



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Willkommen zurück

ERFOLG MESSEN + PRAXIS

Wo wir aufgehört haben

Letzte Sitzung:

- **SE-Auffrischung** — Branches, PRs, Hooks, Claude Code Setup
- **Brownfield-Realität** — 60-80 % Wartung, Lesen > Schreiben
- **Locust-Ökosystem** — Repo geklont, Tests und einen Lasttest ausgeführt
- **CLAUDE.md** — Vorher/Nachher-Demo, das Drei-Schichten-Modell

Jetzt:

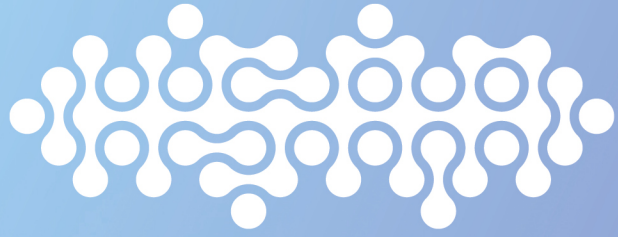
- **Erfolg messen** — woher wissen Sie, dass Ihre Änderung gut ist?
- **Übungspakete** — die spezifischen Ziele Ihres Teams
- **Praxis** — Tests schreiben, Code refactoren, PRs einreichen

Kurzer Check

Bevor wir weitermachen — heben Sie die Hand, wenn:

- Sie das Repo geklont haben und `uv sync` erfolgreich war
- Sie Pre-Commit Hooks aktiviert haben (`pre-commit install`)
- Sie die schnelle Testauswahl ausgeführt haben und alle 125 Tests bestanden
- Sie den Feature Branch Ihres Teams erstellt haben

Falls etwas fehlt, beheben Sie es jetzt während wir sprechen. Fragen Sie Ihr KI-Werkzeug um Hilfe.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Erfolg messen

5

Tests sind Ihr Vertrag

Nach jeder Änderung **müssen** die relevanten Tests bestehen.

Sie führen nicht alle 700+ Tests aus — nur die Ihres Moduls:

```
# 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 = Stopp beim ersten Fehler. **-q** = weniger Ausgabe. **Schnelle Feedback-Schleife.**

Alle Details in den [Brownfield Locust Exercises](#).

Coverage erzählt eine Geschichte

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

- **Niedrige Coverage** = Codepfade, die kein Test ausübt. Blinde Flecken.
- **Ihr neuer Code sollte Tests haben.**
- **Hohe Coverage + schwache Assertions** = schlimmer als niedrige Coverage

Ein Test, der eine Funktion aufruft und `True` assertiert, deckt Zeilen ab, **verifiziert aber nichts**.

Wir haben die Coverage-Lücken vorab analysiert — Ihr Paket sagt Ihnen genau, welche Funktionen zu testen sind.

Der Benchmark-Workflow

```
# VOR Ihrer Änderung
uv run python benchmarks/run_benchmarks.py

# Änderung durchführen

# NACH Ihrer Änderung
uv run python benchmarks/run_benchmarks.py
```

Wie Sie die Zahlen lesen:

- **~10 % Schwankung** — Normales Rauschen (CPU-Last, Hintergrundprozesse). In Ordnung.
- **20 %+ Rückgang konsistent** — Untersuchen. Ihr Refactoring könnte Overhead eingeführt haben.

Nehmen Sie Vorher/Nachher-Zahlen in Ihre PR-Beschreibung auf.

Live-Demo: Benchmark ausführen

```
=== 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
```

```
=====
```

Lassen Sie uns das live ausführen.

Pull Requests als Kommunikation

Jede PR-Beschreibung sollte erklären:

- **Was** sich geändert hat — Zusammenfassung des Diffs
- **Warum** — Motivation, welches Übungsziel
- **Wie zu testen** — der exakte Testbefehl
- **Benchmark-Zahlen** — Vorher/Nachher

KI kann PR-Beschreibungen entwerfen — aber prüfen Sie, ob sie beschreibt, was der Code **tut**, nicht was er **tun sollte**.

Neu bei PRs? Siehe: [Feature Branch Development](#)

KI-Tagebuch in einer gemeinsamen Codebasis

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

Zentrale Änderung gegenüber Vorlesung 2: Jeder Tagebucheintrag wird **zusammen mit dem Code committet**, den er dokumentiert.

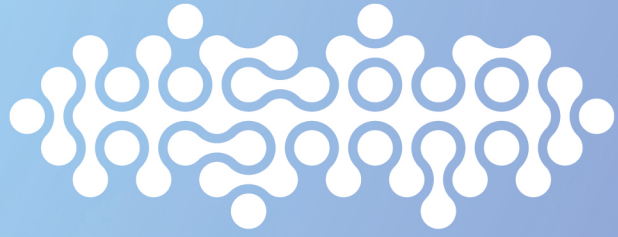
```
git commit -m "[diary] 002 – Write tests for memoize decorator"
```

Alle KI-Interaktionen filtern: `git log --grep='\[diary\]'`

Definition of Done

1. Alle bestehenden Tests bestehen (keine Regressionen)
2. Neue Tests prüfen tatsächliches Verhalten (nicht nur Zeilenabdeckung)
3. Code besteht den Linter (Pre-Commit Hooks oder `uv run ruff check .`)
4. Refactorter Code bewahrt den ursprünglichen Vertrag
5. Performance nicht verschlechtert (Benchmark-Zahlen im PR)
6. Commit-Messages sind klar (`[diary]` -Präfix für KI-gestützte Commits)
7. KI-Tagebucheinträge dokumentieren den vollständigen Prozess

Diese Checkliste steht auch in `CLAUDE.md` — Ihr KI-Werkzeug kennt sie ebenfalls.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Ihr Übungspaket

6

Die drei Pakete

Paket	Modul	Testbefehl	Zeit
A (Am einfachsten)	locust/util/ + log.py	test_util.py test_log.py	~8s
B (Mittel)	stats.py + html.py	test_stats.py	~22s
C (Mittel)	user/ + debug.py	test_tags.py test_locust_class.py	~9s

Ihre Lehrperson weist Ihr Team einem Paket zu. Jedes enthält:

- **Coverage-Ziele** — spezifische Funktionen mit aktuellem % zum Verbessern
- **Refactoring-Ziel** — eine lange Methode in Hilfsfunktionen aufteilen

Alle Details: [Brownfield Exercises](#)

Paket A: Reine Python-Hilfsfunktionen

Coverage-Ziele:

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

Refactoring-Ziel:

- `log.py:setup_logging()` — 54 Zeilen → 3 Funktionen von je ~15 Zeilen

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

Paket B: Statistik und Reporting

Coverage-Ziele:

- `stats.py` — `StatsError.parse_error()`: Hex-Adress-Ersetzungslogik
- `stats.py` — `StatsError.to_name()` und `serialize()`: Fehler-Entpackung
- `html.py` — `get_html_report()`: **34 %** → **~70 %** (keine Testdatei vorhanden!)

Refactoring-Ziel:

- `stats.py:StatsEntry.extend()` — 43 Zeilen → 3 Merge-Hilfsfunktionen extrahieren

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

Paket C: User-Modell und Tasks

Coverage-Ziele:

- `user/task.py` — `filter_tasks_by_tags()` Randfälle: **91 %** → **~95 %**
- `user/users.py` — `HttpUser.__init__()` Pool-Manager-Konfiguration: **81 %** → **~88 %**
- `debug.py` — `PrintListener.on_request()`: **45 %** → **~70 %**

Refactoring-Ziel:

- `user/task.py:filter_tasks_by_tags()` — 41 Zeilen → `_task_matches_tags()`
extrahieren

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

Teamzuweisungen

Finden Sie Ihre Teamnummer und Ihr Paket an der Tafel.

Jedes Team:

1. Erstellt einen Feature Branch: `feature/<team-name>-<package>`
2. Erstellt einen Tagebuch-Unterordner: `diary/feature/<team-name>-<package>/`
3. Beginnt mit den Coverage-Zielen, dann weiter zum Refactoring
4. Reicht einen PR gegen `master` ein mit der gesamten Arbeit + Tagebuch

Der Workflow

explore → test → refactor → PR

1. **Erkunden** — Fragen Sie die KI zu Ihrem zugewiesenen Modul. Vergleichen Sie mit CLAUDE.md. Tagebucheintrag.
2. **Testen** — Schreiben Sie Tests für die spezifischen Funktionen in Ihrem Paket. Coverage vorher/nachher messen.
3. **Refactoren** — Teilen Sie die lange Methode auf. Alle bestehenden Tests müssen unverändert bestehen.
4. **PR** — Branch pushen, PR öffnen, Benchmark-Zahlen einfügen.

Jede KI-Interaktion → Tagebucheintrag + Commit.

Los geht's!

Sie haben 60 Minuten.

Ressourcen:

- [Locust Exploration](#) — Setup-Anleitung
- [Brownfield Exercises](#) — vollständige Paketdetails, Bewertungsmatrix, Benchmark-Anleitung

Schnelle Testbefehle:

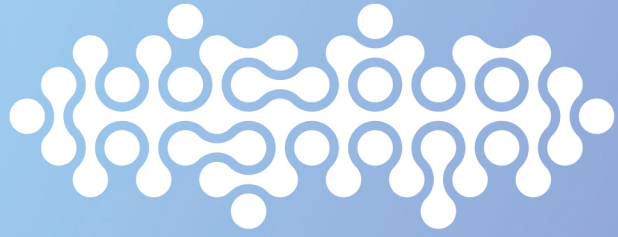
- **A:** `uv run pytest locust/test/test_util.py locust/test/test_log.py locust/test/test_taskratio.py -x -q`
- **B:** `uv run pytest locust/test/test_stats.py locust/test/test_html_filename.py -x -q`
- **C:** `uv run pytest locust/test/test_tags.py locust/test/test_locust_class.py locust/test/test_users.py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

Checkpoint (~30 Min.)

Kurzer Statuscheck:

- **An Coverage arbeiten?** Bestehen Ihre neuen Tests?
- **Beim Setup hängengeblieben?** Fragen Sie Ihr KI-Werkzeug — oder heben Sie die Hand.
- **Pre-Commit Hook lehnt Ihren Commit ab?** Das ist korrektes Verhalten. Beheben Sie die Lint-Verstöße.
- **Bereit zum Refactoring?** Stellen Sie sicher, dass Ihre neuen Tests zuerst bestehen — sie werden Ihr Sicherheitsnetz.

Häufiger Fehler: KI generiert einen Test, der `True` assertiert. Das ist kein Test. Fragen Sie: *„Würde dieser Test einen Bug erkennen?“*



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Was Sie heute gelernt haben

7

Wichtigste Erkenntnisse

- Sie haben eine **10.000-Zeilen-Codebasis** geöffnet, deren Tests ausgeführt, sie erkundet und angefangen beizutragen — alles in einer Sitzung.
- `CLAUDE.md` verbessert die KI-Unterstützung dramatisch. Gleiches Werkzeug, gleicher Code, andere Ergebnisse.
- **CLAUDE.md ist Einfluss, nicht Durchsetzung.** Pre-Commit Hooks und Ihre Disziplin sind das Sicherheitsnetz.
- Die **5 zentralen Unterschiede** zwischen Greenfield und Brownfield: Kontext, Tests, Lesen, DoD, Konventionen.
- Eine konkrete **Definition of Done**: Tests bestehen, Coverage verbessert, Linter besteht, Performance stabil, Tagebuch vollständig.

Hausaufgabe

Abgabe vor der nächsten Vorlesung:

1. Schließen Sie Ihre Coverage-Ziele ab (spezifische Funktionen pro Paket)
2. Vervollständigen Sie das Refactoring (lange Methode → kleinere Hilfsfunktionen)
3. Führen Sie den Benchmark vorher/nachher aus
4. Reichen Sie Ihren PR gegen `master` ein mit:
 - Tests + refactortem Code
 - Benchmark-Zahlen in der PR-Beschreibung
 - Tagebucheinträge für jede KI-Interaktion

Alle Details und Bewertungsmatrix: [Brownfield Exercises](#)

Ausblick

Akt II beginnt: Verstehen, wie KI-Werkzeuge tatsächlich funktionieren.

- Tokenisierung — wie CLAUDE.md zu Token wird
- Attention — warum Anweisungen beeinflussen, aber nicht garantieren
- Kontextfenster — warum 600 Zeilen CLAUDE.md etwas kosten
- Das „Aha“-Erlebnis: *Jetzt wissen Sie, WARUM der Benchmark Lint-Verstöße hatte*

Die Erfahrung, die Sie heute aufgebaut haben, wird zur Grundlage für das Verständnis des Mechanismus.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Fragen?

VIELEN DANK