol. Check that if you add `'` to the Hours column, the values are no longer numbers when you read them back

Troubleshooting

These are problems that occur across exercises. For the symptoms of each exercise, see the table in its section.

Symptom	What to check
Copilot Chat does not open	Click the Copilot icon in the Activity Bar. If you cannot find it, you can also open it with Ctrl+Alt+I
Copilot does not respond, or the model list does not appear	You have been signed out. Check whether the icon at the bottom right of VS Code has a slash through it, then sign in again from the account icon at the bottom left.
No file suggestions appear when you type #	It is a full-width #. Switch to a half-width #.
A file you meant to attach is not being read	Check whether the file name appears above the input box. If it does not, the file is not attached
The answer ignores the existing structure	You did not attach the target file. Attach it so that Copilot reads it, then ask again.
The browser page does not change even though you saved	Either the file was not saved, or it is a cache problem. Make sure the dot next to the tab name is gone, then reload with Ctrl+Shift+R
The screen went blank	Press F12 and paste the first red line in the console into the chat as it is. Do not guess the cause
A feature that used to work is broken	Go back to the copy you kept, rewrite the prompt, and ask again
You do not know how far to go back	Go back to the rule of changing only 1 thing at a time. Decide what the 1 thing is before you ask
You cannot find the file you created	The file was saved somewhere other than tpro-work. Look inside tpro-work in the VS Code Explorer, and if the file is not there, save it again
You stepped away and lost track of the steps	Ask someone near you, or speak to the instructor individually during a break. You can rejoin partway through

Contact information for inquiries will be given on the day. During exercises and breaks, the instructor walks around the classroom. Speak to the instructor on the spot.

What you take home today

Today you worked with a division of work: the AI produces, and the person compares and decides. The AI produces test viewpoints, test case tables and tests in a few minutes. What remains with the person is whether they can say, line by line, where each expected result came from.

A passing test says only that the code behaved as the expected values you wrote. If the source of an expected value is a guess, passing is not evidence that the spec is met.

What to keep until the next session

- The tpro-work folder. Leave index.html, testcases.md, decision-log.md, data.csv, worklog-core.js, the tests folder and security-notes.md in place, and do not delete them. You will use them in the Day 3 E2E test and the practical exam
- Your Copilot sign-in. Day 3 is 1 week away. Do not sign out

On Day 3, you choose 4 cases from today's test case table and run them as E2E tests from import to reload, and you move today's steps into a skill and a custom agent. After that come the Literacy Check and the practical exam.

Rules to follow

You may put only 2 kinds of content into the chat: the contents of the Worklog app in the handouts, and text you wrote yourself. Do not open or paste a client's actual specifications or code. Follow these 4 rules.

1. Generalize client names, product names, project names and partner names to the industry or process level
2. Do not use real data, real files or screenshots
3. For numbers and specifications, avoid concrete values and write at the level of purpose, problem and expected effect
4. Before you submit, check for yourself that the content is safe to share outside your company

There is nothing to submit today. Even when a submission is required, you can submit only text that you have put into words. You cannot submit code or real data.

This material is a copyrighted work created by Givery, Inc. Participants may use it only for their own learning and for reference in their own work. Do not distribute it outside the company, post it on social media, or share it with third parties.

All data in this app is fictional.



Givery, Inc. Do not redistribute this material.