



HANDS-ON GUIDE

Hands-on AI-Assisted Development, Mechanical and Electrical track, Day 3 Hands-on Guide

Hands-on AI-Assisted Development, Mechanical and Electrical track, Day 3

This guide lists the 12 exercises of Day 3 in the order you do them, to keep at hand. The slides keep moving ahead, so when you stall, come back to the matching exercise in this guide. It does not give the full text of the prompts you send. It shows only the pattern of what to write, so fill in the content in your own words.

What you do today

Today's question is just one. From the 24 machines at Plant 2, decide which 1 to stop and inspect now, and put your evidence in a one-page report. You work on the assumption that you will bring it to next week's maintenance meeting.

There is one more goal, though. While you make that report, you save the prompts you type in the folder, in a form you can call up as they are next month. The prompts you typed on Day 1 and Day 2 are in the chat history now. When you do the same work next month, you would have to rebuild the same conversation from zero again. Today you fix that.

By evening you have 1 report and the 6 items that produce it. These 6 items together are called the harness.

Part	Steps	Minutes
Morning, part 1	Write 3 lines of constraints, clean up the Operations data, and narrow 24 machines to 1	[92min]
Morning, part 2	Turn the prompts you typed into a command, then move it into a skill	[87min]
Afternoon, part 1	Swap the output template and merge the 4 records from the floor into one table	[109min]
Afternoon, part 2	Turn the conversation that worked into a skill, split the drawing revision diff work across 2 agents, and take home a one-page report	[192min]

There are 3 breaks along the way. Even if you get stuck in the morning, you can start the afternoon from different material. Recovery slots of 4 to 5 minutes each are built in so you can catch up, so even if you fall behind, you can still get through to the end.

The 6 items you build today

You build them one at a time in the exercises. You do not need to copy them down now. Just look over the names. Next month, when you open the folder, these names are how you find them.

Order	What to build	What it does
1	AGENTS.md	3 lines of constraints that go directly in the work folder. They are added to every chat from then on
2	clean-operations.prompt.md	A command that fixes the data preparation of the monthly Operations sheet as one prompt
3	equipment-stop-report	A skill that picks the machine to stop and writes the report. The judgment rules and the output template go in the same folder

Order	What to build	What it does
4	<code>records.instructions.md</code>	Instructions that apply only when reading the records folder. They fix how to quote
5	<code>records-to-table</code>	A skill that turns the messy records into an 8-column table. A table of how people on the floor phrase things goes in the same folder
6	<code>drawing-diff.agent.md</code> and 1 more	Custom agents that divide the work between drawing revision diffs and checking the bill of materials against the procurement request

What you use

You use one folder. Keep using the `tpro-work` you made on Day 2. The data handed out today and the files you make today all go under it.

Preparing the work folder

1. Right-click the Day 3 handout ZIP and choose "Extract All". Two folders appear: `data` and `tools`
2. Move these 2 folders directly into the `tpro-work` folder on your desktop. Keep the files you made on Day 2 and do not delete them
3. In VS Code, choose File > Open Folder and select `tpro-work`. Select the folder, not a file inside it
4. If the title bar at the top of the screen shows `tpro-work`, you are ready. This is today's workspace
5. Set the mode dropdown at the bottom of the chat to Agent. If you leave it on Ask, a request to create a file does not go through
6. Directly under `tpro-work`, create a new file named `decision-log.md`. You add today's decisions to this file

If the title bar shows only a file name, you have opened a file, not a folder. Start again from "Open Folder". Opening the workspace decides where the `.github` folder that you create today goes. If you get this wrong, it causes trouble all afternoon.

Contents of the handout folder

Location	Contents
<code>data/equipment-data_2026Q3.xlsx</code>	Three months of data for the 24 machines at Plant 2. It has 3 sheets: Operations, Inspections and Defects. It is not in a form you can total as it is
<code>data/equipment-data_2026Q1.xlsx</code>	Another quarter from the same plant. You use it in Exercise 8, when you run through the same work as in the morning on your own
<code>data/records/daily-inspection_2026-07.md</code>	Inspection records written on the floor as they are. Partway through, the person in charge changes, and even the item names change

Location	Contents
data/records/fault-history_2026H1.md	Records of when machines stopped. The way Duration is written differs from row to row
data/records/handover-memo_2026-07.txt	A memo written at a handover of duties. At the end, it lists what the abbreviations stand for
data/records/phone-memo_fault-report.txt	Notes taken from a phone call you received. The parts corrected afterward were not overwritten and remain as they are
data/drawings/	It holds 2 drawings of the conveyor roller shaft, the current revision and old revision A, plus 1 matching bill of materials and 1 procurement request. The _svg folder holds the same 2 drawings written as text, and you have the AI read those
tools/01_data-loader.html	A tool that opens an xlsx in the browser and turns its contents into text. It uses parts in the lib folder, so if you move only the HTML file somewhere else, it does not work
tools/02_data-export_skeleton.html	It holds only the outer frame of the export tool. If you finish saving the CSV in Exercise 2 and have time left, fill in the TODOs with the AI, starting from TODO 1, and use the tool

If you open `equipment-data_2026Q3.xlsx` in Excel and save it over the original, the mess that the exercises work on disappears, and the exercise no longer works. To see the contents, always use `tools/01_data-loader.html`.

Turning an Excel file into text

A browser cannot open files on your PC by itself. You give it a file by dragging. Exercises 2 and 8 both start with these steps.

1. In File Explorer, open `tpro-work/tools` and double-click `01_data-loader.html`. Open it in the browser, not in VS Code
2. A dotted frame for dropping the file appears in the browser
3. From the `data` folder, drag `equipment-data_2026Q3.xlsx` and drop it into the dotted frame
4. The names of the 3 sheets, Operations, Inspections and Defects, are listed. Click the sheet name you want to see
5. Click "Copy this table" and that sheet is copied as tab-separated text. Paste it into the Copilot Chat input box

If double-clicking opens VS Code, right-click the file, choose "Open with", and select a browser. If you attach an xlsx directly to Copilot, you are asked for permission to run a script instead of having the contents read, and you get stuck. Everyone went through this once on Day 2.

The 4 layers of the harness

You move the preamble you type into the chat every time out of the chat and split it into 4 layers. You build all 4 layers today, so you do not need to memorize them now. Use this table to check where things go when you get stuck.

Layer	Location	What goes in	How it applies
instructions	<code>tpro-work/AGENTS.md</code>	Premises and prohibitions. Short instructions, one line each	Added every time, with no condition
Command	<code>.github/prompts/<name>.prompt.md</code>	One prompt. The parts that change can be arguments	Type a slash in the chat and call it by name
Skill	<code>.github/skills/<name>/</code>	Steps and assets. The judgment rules and the output template go in the same folder	The AI reads it based on its description. You can also call it yourself
Custom agent	<code>.github/agents/name.agent.md</code>	The role and the model to use. A limit on the tools it may use	Called when the parent decides it is needed

The further left you go, the less work it takes, and the further right you go, the more it can hold. `AGENTS.md` takes effect from the next chat once you write 3 lines. It takes less than 5 minutes to make, but it keeps applying in all the exercises after that.

The difference between a command and a skill is one file versus one folder. A command holds a single prompt, so if you want to add an output template or judgment rules, the body gets that much longer. A skill is a folder, so besides `SKILL.md` with the steps, you can put judgment rule and template files in the same folder and have the body read them by relative path. Move to a skill when you want the template and judgment rules to go along with it.

If you put a file in the wrong place, it is not loaded, however correct its contents are. None of the 4 layers needs any extra settings. The file name `SKILL.md` is in capital letters, and `AGENTS.md` goes directly at the workspace root. Remember just these 2 points.

The skill format follows the common specification at agentskills.io. Copilot also reads skills placed in `.claude/skills/` as they are. You can use the same assets across tools, so this helps when you roll them out inside your company.

Comprehension check 1

You created `AGENTS.md` under the `.github` folder. What happens?

- A Under `.github`, it is read in the same way
- B It is not read. It must go directly at the workspace root
- C It is read if you add the location in the settings
- D The one under `.github` takes priority

Read this after you have thought it through yourself. The correct answer is B. For `AGENTS.md`, the location itself, directly at the workspace root, is the condition. If you move it down even one level, it is not read. Some people start looking for a setting that adds a location to load from, as in C, but you do not touch it today. It is read by default. Even if the contents of the file are correct, it has no effect if the location is wrong. The same kind of failure appears again with skills and agents later.

Exercise 1: 3 lines of AGENTS.md

The work takes [5min]. The breakdown is: restate the constraint [2min], create the file and write 3 lines [2min], and check that it is read [1min].

Scenario

On Day 2, when you attached an `xlsx` as it was, you were asked for permission to run a script instead of having the contents read. Your PC did not let you grant the permission, so you got stuck there. The same thing happens today.

You can avoid it by typing "I will turn the contents of the file into text first and paste it" at the start of every chat. But you will not keep typing it every time. Write it in a file directly under the workspace and fix it there. You write 3 lines, and they apply to all chats from today on.

Steps

1. Write 1 line in your own words about why you got stuck on Day 2. Also write 1 line that restates it as a constraint from today on [2min]
2. Create a new file directly under `tpro-work` and save it with the name `AGENTS.md`. You do not need front matter [1min]
3. Fill in the parentheses of the template below in your own words, then write the 3 lines and save [1min]
4. Ask the chat any one question, and check that a line for `AGENTS.md` appears in References above the answer [1min]

- Convert files to (_____) before handing them over
- Do not ask for processing that assumes (_____) will run
- Do not fill in (_____) in numeric data

Keep each line to about 10 words. The longer a line is, the less it is followed. Write in the imperative or as a prohibition.

What to think through yourself

- 1 line, in your own words, about why you got stuck on Day 2
- 1 line that restates it as a constraint from today on
- 1 line of the preamble you type into the chat every time now. It does not need to be a polished sentence

Result

`tpro-work/AGENTS.md` is created. It is plain Markdown with just 3 lines. All the exercises after this proceed with these 3 lines in effect.

Check your result

- `AGENTS.md` is directly under `tpro-work`
- All 3 lines are written in the imperative or as prohibitions
- `AGENTS.md` appears in References of the chat reply

If it does not work

Symptom	Fix
It does not appear in References	Open a new chat. It may not be added to a chat that is already in progress
You created it under <code>.github</code>	Move it directly to the workspace root. It is not read in that location
You started looking for a setting	Do not touch it. <code>chat.useAgentsMdFile</code> is a setting you turn off when you do not want the file read, and it is on by default
You also use Claude Code and the results do not match	<code>CLAUDE.md</code> and <code>.claude/rules/</code> are also loaded. If you need to separate the causes, look there

If time allows

- Add one rule that applies only to the CSV files in the `data` folder to `.github/instructions/`, write `applyTo`, and check whether what References shows changes.

Exercise 2: Narrow 24 machines down to 1

The working time is [45min]. The breakdown: paste the data as text [4min], count the format problems [8min], fix the 3 types together [8min], check missing values [5min], check duplicates and empty rows [4min], save the file [5min], aggregate [6min], filter by conditions [3min], and write the memo [2min].

Scenario

The Operations data for the 24 machines at Plant 2, from July to September, takes up 83 rows below the header. From these, you decide which 1 machine to stop and inspect now. Each of you decides the conditions. Do not let the AI decide.

The Operations sheet keeps the kinds of format problems that are common in files from the shop floor, exactly as they are.

What to check	What is in the sheet
Columns	6 columns: Equipment ID, Equipment name, Month, Operating hours, Stops and Defects count
Rows 1 and 2	Row 1 is a title and row 2 is empty. The header is in row 3
Units	Units are stored as text, as in 186.4 h and 214.2 hours
Numbers	Some cells still contain full-width digits
Month	3 formats are mixed: 2026-07, 2026/8 and R8.9. R8.9 is September of Reiwa 8 (a Japanese era year)
Empty cells	Empty cells appear in each of Operating hours, Stops and Defects count
Duplicates and empty rows	Rows with identical content and rows with every cell empty are mixed in. The guide does not say how many there are

The prompts you type in this exercise become commands as they are in the next exercise. Type them knowing that you will save them to a file later.

Steps

1. Load the xlsx in `01_data-loader.html` , copy the Operations sheet as tab-separated text and paste it into the chat. Do not hand the file over directly, as written in `AGENTS.md` [4min]
2. After pasting, have the AI list only the types and the counts of the format problems. Do not have it fix anything yet [8min]
3. Have the AI fix the 3 types together: strings with units, full-width digits and Month formats. Have it list the changed cells with row numbers. Do not let it touch empty cells [8min]
4. Have the AI list the empty cells by column, and write the valid count for each column. Do not let it fill them [5min]
5. Have the AI list the duplicate rows and empty rows by row number before it deletes them, and then have it delete them [4min]
6. Have the AI save the result as `cleaned_q3_operations.csv` and report the number of valid rows and the valid count of Operating hours [5min]
7. Have the AI aggregate by Equipment ID into 3 columns: the mean of Operating hours, the sum of Defects count and the sum of Stops. Use the number of valid months as the denominator of the mean [6min]
8. Send the conditions you wrote earlier and have the AI list the machines that meet all of them [3min]
9. In `decision-log.md` , write 3 lines for the conditions, 1 line for the machine you chose, 2 lines for the candidates you dropped and 2 lines for how you treated the denominator [2min]

The prompt you send includes the following 5 elements. The instructor shows the full text on the screen. On your own screen, only check that these 5 are included.

Element	What you write
Scope	Which sheet and which columns to look at. State how much of the pasted table is covered
What to fix	Strings with units, full-width digits and Month formats. Name what you want fixed
What not to fix	Do not touch empty cells. If you leave this out, the AI fills them with the mean
Output format	List the changes in 4 columns: row number, column name, value before and value after
Where to save	Write the file name to save as, including the extension

If you have the AI clean everything in one pass, you cannot say which cell changed and why. If you count first and then fix, the evidence stays.

What to think through yourself

- Decide 3 filter conditions yourself and write them down before you ask the AI. If you add conditions after you see the sort order, they become reasons that fit the conclusion
- The 1 machine you chose and the 2 candidates you dropped, with the reason for dropping them
- 2 lines on how you treated the denominator when you calculated the mean. Include why you did not divide by the total number of rows

The way you take the denominator changes who ends up at the top. If you divide by the number of rows for 3 months, the mean of a machine that has values for only 2 months comes out lower than it really is. It is not that the machine was idle. Its records are just missing, yet it rises to the top of the candidates to stop. A number is produced anyway, so you cannot notice the problem unless you write down how you divided. This is the first thing people ask about in the meeting.

Result

You end up with `tpro-work/cleaned_q3_operations.csv` and 8 lines added to `decision-log.md`. You use this CSV until the afternoon report, so be sure to save it.

Check your result

- `cleaned_q3_operations.csv` is directly under `tpro-work`
- Empty cells are still empty, and not a single value has been filled in
- Duplicate rows and empty rows are removed, and you can say how many rows you removed
- The denominator of the mean of Operating hours is the number of valid months
- The 3 conditions were written in `decision-log.md` before you asked the AI
- `decision-log.md` has 8 added lines

It is fine if you cannot narrow it down to 1 machine. If several machines remain, compare the denominators of their means. You decide whether to add the treatment of the denominator to your conditions. If you do not add it, write that you could not narrow it down. Do not loosen the conditions to make the number fit.

If it does not work

Symptom	Fix
Everything was clean after the first pass	Always have the AI list the changed cells. If you cannot say which cell changed and why, it is not a basis for the report
A number was put into an empty cell	Add "Never fill in any values" and send the prompt again. If you go on with the cells filled in, the denominator of the mean changes and the ranking is reordered
The row count stays the same as the original	Duplicate rows and empty rows remain. Have the AI list the row numbers again
Too many machines show a value higher than the overall mean	The denominator is the total number of rows. Add an instruction to divide by the number of valid months and send the prompt again
The AI picked 1 machine	Do not let it choose. Decide yourself against your conditions
The reply is cut off partway	You pasted all 3 sheets together. Paste only the Operations sheet again

If time allows

- Apply the same prompt to the Inspections sheet and see what you must rewrite for it to work
- The Result column of the Inspections sheet contains variant spellings and typos. If you do not judge from the spec of 9.9-10.3mm and the measured sizes, you miss lots that are out of spec
- Load the saved CSV into the data loader and check that it gives the same table as the original sheet
- Open `tools/02_data-export_skeleton.html` in the browser and fill in the TODOs inside it with the AI, starting from number 1, to make a tool that turns CSV back into xlsx. Load `quality-inspection_2026H1.csv` and `maintenance-records_2026H1.csv` from `data`

Exercise 3: Turn a prompt into a command

The working time is [40min]. The breakdown: write a prompt yourself [6min], compare it with the sample [4min], send it and run it [5min], turn it into a command [4min], read the generated file [5min], call it from a new chat [5min], fix the differences [4min], call it again [4min], and look at the migration screen [3min].

Scenario

The Operations sheet arrives every month. You prepare the data the same way next month, and rebuilding the same conversation again is a waste. Save the prompt that just worked to a file so that you can call it from the chat with a slash.

The order matters in this exercise. Before you have the AI write anything, write 1 prompt on your own. Compare it with the instructor's sample and count the missing elements yourself. Then send it and run it.

Steps

1. In `tpro-work`, create `draft-prompt.md`. Recall the prompts you typed earlier and combine them into 1 prompt. Do not send it yet [6min]
2. The instructor shows a sample on the screen. Compare its elements with your own, count the missing elements and write them in `decision-log.md` [4min]
3. Add the missing elements, then send the prompt together with the Operations sheet and run it. You may also send it without adding them. The difference you get back is useful material as it is [5min]
4. In the same conversation, type `/create-prompt` in the input box. On the same line, write what you want: turn the prompt you just sent into a command named `clean-operations`, with a description saying it is used for monthly operations data preparation. Then send it. Choose Workspace as the save location [4min]
5. Open the generated `.github/prompts/clean-operations.prompt.md` and read it. It has 2 parts: the frontmatter, enclosed by lines of 3 hyphens, and the body below it. The text you wrote is in the body [5min]
6. Open a new chat, paste the Operations sheet and call the command with `/clean-operations`. Check that the result is the same as when you typed the prompt by hand [5min]
7. Fix the places where the results differ. Most of them come from the target sheet not being pinned down. Change the command so that it receives the sheet as an argument [4min]
8. Call the command again and check that you get the same result. Next, copy the Inspections sheet in `01_data-loader.html`, paste it into the chat, and then call the command with Inspections as the argument. Only the sheet name is passed as the argument, not the contents of the table. Add "Do not save", so that the CSV of Operations is not overwritten. Write where it failed in `decision-log.md` [4min]
9. In the Command Palette, open `Migrate Prompt Files` and just check that `Convert to Skills...` is there. You use it in the next exercise [3min]

A command starts with frontmatter. It has 6 keys, and all of them are optional. This time, filling in just the top 3 is enough.

Key	What it is used for	What to write this time
<code>name</code>	The name after the slash. If you omit it, the file name is used	<code>clean-operations</code>
<code>description</code>	A short description shown in the suggestion list. This is where you write what the command is for	Used for monthly operations data preparation
<code>argument-hint</code>	A hint shown in the chat input box. It shows what to pass	Sheet name
<code>agent</code>	Which of ask / agent / plan the command runs in	Omit it
<code>model</code>	The model to use. If you omit it, the selected model is used	Omit it
<code>tools</code>	The list of tools the command may use	Omit it

Only the target sheet name changes each time. Make that part a placeholder. The skeleton of the file looks like this.

```

---
name: clean-operations
description: Used for monthly operations data preparation
argument-hint: <sheet name>
---

The target sheet is ${input:sheet:Operations}.
If no sheet name is passed, use the Operations sheet as the target.

```

In `${input:sheet:Operations}` , the part after the last colon is a hint shown in the input box and does not work as a default value. Write 1 line in the body for what to do when no value is passed.

Frontmatter copied from an old article may say `mode: agent` . This is the syntax used until around 2025. Now use `agent:` . If you leave `mode:` as it is, it does not work as frontmatter and the setting you meant to specify is ignored. If you see `mode:` on your screen, rewrite it right away.

What to think through yourself

- Write 1 prompt yourself before you ask the AI. Count for yourself whether the 5 elements are in it
- Compare it with the instructor's sample and write out the elements that were missing
- 1 line on where the output differed between typing the prompt by hand and calling it with a slash
- 1 line on the part you need to rewrite when you use this command next month

Result

You end up with `tpro-work/.github/prompts/clean-operations.prompt.md` and 5 lines added to `decision-log.md` . Keep `draft-prompt.md` as well. It shows how the text you first wrote changed.

Check your result

- `clean-operations.prompt.md` is in `.github/prompts/`
- `name` uses only lowercase letters, digits and hyphens
- `clean-operations` appears in the suggestion list when you type a slash
- The valid row count is the same whether you type the prompt by hand or call the command
- When you paste the Inspections sheet and call the command, you can write in `decision-log.md` where it failed

If it does not work

Symptom	Fix
It does not appear in the suggestions when you type a slash	Open a new chat again. If it still does not appear, switch to <code>/create-skill</code> . That creates <code>.github/skills/clean-operations/SKILL.md</code> , but the way you call it by name is the same

Symptom	Fix
The name contains Japanese characters	Change it to lowercase letters, digits and hyphens. Write any Japanese in <code>description</code> and the body
The output format changes every time you call the command	The body has no instruction about the output. Add the instruction down to the column names
You cannot find where the file was saved	In the Command Palette, run <code>Configure Prompt Files...</code> and look at the sources shown on the screen. The path differs by environment, so you do not need to remember it
The frontmatter says <code>mode:</code>	Rewrite it as <code>agent:</code>

If time allows

- Turn the prompt you type most often into one more command. Decide the name and description first and then write the contents, and you finish faster
- Change the text of `argument-hint`, type the slash again and see how the hint in the input box changes

Where commands stand now

The mechanism you just built is in transition. The official VS Code documentation says that prompt files are deprecated and are not loaded in Agent Host sessions. For now they work on the existing side, but it has been announced that the existing side will also go away eventually. The successor is Agent Skills, which you build in the next exercise.

There is a way to carry over what you built without throwing it away. In the Command Palette, choose `Convert to Skills...` from `Migrate Prompt Files` to convert a `.prompt.md` file into a skill. Today you only check where it is. In the next exercise, you build the skill by hand without using the conversion.

Exercise 4: Move the command into a skill

The working time is [38min]. The breakdown is: check the limits of the command [3min], write the judgment rules [5min], create the skill [6min], put the judgment rules in [5min], create the template [5min], write the body [6min], call it and check [5min], rewrite the description and call it again [3min].

Scenario

A command can hold only 1 prompt. It can get as far as deciding which machine to stop, but you have to add the report template and the judgment rules every time. If you add "in the maintenance meeting template" to the output of `/clean-operations`, the format changes only that once. When you call it again in a new chat, it goes back. The number and order of headings vary every time you send it.

A skill is a folder. Next to the steps file, you can put the judgment rules and the output template in the same place. You move the same job into a form that you can hand over together with its assets.

What goes in it	Contents
equipment-stop-report/	The folder name becomes the skill name. Use only lowercase letters, digits and hyphens
SKILL.md	The steps file. The file name is in capitals. Write <code>name</code> and <code>description</code> in the frontmatter, and the steps in the body below it
references/judge-rules.md	A place for the material you judge by. Put the criteria for stopping a machine here
templates/meeting.md	A place for the output template. The report comes back in the order of the headings here

Steps

1. Check the limits of the command. Add a template specification to the output of `/clean-operations`, and ask how to get the same format every time you call it [3min]
2. Write 3 lines of criteria for deciding which machine to stop, by hand. They are how to compare with the overall mean, the ranking of Defects count, and the lower limit of Stops. Add a 4th line: if no machine meets the criteria in a month, write "not narrowed down" [5min]
3. In the Command Palette, run `New Skill File...`. Set the location to the workspace and the name to `equipment-stop-report`. In the Explorer, check that the folder name and the `name` in `SKILL.md` are the same string [6min]
4. In the same folder, create `references/judge-rules.md` and put in the 4 lines of judgment rules you wrote [5min]
5. Create `templates/meeting.md` and list 5 headings for the template you bring to the maintenance meeting: target machine, supporting numbers, stop period, work during the stop, and outlook if not stopped [5min]
6. Have Copilot write 3 steps in the body of `SKILL.md`: read the judgment rules, total the data, and output in the template. Do not fix the name of the CSV used for the totals. Use the one that starts with `cleaned_` and was saved last, and have Copilot write the name of the file it used at the top of the output. Also check by eye that the references are relative paths [6min]
7. Call `/equipment-stop-report` and compare 2 things: whether the CSV file name appears at the top, and whether the order of headings matches `meeting.md` [5min]
8. Shorten `description` to only "creates a report" and call the skill in a new chat. Then restore the text that says when to use it, and call it again. Write 1 line on the difference between the 2 calls [3min]

The top of `SKILL.md` has the form below. The 2 lines between the `---` lines are the core.

```
---
name: equipment-stop-report
description: (What the skill does). (When to use it).
---
```

`name` may use only lowercase letters, digits and hyphens, up to 64 characters, and cannot start or end with a hyphen. If it differs from the folder name by even 1 character, the skill is not loaded. In `description`, write both what the skill does and when to use it. If this is vague, Copilot does not load the skill on its own.

Line 4 of `judge-rules.md`, the line that says to write "not narrowed down" for a month that meets no criteria, is what matters. Without it, the AI forces out 1 machine even for a month that fits none of the criteria.

What to think through yourself

- Decide 3 criteria for stopping a machine yourself and write them in `judge-rules.md`. Do not let the AI decide them
- Line up 5 headings for the maintenance meeting template yourself
- 1 line on how the content you can write differs between a command and a skill
- 1 line on the difference in behavior when you shortened `description`

Result

`tpro-work/.github/skills/equipment-stop-report/` now holds 3 files (`SKILL.md`, `references/judge-rules.md` and `templates/meeting.md`). 4 lines are added to `decision-log.md`.

Check your result

- The `name` in `SKILL.md` matches the folder name
- `name` uses only lowercase letters, digits and hyphens
- `description` says both what the skill does and when to use it
- `references` and `templates` are in the same folder and are referenced from `SKILL.md` by relative path
- The output of `/equipment-stop-report` follows the heading order of `templates/meeting.md`

If it does not work

Symptom	Fix
The folder name is in Japanese and the skill does not work	<code>name</code> must use only lowercase letters, digits and hyphens, and it must match the folder name. Japanese goes in <code>description</code> and the body
The file is named <code>skill.md</code>	The file name must be <code>SKILL.md</code> in capitals
The output ignores the template	Check that the references in the body are relative paths. Use forward slashes as separators.
You are trying to put a script in the skill	Bundling a script is possible, but it cannot run in this environment. Build the skill from the steps file and the template only
The skill is not loaded automatically	If <code>description</code> is vague, the skill is not picked up. To make sure it is used, call it explicitly with a slash
An old setting has a path in it	If anything is written in <code>chat.agentSkillsLocations</code> , the skill is not picked up. Delete it

If time allows

- Make just 1 criterion in `judge-rules.md` stricter, call the skill again, and see whether the machine that gets chosen changes. This is the first time you can check whether the way you wrote the judgment rules affects the output

Exercise 5: Swap the template

The working time is [38min]. The breakdown is: create the client template [5min], create the daily report template [4min], add the condition to the body [6min], call it for the maintenance meeting [5min], call it for the client [5min], call it for the daily report [4min], line up the 3 outputs [5min], add the missing field and call it again [4min].

Scenario

Even for the same decision, the way you write it changes with who receives it: the maintenance meeting, the client or the daily report. Instead of rewriting it every time, you add a template file to the folder and call the one you need. This is where it first pays off that a skill is a folder.

Template	How to write it
<code>meeting.md</code>	This template is for the maintenance meeting. It has these headings: target machine, supporting numbers, stop period, work during the stop, and outlook if not stopped. Equipment IDs and numbers appear as they are
<code>customer.md</code>	This template is for client submission. Do not show Equipment IDs. Do not assert the cause. Write actions and dates first. A date that is not in <code>decision-log.md</code> stays blank
<code>daily.md</code>	This template is for the daily report. It ends in 3 lines: what, by when, and who checks. If you do not write a line limit in the template, it grows longer

Steps

1. Create `templates/customer.md`. Besides the headings, write 2 lines in the template: do not show Equipment IDs, and do not assert the cause [5min]
2. Create `templates/daily.md` with a form that ends in 3 lines, and write the line limit in the template [4min]
3. Have Copilot add to the body of `SKILL.md` the condition for choosing the template by argument. Also check that `argument-hint` was added to the frontmatter [6min]
4. Call `/equipment-stop-report` with no argument, check that the output uses the maintenance meeting template, and keep a copy of the output [5min]
5. Call it with the customer argument and have Copilot check that no internal Equipment IDs remain [5min]
6. Call it with the daily argument and count by yourself whether it fits in 3 lines [4min]
7. Line up the 3 outputs and have Copilot list the numbers that appear in all of them and the information that each template dropped [5min]
8. Add a field to `customer.md` for what to report by when, and call it with customer again [4min]

You add 2 parts to `SKILL.md`. The upper part goes in the frontmatter, and the lower part is how to write the body.

```
argument-hint: meeting | customer | daily

If the argument is meeting, use templates/meeting.md;
if customer, templates/customer.md;
if daily, templates/daily.md
as the heading structure of the output.
If no argument is given, use templates/meeting.md.
```

What switches the template is the condition you wrote in the body. `argument-hint` only shows how to write the argument, so it does not switch anything by itself.

What to think through yourself

- Decide yourself 3 items that you leave out when the report goes to a client
- Decide yourself which 3 lines to keep in the daily report
- Of the information that a template dropped, 1 line on what must not be dropped

Result

You get 3 files in `templates/`, a `SKILL.md` that accepts an argument, and 3 lines added to `decision-log.md`. Keep the outputs from the 3 templates and do not delete them, because you use them for the comparison.

Check your result

- `templates/` has 3 files: `meeting.md`, `customer.md` and `daily.md`
- Calling with no argument uses the `meeting.md` template
- The client output shows no internal Equipment IDs
- The daily report output fits in 3 lines
- The supporting numbers have the same values in the maintenance meeting output and the daily report output

If it does not work

Symptom	Fix
The template does not change even when you add an argument	The body of <code>SKILL.md</code> has no condition. <code>argument-hint</code> alone does not switch the template
Equipment IDs remain in the client output	Headings in <code>customer.md</code> are not enough. Write it in the body as a prohibition
The numbers differ between the maintenance meeting output and the daily report output	The totals are being recalculated. Check that the body has an instruction to always refer to the cleaned CSV
The daily report output is over 3 lines	The template has no line limit. Add one to the template

Symptom	Fix
The skill does not appear as a candidate	Check that <code>user-invocable</code> is not set to <code>false</code> . The default is <code>true</code>

If only the maintenance meeting template works out of the 3, it is fine to move on. You can add the client and daily templates later. Do not spend too much time here.

If time allows

- Instead of adding a 4th template, add the line "do not assert the cause" to `customer.md` and see how the output changes. A template can hold prohibitions on how to write, not only headings

Exercise 6: Put the 4 shop-floor documents into 1 table

The working time is [40min]. The breakdown is: skim the 4 files [5min], create the instructions file [6min], check that it works [4min], put the fault history into 7 columns [7min], put the inspection record into 7 columns [6min], import the 2 memos [5min], merge into 1 file and save [4min], write in `decision-log.md` [3min].

Scenario

The equipment data you handled in the morning had 1 row per machine, fixed columns and the unit written in the column name. Even when a value is missing, you can see where it is missing. The 4 files you handle next are different. A row may be 1 day or 1 event, and neither the way dates are written nor the wording of the result is consistent.

They are messy not because the writers were careless, but because they were never written to become a table. They are written on the assumption that the reader works in the same workplace, so they lack clues for a tool that comes from outside.

What varies	How it actually appears	Normalize to
Date	7/1(Wed), July 2, 07/03, 2026/7/6, 7.11, full-width 7 / 9	2026-07-01 format
Result	No abnormality, OK, ○, No problem, No anomaly, Watch	Normal or other
Duration	40 min, 0.5h, an hour and a half, 1.5h, 1 hr 20 min	Number in minutes
Pressure unit	14.2MPa, 14.0, with full-width 14.1M P a	Half-width number; unit in the column name
Item names	Hydraulic, Air and Noise change partway through to Pressure, Air pressure and Sound	Map to 1 column with a mapping table

There are 4 points to watch. Shorthand such as "that bracket" in the handover memo is understood only between the person who wrote it and the person who received it. The phone memo keeps corrections without overwriting

the original, so if you pick up the retracted side, a wrong value remains. Partway through July the assignee changes and the item names change. A day off has no blank line and no note, so if you skip over it, the dates shift by 1 day.

Steps

1. Open the 4 files in VS Code and skim them. Write 2 spots that are not in table form in `decision-log.md`. Also see how many writers there are and where they change [5min]
2. Create `.github/instructions/records.instructions.md`. Set `applyTo` to the files under records, and write 3 lines in the body: how to quote, the date format and how to handle abbreviations [6min]
3. With a file inside records open, ask 1 question and check that the file name appears in References. Try again with only a file outside the scope open, and check that it does not appear [4min]
4. Have Copilot put the fault history into 7 columns: Date, Equipment ID, Equipment name, Symptom, Action, Duration and Assignee. Have it unify Duration to numbers in minutes. For rows where Equipment ID and Equipment name do not match, have it only list them and do not have it fix them [7min]
5. Have Copilot put the daily inspection records into the same 7 columns. First have it write the date when the item names changed and how the old and new names correspond, then have it make the table [6min]
6. Have Copilot put only the events whose date can be pinned down from the handover memo and the phone memo into the same 7 columns. Have it list the ones it cannot date separately, worded as in the original. Have it list the abbreviations in two groups: those whose meaning is given at the end of the memo and those whose meaning is not [5min]
7. Have Copilot combine the 4 files into one event list, add a Source column, and save it as `events.csv` [4min]
8. Add 8 lines to `decision-log.md` [3min]

Put the instructions file at `tpro-work/.github/instructions/records.instructions.md`. The part enclosed by the --- lines sets the scope.

```
---
applyTo: 'data/records/**'
---

When quoting, give the original file name and item number.
Convert dates to YYYY-MM-DD and keep the original form in the Source column.
Expand only abbreviations listed at the end of the handover memo; leave others as they are.
```

Three lines of body text are enough. The longer you write, the harder it gets to tell which line worked. If you do not write `applyTo`, the file does not apply automatically, and it takes effect only when you attach it by hand. If its name does not appear in References, the instructions you wrote did not reach the chat.

What to think through yourself

- 3 lines on information that did not fit in the table. Choose information that you must not discard just because it did not fit
- Try replacing 3 abbreviations from the handover memo with the terms used at your own workplace
- 1 line with the number of items whose date could not be pinned down

- 1 line with your own count of how many entries about the 1 machine you narrowed down in the morning appear in these 4 files

Result

You get `tpro-work/events.csv` (the 7 columns plus Source, 8 in all), `records.instructions.md`, and 8 lines added to `decision-log.md`.

Check your result

- In `records.instructions.md`, `applyTo` points to the records folder, and the file name appears in References
- `events.csv` has 8 columns, and the Source column holds the file name and item number
- The date is in the form 2026-07-01 in every row
- Items whose date cannot be pinned down are in a separate list and are not filled in by guessing
- You have compared the days marked "No abnormality" in the inspection record with the fault history and listed the days that do not match yourself
- You have counted for yourself how many entries about the 1 machine you narrowed down in the morning appear in these 4 files, and written it down

If it does not work

Symptom	Fix
<code>applyTo</code> is left out	If you leave it out, the file does not apply automatically. Specify the records folder explicitly
It does not appear in References	The target file is not open. Compare the results with the file open and not open
Dates get filled in by guessing	Send the instruction not to fill them in again. If you go on with them filled in, the order of the events changes
Duration stays as text	Have Copilot state how to convert values such as an hour and a half and 1 hr 20 min. If the values are not all numbers, you cannot get a total
Copilot fixes mismatched equipment names on its own	Stop it with an instruction to list the mismatches and not fix them. When you find a mismatch, do not decide which one is correct; leave it in the list
Abbreviations get interpreted on their own	Make Copilot always use the abbreviation list at the end of the handover memo. Have it leave abbreviations that are not in the list as they are, without expanding them
Processing all 4 files at once breaks the result	Go back to 1 file at a time. Do not carry a state where not even 1 file has become a table into the next step

Comprehension check 2

On a day when the daily inspection record says "No abnormality", the fault history has a record for the same day. How do you handle it?

- A The fault history is correct, so rewrite the row in the inspection record
- B The inspection record is in date order, so use that one
- C Do not decide which is correct; keep both entries side by side as they are
- D The day is contradictory, so remove both from the table

Read this after you have thought it through yourself. The correct answer is C. When you compare the two and find a day that does not match, choosing one side on the spot erases the information from the side you did not choose. "No abnormality" in the inspection record only means that nothing came up in that day's inspection items. It does not mean the machine did not stop. The writers differ, and so does what each of them looked at. A and B both throw one side away without grounds. D deletes the mismatching day itself, so it loses the most information. Finding the mismatching day is the result you want, so do not have Copilot summarize it; keep both entries side by side.

If time allows

- Find the drawing number of the part that appears in the handover memo from the May records in the fault history. The memo does not give the drawing number; it only says to look at May in the fault history
- Have Copilot produce 2 lists, one that reflects the corrections in the phone memo and one that does not, and count how many values change

Exercise 7: Turn the flow that worked into a skill

The working time is [35min]. The breakdown is: review the conversation [4min], turn it into a skill [6min], read the generated steps [6min], add the missing assumptions [6min], make the abbreviation table [5min], call it in a new chat [5min], and write in decision-log.md [3min].

Scenario

The steps you just used to turn the 4 files into a table took many rounds of back-and-forth to get right. You will do the same thing with next month's records. Instead of having the AI write something new, you cut the conversation that just worked into a skill. Of today's exercises, this is the closest to real work.

If you closed the conversation from before the break, the AI cannot pick up the flow. Continue in the same chat.

Remember one thing. The AI does not write down the places where it got stuck. The steps it generates are just the results rearranged into the most efficient order. What you learned from redoing something 3 times, and the assumptions you should have checked first, are not written. Whoever follows these steps next month will get stuck in the same places. That is why this exercise takes time to read the steps and add what is missing.

Steps

1. Look back at this conversation and write 2 lines in `decision-log.md` on where you had to redo work and which instruction worked [4min]
2. Ask Copilot to turn the current flow into a skill. Name it `records-to-table` and have it write the steps in the order you actually went through them [6min]
3. In the Explorer, open `.github/skills/records-to-table/SKILL.md` and read it. Compare the 4 points in the table below with the steps you actually did [6min]
4. Add the missing assumptions yourself. There are 3: always keep a Source column, do not fill in dates by guessing, and look up abbreviations in the abbreviation table [6min]
5. In the same folder, create `references/glossary.md` and put in the abbreviation list from the end of the handover memo. Have Copilot add a note that terms the memo does not define are left as they are, without expanding them [5min]
6. Open a new chat, type a slash, and check that `records-to-table` appears in the list. Call it with only the phone memo as the target, and check that the columns of the resulting table are the same as when you combined the 4 files [5min]
7. In `decision-log.md`, write 3 lines on what the generated skill was missing [3min]

What to check	Points to compare with your steps
Step numbers and order	In practice, you read the records and then decided the columns. If the columns are written first, have Copilot reorder the steps to the order you actually went through
Column specification	Does it name all 8 columns exactly: Date, Equipment ID, Equipment name, Symptom, Action, Duration, Assignee and Source?
How references are written	Does the body tell Copilot to read the files in references? Is it written as a relative path?
Target file	Is it set up to process all 4 files at once? Can it take 1 file at a time?

What to think through yourself

- 2 lines on where you had to redo work in the earlier conversation. Check whether that rework is reflected in the skill
- 3 lines on the assumptions the generated `SKILL.md` was missing
- 1 line on what else, besides this skill, you need to hand over to the person who works with these records next month

Result

2 files are created in `tpro-work/.github/skills/records-to-table/` (`SKILL.md` and `references/glossary.md`). You add 6 lines to `decision-log.md` .

Check your result

- `records-to-table` is under `.github/skills`
- The steps in `SKILL.md` are in the order you actually went through them
- It includes an instruction not to fill in dates by guessing
- `references/glossary.md` is in the same folder and is referenced from `SKILL.md`
- `/records-to-table` can process just 1 file

If it does not work

Symptom	Fix
The steps differ from the actual order	They have been rewritten in the most efficient order. Have Copilot put them back in the order you went through them. If the reasons for the rework are not left in the steps, you will get stuck in the same place again next month
The steps are too abstract to use	Have Copilot write down even the names of the 8 columns
Abbreviations get expanded on their own	Check that the reference to <code>glossary.md</code> in the body is a relative path
Calling it processes all 4 files	Have Copilot change it so that it takes the target file as an argument

If time allows

- Think of 1 procedure document at your workplace that still circulates on paper, and lay out just its headings in the form of `SKILL.md`. Leave the contents empty and check only whether it fits

Exercise 8: The Q1 data on your own

The working time is [20min]. The breakdown is: turn the sheet into text and paste it [4min], run the data preparation [6min], name 1 machine [6min], and write notes [4min].

Scenario

With the Q1 data, go as far as you did in the morning. Use only the command and skills you made today. During this time, the instructor does not explain the steps. If you get stuck, raise your hand. The instructor will only tell you which file to look at. The instructor will not give you the contents of a prompt or how to fix one.

If you get through this part on your own, you can do the same thing alone next month.

Before you start, check that these 4 items are under `.github`: in `prompts`, `clean-operations.prompt.md`; in `skills`, `equipment-stop-report` and `records-to-table`; in `instructions`, `records.instructions.md`. If any is missing, raise your hand. The instructor will paste the matching files into the Zoom chat, and you save them in the same places. Without all 4 in place, you will get stuck during this exercise.

Steps

1. Open `data/equipment-data_2026Q1.xlsx` in `01_data-loader.html` , turn the Operations sheet into text, and paste it. The sheet names are the same 3 as in the morning, but check the columns and the number of machines after you open the file, before you go on [4min]
2. Run the data preparation with `/clean-operations` . If it does not go through, fix the command itself [6min]
3. Call `/equipment-stop-report` with the template for the maintenance meeting and name 1 machine. Check the CSV name at the top of the output, and whether the criteria in `judge-rules.md` still work for Q1 [6min]
4. Write 3 lines in `decision-log.md` on what did not work and which part of the harness you fixed [4min]

What to think through yourself

- 1 line on where you got stuck: what you called, and which part of the returned content did not fit
- 1 line on which harness file you fixed: `.prompt.md` , `SKILL.md` or `judge-rules.md` . Also write the line you changed
- 1 line on the part you did not have to fix. That part works as it is next month too

Result

You get the 1 machine from Q1, the diff of your harness fixes, and 3 lines added to `decision-log.md` . The fixed harness goes into the take-home folder as it is.

Check your result

- You have named 1 machine from the Q1 data
- You have gone through to the end without asking for the steps
- What you fixed, or why you did not need to fix anything, is written in `decision-log.md`

If it does not work

Symptom	Fix
The command fails partway through	Fixing it is the right answer. Next month's data may not have the same shape. Noticing that is the point of this exercise
The criteria do not narrow it down to 1 machine	It is fine to write that it could not be narrowed down. Do not bend the criteria
You are stuck	Raise your hand. The instructor will only tell you which file to look at
The file exists but does not appear as a candidate	Open a new chat

If time allows

- In `data` , also run the same command on `quality-inspection_2026H1.csv` and `maintenance-records_2026H1.csv` . You will see where it fails when the columns differ

Subagents and model assignment

So far you have done everything in a single conversation. The drawing revision diff and the comparison of the bill of materials with the procurement request, which come next, look at different things. If you ask for both one after the other in a single conversation, the viewpoint you added later pulls the AI along, and it misses more of what you asked for first. Prepare 2 files that each have a single role, and run them separately from the parent.

Agent files

You create only the 2 child agents. In a workspace, put them in `.github/agents`. The extension is `.agent.md`. In the part enclosed by the `---` lines, write the following 4 items.

```
---
name: a name of lowercase English letters and hyphens only
description: one sentence on when to use it
model: the model name exactly as it appears on screen
user-invocable: false ← callable only from the parent
---
Below this line, write in English what the agent does
```

You do not create the parent. The built-in Agent in the chat acts as the parent. A child is called only when 2 conditions are met: the mode at the bottom of the chat is set to Agent, and `agent` (runSubagent) is enabled in the tool selection in the input box.

How it is invoked

You do not call a subagent yourself the way you call a command. In the parent conversation, ask "Run the drawing-diff agent as a subagent", and the parent decides it is needed and delegates to the child. The child does not inherit the parent conversation. It sees only the documents you hand over and its own role. Put all the information it needs in the request. You cannot add messages midway, and the parent summarizes the results that come back.

How the model is decided

Priority	Where it is decided	How you use it today
1	When you specify a name at the time of the call	Not used today
2	The name written in <code>.agent.md</code> under <code>model</code>	You set it here today. Paste the string exactly as shown on screen
3	The model chosen in the parent conversation	When 2 is not written, this one is used as it is

A subagent's model cannot exceed the cost tier of its parent. The only direction that works is a higher-tier parent with lighter subagents. Hover over a subagent's section to see what that delegation used. The numbers depend on your plan, so today just learn how to read them.

Billing is in AI credits. 1 credit is \$0.01, and credits are used up according to the number of tokens exchanged. Inline suggestions and next edit suggestions do not use these credits. Each plan has a monthly credit allowance. Ask your IT department about your company's allowance.

Reading drawings has limits. On MechVQA, a benchmark of mechanical drawings, general models scored 63-79% overall. Reading dimensions and symbols as text scored 72-84%, and reasoning from the values it read scored 52-67%. Projection and multi-view tasks, which link a front view to a section, scored 8-64%. Today you leave only the reading to the agent, and you write how it affects the order.

Exercise 9: Note the model names

The work takes [5min]. The breakdown is: open the list [2min], copy 2 names [2min], and ask 1 question to check [1min].

Scenario

A model name written in an agent file has no effect if it differs from the screen by even 1 character. The models in the list depend on your organization's settings, so first note which models this PC shows now.

Steps

1. Open the dropdown at the bottom of the Chat view that shows the current model name [2min]
2. Copy one higher-tier model and one light model, exactly as shown, into `decision-log.md` [2min]
3. Ask 1 question and check which model answered [1min]

What to think through yourself

- 1 task in your own work that needs a higher-tier model. It is the part where you isolate a cause or decide the structure
- 1 task that a light model can handle. It is the part where you format a table, fix wording or tidy the layout

Result

`decision-log.md` now has 2 lines of model names and 2 lines on how you would assign them. In the next exercise, you paste the model names you noted as they are into `.agent.md`.

Check your result

- `decision-log.md` has 2 lines of model names
- The names are copied exactly as shown on screen, including text in parentheses and version numbers

If it does not work

Symptom	Fix
Only 1 model appears in the list	Your organization's settings have narrowed the list. In the next exercise, do not write the <code>model</code> line, and do only the part where you split the roles between agents
You write the name from memory	Copy it from the screen. If even 1 character differs, the setting in <code>.agent.md</code> has no effect

If time allows

- In the Command Palette, open `Chat: Manage Language Models` and see the models your plan lets you use

Exercise 10: Split the drawing work between subagents

The work takes [40min]. The breakdown is: create the diff agent [5min], create the cross-check agent [5min], run them from the parent [5min], check the diffs [6min], check the cross-check [6min], summarize in the order memo [5min], look at the usage [4min], and write in the decision log [4min].

Scenario

You will replace 6 conveyor roller units RL-300 on conveyor line 3. The parts are replaced while the line is stopped. Conveyor roller shaft DRW-2026-042 has rev A and rev B, and the bill of materials and the procurement request have also arrived. All of them are fictional parts of a fictional company.

Going through the revision diffs and cross-checking the bill of materials against the procurement request look at different things. Split them between 2 agents, run each on a light model, and let the parent merge the results. A wrong order wastes the time the line is stopped.

Agent	Role
drawing-diff	Compares the drawings of rev A and rev B and lists diffs in 4 points: dimensions, fit symbols, thread sizes and keyways. Makes no procurement decision
doc-crosscheck	Reads the bill of materials and the procurement request against the latest drawing (<code>_svg</code>) and lists mismatches in 4 points: drawing no., rev, qty per unit and quantity. Does not judge which is correct

Both only list items. Set `user-invocable: false` so that only the parent can call them.

The drawings are in the `drawings` folder. You look at the 2 PNG files yourself, and you have the AI read the 2 files in `_svg`, which are in a form readable as text. If you cannot find the `_svg` folder, raise your hand. Passing a PNG as an image behaves differently depending on the environment, so you do not use it today.

Steps

- Create `drawing-diff.agent.md` in `tpro-work/.github/agents`. Limit its role to listing diffs, and paste into `model` the name of the light model you noted in `decision-log.md` [5min]

2. Create `doc-crosscheck.agent.md` in the same place. Limit its role to reading the bill of materials, the procurement request and the latest drawing (`_svg`) and listing mismatches in 4 points: drawing no., rev, qty per unit and quantity. Paste the name of the light model into `model` here too [5min]
3. In the parent chat, ask it to run the 2 agents as subagents. Replace the `<>` parts of the template below with the names and locations in your environment, then send it [5min]
4. Check the result of `drawing-diff` . Have it split the diffs into those listed in the revision block of the drawing and those not listed [6min]
5. Check the result of `doc-crosscheck` . Look at 3 axes: the qty per unit in the bill of materials, the rev in the request, and the quantity [6min]
6. Have the parent merge the results of the 2 agents into a 1-page `order-memo.md` . The 4 items are drawing no., rev, quantity and notes for the machine shop. Always put changes that do not appear in the revision block into the notes [5min]
7. Hover over the subagent section and look at what that delegation used. You do not need to note the numbers [4min]
8. Add 5 lines to `decision-log.md` [4min]

```
Run <agent name> as a subagent,  
and have it compare the documents in <location of the documents>.  
At the same time, run <other agent name> as a subagent,  
and have it cross-check <the other set of documents>.
```

The cross-check looks at 3 axes. For qty per unit, put the qty per unit in the bill of materials next to the count written on the drawing. The bill of materials rules say that for machining where the drawing states a count, the drawing is right. For rev, look at the rev field in the request and the issue date in the revision block of the drawing, and say which was the latest on the request date. For quantity, multiply the number of units in the Purpose field of the request by the qty per unit, and write the basis for the number of pieces in 1 line. If the subagent writes "probably", that part is not written in the documents. Do not let it fill the gap with estimates.

What to think through yourself

- 1 line on why you split the work into 2 agents. Also write what would happen if you did it with 1
- 2 lines on what would have happened if you had ordered by looking only at the revision block
- 1 line on a task at your own workplace that you could split like these 2

Result

You now have `drawing-diff.agent.md` and `doc-crosscheck.agent.md` in `tpro-work/.github/agents/` , `tpro-work/order-memo.md` , and 5 lines added to `decision-log.md` . The order memo goes into the "Work during the stop" section of the next report.

Check your result

- There are 2 `.agent.md` files in the `agents` folder, and in both, `model` is written exactly as shown on screen
- The revision diffs are listed in 2 groups: those in the revision block and those not in it

- Rows where the count differs between the bill of materials and the drawing are listed
- All 4 items of the order memo are filled in, and quantity has 1 line of basis that uses the number of units in the request

If it does not work

Symptom	Fix
The subagent is not called	Check that the mode at the bottom of the chat is Agent, and that <code>agent</code> (runSubagent) is on in the tool selection of the input box
It gets stuck while loading	Have it read the files in <code>_svg</code> , not the PNG files. The PNG files are for your own eyes
The diffs show only what is in the revision block	It is only copying the revision block. Compare the leader lines, hidden lines and section B-B with rev A
Adding a model name in the chat changes nothing	Naming a model in plain sentences has no effect. What works is the string you wrote in <code>.agent.md</code> under <code>model</code>
The subagent acts as if it knows nothing about the earlier conversation	That is how it works. Put all the information it needs in the prompt
You are looking for a command like <code>/subagent</code>	There is no such command. Call it by asking the parent in the chat
You could not get it to work to the end	Create only the frames for the 4 items of the order memo, leave the contents blank, and move on. You may put it in the report in a form that shows it is not filled in

If time allows

- Add surface roughness symbols to `drawing-diff` as a 3rd check and run it again. See whether the accuracy of the earlier checks drops when you add a check. This is where you see why narrowing the role matters
- Link "that bracket" in the handover memo to the drawing number in the fault history, and identify which part of which machine it is. A part called by a nickname becomes a different part once it is handed to the Purchasing department

Exercise 11: Report on the one machine to stop

The work takes [45min]. The breakdown is: choose the evidence [5min], produce page 1 [6min], add evidence from the records [6min], decide how to write it [5min], add the order memo [6min], write the outlook [5min], look for unsupported claims [5min], print preview [4min], and write in the decision log [3min].

Scenario

You have the morning's numbers, the afternoon's records and the order memo from the drawings. These 3 are still separate. You put them on 1 page. You use the `equipment-stop-report` that you made. You do not build anything new.

When you line up the 1 machine from the morning's numbers with the 4 records from the afternoon, you may find that one side is left blank. Having no record does not prove that there is no abnormality. Either there is no abnormality, or nobody took records. You go as far as writing it in a form that says you cannot tell which.

Steps

1. Choose 3 pieces of evidence for the report yourself: 1 from the numbers, 1 from the records and 1 from the drawings [5min]
2. Call `/equipment-stop-report` with the maintenance meeting template and produce page 1. Check that the CSV name at the top is `cleaned_q3_operations.csv`. You ran Q1 through it in Exercise 8, so if the name is different, call it again and give that file name. Also check that the output follows the criteria in `judge-rules.md` [6min]
3. Add evidence from the records. Have it count how many records this machine has in `events.csv`, and put the result in the evidence section. Build the prompt you send from the 4 elements of the template below [6min]
4. Have it write what the record count indicates, without asserting it. Have it give 2 possible reasons and write that you cannot tell which [5min]
5. Have it put the drawing no., rev, quantity and notes from `order-memo.md`, and 1 line of basis for the quantity, into the "Work during the stop" section [6min]
6. Have it write the outlook if the machine is not stopped. Have it use only the numbers that are known, and do not let it make up estimated numbers. For parts it cannot write, have it say they cannot be given at this point [5min]
7. Have it reread the report, list the unsupported claims, and rewrite them as statements of fact. Have it list the source of each number in the report, and delete the numbers whose source it cannot name [5min]
8. Save it as `tpro-work/report_machine-to-stop.md` and check in the print preview how many A4 pages it takes [4min]
9. Write 3 lines in `decision-log.md` about what you could not write in the report [3min]

- Target file name
- What to count (how many records this machine has)
- How to write it when there are 0 (write "0 records" as it is)
- Instruction not to assert (give 2 reasons and say you cannot tell which)

Have it count first, and then tell it how to write. If you reverse the order, the count gets lost in the sentences.

To make a PDF, have it output HTML, and put 3 things in the prompt: it will be printed on A4 portrait, it must not load external files, and it must be a single HTML file. Save it as `tpro-work/report_machine-to-stop.html`, double-click it to open it in the browser, press Ctrl+P, set the destination to PDF, and save. You can do the same inside your company without any extra software.

What to think through yourself

- Choose 3 pieces of evidence for the report yourself, 1 each from the numbers, records and drawings
- 2 lines, in your own words, on how to write the record count
- 2 lines on how to write the outlook if the machine is not stopped, without making up numbers
- 3 lines on what you could not write in the report. Type, in your own words and not written by the AI, the places where you almost filled in a guess, the places you left with weak evidence, and the choices you made by your own criteria

Result

You now have `tpro-work/report_machine-to-stop.md` , a print preview or PDF, and 7 lines added to `decision-log.md` . If you save them outside `tpro-work` , they are not included in the ZIP in the next exercise.

Check your result

- The report is saved to a file
- It has 3 pieces of evidence, and you can name the source file of each
- The record count is written in a form that does not assert the reason
- Work during the stop includes the drawing no., rev and quantity
- There are 3 or fewer unsupported claims
- It fits on 2 A4 pages or fewer

3 or fewer unsupported claims is enough. If you try to get it down to 0, the page says nothing.

If it does not work

Symptom	Fix
An amount of money appears in the outlook	The handout data has no amounts of money. Have it write that it cannot give one
It asserts the reason why there are no records	Send the prompt again, telling it to give 2 reasons and write that you cannot tell which
The format breaks	You added text by hand without calling the skill. Run it through the template again
The numbers do not match the morning totals	It recounted inside the report. Have it take the numbers again from <code>cleaned_q3_operations.csv</code>
It runs over 3 A4 pages	Run it once with the daily report template, decide what to cut, and then go back to the maintenance meeting template
It is not saved	Copy the chat output, paste it into a new file, and save it as <code>report_machine-to-stop.md</code> . It is fine if the content is thin.

If time allows

- Output the same content in the client template, and check that internal Equipment IDs and assignee names are gone. Use this as the last check before it goes to a client.

Exercise 12: Taking the harness home

The work takes [12min]. The breakdown is: list the files [3min], write next month's steps [4min], make the ZIP [3min], and write down the file to rewrite [2min].

Scenario

The files you made today are scattered under `.github`. Put them in a form you can understand next month, and take them home as one bundle. If you skip this, today's results only stay on this PC.

Steps

1. Have Copilot list everything under `.github` and write one line on what each file does. Have it read the actual contents of the files [3min]
2. Create `tpro-work/README.md` and type next month's steps in 4 lines, in your own words. Write which command to call first, which skill to apply to which material, and where you need to rewrite something. Do not keep the AI's wording as it is [4min]
3. Leave out `data` and make a ZIP. On Windows, open `tpro-work` in File Explorer and show hidden files from the View tab. Then select all items including `.github`, except `data`, and compress them from the right-click menu. On Mac, open `tpro-work` in Finder and press Command+Shift+. to show hidden files. Then select everything except `data` and compress it from the right-click menu [3min]
4. Write 1 line naming the file you rewrite first when you place this in a folder at your workplace [2min]

What to think through yourself

- Next month's steps, 4 lines, in your own words. If this part is not in your own words, you will not be able to act on it when you read it next month.
- 1 line naming the file you rewrite first when you place this at your own workplace

Result

`tpro-work/README.md` and a ZIP to take home are created. The ZIP contains the whole `.github` folder, the report, `decision-log.md`, `events.csv`, `order-memo.md` and `cleaned_q3_operations.csv`.

Check your result

- `README.md` has at least 4 lines of steps for next month
- The list includes the 6 harness items (`AGENTS.md`, 1 command, 2 skills, 1 instructions file and 2 agents)
- The ZIP does not contain the `data` folder
- The file to rewrite first at your workplace is written down

If it does not work

Symptom	Fix
The ZIP is large	The <code>data</code> folder is inside it. Take it out
The README is still the AI's text	Type just the 4 lines of next month's steps yourself
There are not enough harness items	It is fine if all 6 are there, including the ones you copied along the way

If time allows

- Assume you put this in a shared folder. Write a 1-line note on which parts of `AGENTS.md` and the instructions are common across the department. Skills and agents are easier to manage when split by person in charge.

3 lines for the chat

At the end, paste the one machine to stop and your evidence into the Zoom chat in 3 lines. Line 1 is the Equipment ID and Equipment name of the machine to stop. Line 2 is one numerical piece of evidence. Line 3 is what you found on the records side and the procurement that is stalled on the drawing side.

It is fine if you cannot narrow it down past 2 machines. Write them as 2. Even if you pick the same one machine as the person next to you, the numbers you chose as evidence and how you handled the denominator of the mean should differ. The difference is only in a few lines of `judge-rules.md`. What you bring to the meeting is both the one machine and the criteria.

Troubleshooting

This table collects problems that can happen across exercises. If none of the symptoms applies to you, you can skip this table.

Symptom	What to check	Fix
The files you made are not taking effect	References above the answer	If the name does not appear, the file was not read. Check its location and the upper and lower case of the file name.
It does not appear in the suggestions when you type a slash	How to open the chat	Open a new chat. It may not be added to a chat that is already in progress
<code>.github</code> is not read	Title bar	Check that <code>tpro-work</code> is open as the workspace. If another folder is open, it is not read.
You are asked for permission after giving it an <code>xlsx</code>	How you pass the file	Do not grant permission. Turn the file into text with <code>01_data-loader.html</code> , then paste it.
VS Code opens when you try to open the file in the browser	The app that opened the file	Right-click > Open with > choose a browser.

Symptom	What to check	Fix
The loader tool does not work	Where the file is placed	Check that you have not moved <code>01_data-loader.html</code> to a place away from the <code>lib</code> folder.
The CSV shows garbled characters	Encoding shown in the Status Bar	Save it again as UTF-8. Do not dig any deeper.
The file name got .txt added	End of the file name	In File Explorer, turn on View > File name extensions and check.
You fell behind during an exercise	Recovery slot	Raise your hand. The instructor pastes the matching file into the Zoom chat. Save it in the same place and catch up with the next exercise.

The source for where harness files go is <https://code.visualstudio.com/docs/agents/concepts/agents>. The chat in VS Code has 3 modes: Ask, Agent and Plan. The Edit mode that used to exist is gone now. Agent also makes the edits that show up as diffs.

What you take home today

Check that the following are in the `tpro-work` folder. The top 6 rows are the 6 harness items. Only the agents row has 2 files.

Location	Contents	Use next month
<code>AGENTS.md</code>	3 lines of constraints	Added automatically to every later chat
<code>.github/prompts/clean-operations.prompt.md</code>	A command that fixes the data preparation of Operations	Call it first when next month's sheet arrives
<code>.github/skills/equipment-stop-report/</code>	A skill with 1 file of judgment rules and 3 templates	Call the meeting, client or daily template as needed
<code>.github/skills/records-to-table/</code>	A skill that turns messy records into an 8-column event list. It comes with an abbreviation list	Apply it when you have more inspection records or handover memos
<code>.github/instructions/records.instructions.md</code>	Rules for quoting when reading files under records	Applies automatically in chats that work with records
<code>.github/agents/</code>	2 agents: one checks drawing revision diffs, and the other cross-checks the bill of materials against the procurement request	Run both together when a drawing is revised

Location	Contents	Use next month
report_machine-to-stop.md	Today's deliverable. The 3 pieces of evidence and what to do while the machine is stopped	Bring it to the maintenance meeting
cleaned_q3_operations.csv , events.csv , order-memo.md	Intermediate files made along the way	Keep them as the source of the numbers in the report
decision-log.md	A record of the day's decisions	Read it first when you make the same document next month

When next month's Operations sheet arrives, call `/clean-operations` first. If a column name has changed, it fails there. The place where it fails is the place to fix the harness.

Rules to follow

Do not use real data from your own work in the exercises. All the data handled today is fictional and has no connection to real equipment, products or companies. Before you swap in data from your workplace, read the 3 lines of `AGENTS.md` again.

Before you paste, generalize the text with 4 rules.

Rule	How to apply it
Generalize client names, product names, project names and partner names to the industry or process level	Write something like "the design process in the automotive industry" instead of a specific company name or equipment model number.
Do not use real data, real files or screenshots	Do not paste screens of your own Excel files or drawings. Replace the values with fictional ones, like today's exercise data.
Avoid specific values for figures and specifications, and write at the level of purpose, issues and expected effects	Write "judging dimensions takes time" instead of the actual tolerance values.
Before you submit, check yourself that the content is safe to send outside the company	Read it again when you finish. Leave out any part you are unsure about.

The same applies when you put the harness you took home in a shared folder inside your company. Do not write client names or real equipment numbers in `judge-rules.md` or the template files. In a client's environment, the client's rules come first. If you are unsure about something, do not put it there. Take it back with you.

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.



Givery, Inc. Do not redistribute this material.