



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Welcome Back

MEASURING SUCCESS +
HANDS-ON

Where We Left Off

Last session:

- **SE Refresher** — branches, PRs, hooks, Claude Code setup
- **Brownfield Reality** — 60-80% maintenance, reading > writing
- **Locust Ecosystem** — cloned the repo, ran tests and a load test
- **CLAUDE.md** — saw the before/after demo, the three-layer model

Now:

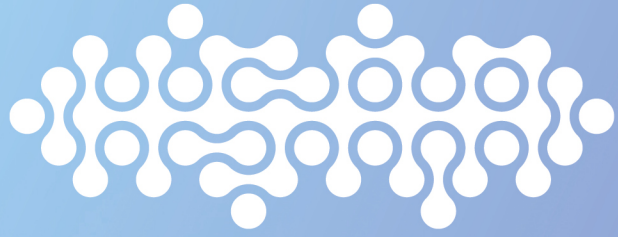
- **Measuring success** — how do you know your change is good?
- **Exercise packages** — your team's specific targets
- **Hands-on** — write tests, refactor code, submit PRs

Quick Check

Before we continue — raise your hand if:

- You have the repo cloned and `uv sync` succeeded
- You have pre-commit hooks activated (`pre-commit install`)
- You ran the fast test subset and all 125 tests passed
- You created your team's feature branch

If anything is missing, fix it now while we talk. Ask your AI tool for help.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Measuring Success

5

Tests Are Your Contract

After any change, the relevant tests **must pass**.

You do not run all 700+ tests — just your module:

```
# Package A (~8s)
uv run pytest locust/test/test_util.py locust/test/test_log.py -x -q

# Package B (~22s)
uv run pytest locust/test/test_stats.py -x -q

# Package C (~9s)
uv run pytest locust/test/test_tags.py locust/test/test_locust_class.py -x -q
```

`-x` = stop on first failure. `-q` = less noise. **Fast feedback loop.**

Full details in the [Brownfield Locust Exercises](#).

Coverage Tells a Story

```
uv run pytest --cov=locust --cov-report=term-missing <your-test-files> -q
```

- **Low coverage** = code paths no test exercises. Blind spots.
- **Your new code should have tests.**
- **High coverage + weak assertions** = worse than low coverage

A test that calls a function and asserts `True` covers lines but **verifies nothing**.

We have pre-analyzed the coverage gaps — your package tells you exactly which functions to test.

The Benchmark Workflow

```
# BEFORE your change
uv run python benchmarks/run_benchmarks.py

# Make your change

# AFTER your change
uv run python benchmarks/run_benchmarks.py
```

How to read the numbers:

- **~10% variation** — Normal noise (CPU load, background processes). Fine.
- **20%+ drop consistently** — Investigate. Your refactoring may have added overhead.

Include before/after numbers in your PR description.

Live Demo: Running the Benchmark

```
=== Locust Performance Benchmark ===
```

```
Package A targets:
```

```
log.py: setup_logging()          – 11,442 ops/sec
```

```
Package B targets:
```

```
stats.py: StatsEntry.log()       – 1,054,805 ops/sec
```

```
stats.py: StatsEntry.extend()    – 17,740 ops/sec
```

```
Package C targets:
```

```
task.py: filter_tasks_by_tags()  – 6,597 ops/sec
```

```
=====
```

Let's run it live.

Pull Requests as Communication

Every PR description should explain:

- **What** changed — summary of the diff
- **Why** — motivation, which exercise target
- **How to test** — the exact test command
- **Benchmark numbers** — before/after

AI can draft PR descriptions — but verify they describe what the code **does**, not what it **should** do.

New to PRs? See: [Feature Branch Development](#)

AI Diary in a Shared Codebase

```
diary/feature/<team-name>-<task>/  
├─ 001-onboarding.md  
├─ 002-coverage-memoize.md  
└─ 003-refactoring-setup-logging.md
```

Key change from Lecture 2: Every diary entry is **committed with the code it produced**.

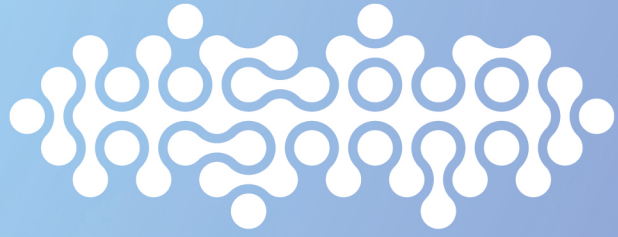
```
git commit -m "[diary] 002 – Write tests for memoize decorator"
```

Filter all AI interactions: `git log --grep='\[diary\]'`

Definition of Done

1. All existing tests pass (no regressions)
2. New tests check actual behavior (not just line coverage)
3. Code passes the linter (pre-commit hooks or `uv run ruff check .`)
4. Refactored code preserves the original contract
5. Performance not degraded (benchmark numbers in PR)
6. Commit messages are clear (`[diary]` prefix for AI-assisted commits)
7. AI Diary entries document the full process

This checklist is also in `CLAUDE.md` — your AI tool knows it too.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Your Exercise Package

6

The Three Packages

Package	Module	Test Command	Time
A (Easiest)	locust/util/ + log.py	test_util.py test_log.py	~8s
B (Medium)	stats.py + html.py	test_stats.py	~22s
C (Medium)	user/ + debug.py	test_tags.py test_locust_class.py	~9s

Your instructor assigns your team to a package. Each includes:

- **Coverage targets** — specific functions with current % to beat
- **Refactoring target** — one long method to split into helpers

Full details: [Brownfield Exercises](#)

Package A: Pure Python Utilities

Coverage targets:

- `util/cache.py` — `memoize()` decorator: **10%** → **~100%**
- `util/exception_handler.py` — `retry()` decorator: **54%** → **~100%**
- `user/inspectuser.py` — `_calc_distribution()`: **60%** → **~80%**

Refactoring target:

- `log.py:setup_logging()` — 54 lines → 3 functions of ~15 lines each

Test command: `uv run pytest locust/test/test_util.py locust/test/test_log.py locust/test/test_taskratio.py -x -q`

Package B: Statistics and Reporting

Coverage targets:

- `stats.py` — `StatsError.parse_error()`: hex address replacement logic
- `stats.py` — `StatsError.to_name()` and `serialize()`: error unwrapping
- `html.py` — `get_html_report()`: **34%** → **~70%** (no test file exists!)

Refactoring target:

- `stats.py:StatsEntry.extend()` — 43 lines → extract 3 merge helpers

Test command: `uv run pytest locust/test/test_stats.py
locust/test/test_html_filename.py -x -q`

Package C: User Model and Tasks

Coverage targets:

- `user/task.py` — `filter_tasks_by_tags()` edge cases: **91%** → **~95%**
- `user/users.py` — `HttpUser.__init__()` pool manager config: **81%** → **~88%**
- `debug.py` — `PrintListener.on_request()`: **45%** → **~70%**

Refactoring target:

- `user/task.py:filter_tasks_by_tags()` — 41 lines → extract `_task_matches_tags()`

Test command: `uv run pytest locust/test/test_tags.py
locust/test/test_locust_class.py locust/test/test_users.py -x -q`

Team Assignments

Find your team number and package on the board.

Each team:

1. Creates a feature branch: `feature/<team-name>-<package>`
2. Creates a diary subfolder: `diary/feature/<team-name>-<package>/`
3. Starts with coverage targets, then moves to refactoring
4. Submits one PR against `master` with all work + diary

The Workflow

explore → test → refactor → PR

1. **Explore** — Ask the AI about your assigned module. Compare to CLAUDE.md. Diary entry.
2. **Test** — Write tests for the specific functions in your package. Run coverage before/after.
3. **Refactor** — Split the long method. All existing tests must pass unchanged.
4. **PR** — Push branch, open PR, include benchmark numbers.

Every AI interaction → diary entry + commit.

Go!

You have 60 minutes.

Resources:

- [Locust Exploration](#) — setup guide
- [Brownfield Exercises](#) — full package details, rubric, benchmark instructions

Quick test commands:

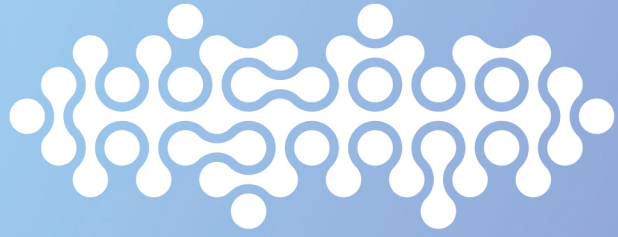
- **A:** `uv run pytest locust/test/test_util.py locust/test/test_log.py locust/test/test_taskratio.py -x -q`
- **B:** `uv run pytest locust/test/test_stats.py locust/test/test_html_filename.py -x -q`
- **C:** `uv run pytest locust/test/test_tags.py locust/test/test_locust_class.py locust/test/test_users.py -x -q`

Checkpoint (~30 min)

Quick status check:

- **Working on coverage?** Are your new tests passing?
- **Stuck on setup?** Ask your AI tool — or raise your hand.
- **Pre-commit hook rejecting your commit?** That is correct behavior. Fix the lint violations.
- **Ready for refactoring?** Make sure your new tests pass first — they become your safety net.

Common mistake: AI generates a test that asserts `True`. That is not a test. Ask: *"Would this test catch a bug?"*



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

What You Learned Today

7

Key Takeaways

- You opened a **10,000-line codebase**, ran its tests, explored it, and started contributing — all in one session.
- `CLAUDE.md` dramatically improves AI assistance. Same tool, same code, different results.
- **CLAUDE.md is influence, not enforcement.** Pre-commit hooks and your discipline are the safety net.
- The **5 key differences** between greenfield and brownfield: context, tests, reading, DoD, conventions.
- A concrete **Definition of Done**: tests pass, coverage improved, linter passes, performance stable, diary complete.

Homework

Due before next lecture:

1. Finish your coverage targets (specific functions per package)
2. Complete the refactoring (long method → smaller helpers)
3. Run the benchmark before/after
4. Submit your PR against `master` with:
 - Tests + refactored code
 - Benchmark numbers in PR description
 - Diary entries for every AI interaction

Full details and rubric: [Brownfield Exercises](#)

Coming Next

Act II begins: Understanding how AI tools actually work.

- Tokenization — how CLAUDE.md becomes tokens
- Attention — why instructions influence but do not guarantee
- Context windows — why 600 lines of CLAUDE.md costs something
- The "aha" moment: *now you know WHY the benchmark had lint violations*

The experience you built today becomes the foundation for understanding the mechanism.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Questions?

THANK YOU