Agile Software Development Model

Agile development delivers working software in short iterations with close customer collaboration. This chapter introduces the agile values and practises Scrum on the ITC Club Hub case study.

Agile Manifesto 2001 | Scrum Guide 2020 | Kanban · XP | C++20 · GoogleTest · GitHub Projects

AGENDA

What we will cover

1. The Agile Manifesto: four values and twelve principles
2. Scrum: Product Owner, Scrum Master and Developers
3. Scrum events: sprint, sprint planning, daily scrum, sprint review, retrospective
4. Scrum artifacts: product backlog, sprint backlog, increment, definition of done
5. Estimation: story points and planning poker
6. Tracking: task boards, velocity and burndown charts
7. Kanban: visualising work and limiting work in progress
8. Extreme Programming practices: pair programming, test-driven development, continuous integration, refactoring
9. Agile versus plan-driven: when each fits, and hybrid approaches
10. Common agile anti-patterns
11. Chapter Quiz
12. Lab-07

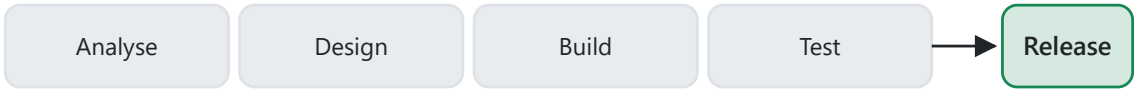
Click any item to jump directly to that topic.

From one big delivery to many small ones

- **Agile** is a family of methods (Scrum, Kanban, Extreme Programming and others) that share one set of values: the Agile Manifesto of 2001.
- **Iterative**: the team repeats a short cycle (plan, build, test, show). **Incremental**: every cycle adds a usable piece of the product.
- **Empiricism**: decide from what you observe, not from predictions. Each iteration ends with a working increment that the customer can try, so wrong requirements surface after two weeks, not after two semesters.
- In Chapter 04 you compared process models; here we practise one agile framework in detail.

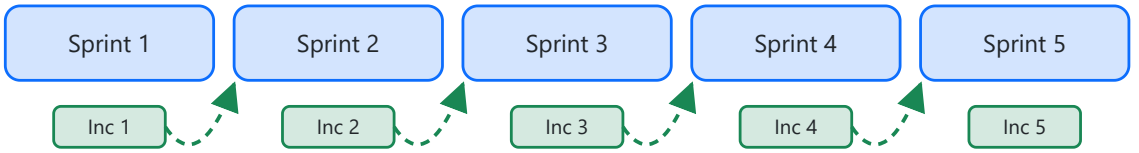
Club Hub so far: Labs 01 to 06 produced requirements, a backlog and UML models. Weeks 9 to 12 turn that backlog into running C++ in two Scrum sprints.

Plan-driven



the customer sees working software once, at the end

Agile (Scrum)



every increment is shown to the customer; the feedback shapes the next sprint

Four values: the left side is valued more than the right

valued more		still has value
People and how they interact	over	Processes and tools
Software that works	over	Exhaustive documentation
Working with the customer	over	Negotiating the contract
Responding to change	over	Sticking to the plan

Paraphrased. The manifesto was written by 17 practitioners in February 2001 at Snowbird, Utah.

Value	What it looks like in Club Hub
People	A 15-minute daily conversation beats a long ticket thread.
Working software	Progress is shown as `clubhub list-events` running, not as "UML 80 % done".
Customer	The instructor (client) tries the increment at every sprint review.
Change	After Sprint 1 the client asks for a waiting list: it goes into the backlog, not into a dispute.

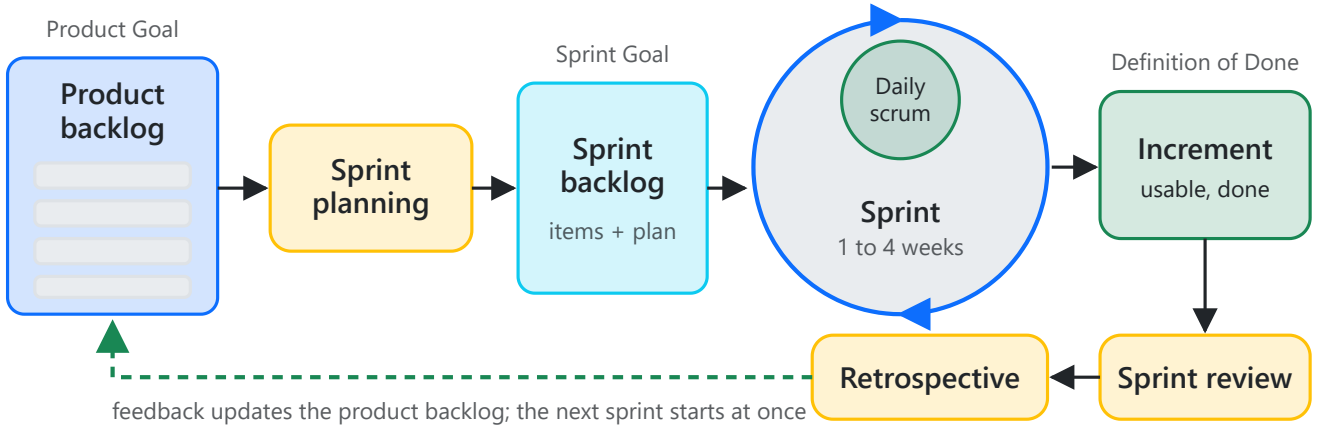
⚠ "Over" does not mean "instead of". Agile teams still write documents, use tools and make plans; they just keep them lean and useful.

Twelve principles in five groups

Principle (paraphrased)	Practice you will use
Deliver working software every few weeks	Two-week sprints, a demo at every review
Working software is the main measure of progress	Only stories that meet the definition of done count
Sustainable pace	Plan with a realistic capacity, no all-nighters before the demo
Simplicity, the art of not doing work	Build Must stories first; no "might need it later" code
Reflect regularly and adjust	A retrospective with concrete actions every sprint

Values say *what matters*; principles say *how to decide*; frameworks such as Scrum turn them into roles, events and artifacts.

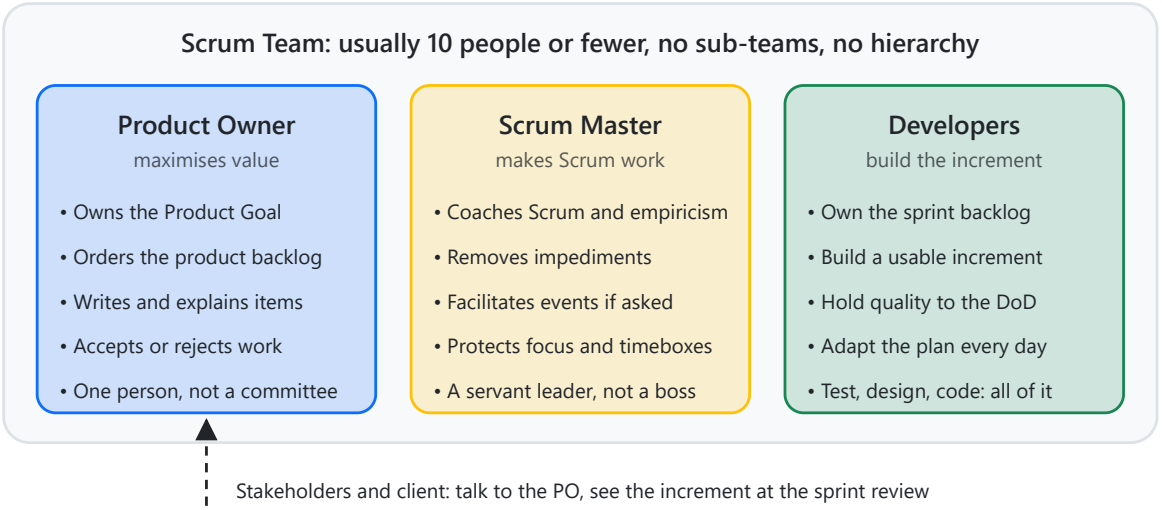
Scrum: a lightweight framework built on empiricism



- **Three pillars**: *transparency* (work is visible), *inspection* (look at it often), *adaptation* (change course when it drifts).
- **3 accountabilities**: Product Owner, Scrum Master, Developers.
- **5 events**: the sprint, which contains planning, daily scrum, review and retrospective.
- **3 artifacts**, each with a commitment: product backlog (Product Goal), sprint backlog (Sprint Goal), increment (Definition of Done).
- Scrum is deliberately incomplete: it says *what* must happen, the team chooses *how* (for example TDD from XP, boards from Kanban).

Source: Schwaber and Sutherland, *The Scrum Guide*, November 2020.

One team, three accountabilities

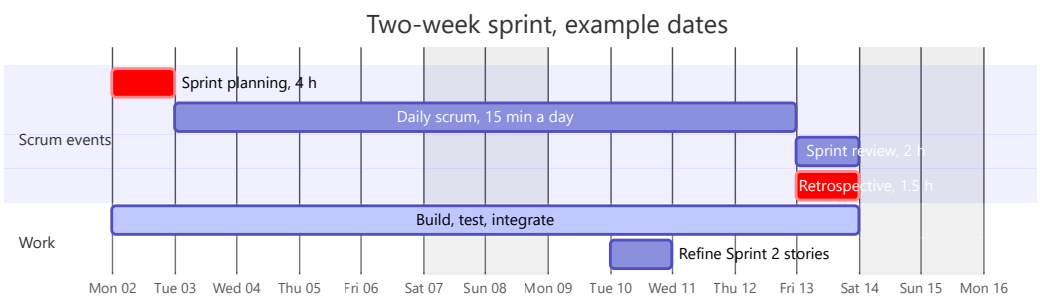


Club Hub team (example)	Sprint 1	Sprint 2
Product Owner	Dara (liaises with the instructor, who plays the client)	
Scrum Master	Sokha	Vicheka (rotates)
Developers	Sokha, Vicheka, Rithy, Malis (the SM and PO may also develop)	

- "Developers" means everyone who creates the increment: testers, designers and writers too.
- The Scrum Master has no authority over tasks; the Developers decide who does what.

⚠ In a student team the SM usually also codes. That is fine, as long as someone is explicitly watching the process and removing blockers.

Five events inside one two-week sprint



Event	Purpose	Max for 1 month
Sprint	Fixed-length container for all work; a new one starts right after the last	1 month
Sprint planning	Why (goal), what (items), how (plan)	8 h
Daily scrum	Inspect progress toward the Sprint Goal, adapt the plan	15 min
Sprint review	Show the increment to stakeholders, adapt the backlog	4 h
Retrospective	Inspect how the team worked, choose improvements	3 h

Shorter sprints get shorter events: Club Hub uses 4 h, 15 min, 2 h and 1.5 h. Refinement is an ongoing activity, not an event.

> Every event is an *inspect and adapt* point. Skip one and you lose a feedback loop.

Sprint planning for Club Hub Sprint 1

Sprint planning answers three questions (Scrum Guide 2020):

1. **Why is this sprint valuable?** The whole team writes one *Sprint Goal*: an outcome, not a list of stories.
2. **What can be done?** The Developers pull the top items of the ordered product backlog until the forecast fills their capacity.
3. **How will it get done?** Each item is split into tasks of one day or less; the goal, the items and the plan together form the sprint backlog.

⚠ Sprint 1 has no velocity history yet. Forecast carefully, and write down the assumption so the retrospective can check it.

☑ The goal lets the team swap tasks during the sprint without losing the point: if cancelling proves hard, a simpler cancel still meets the goal.

```
Sprint 1 · weeks 9-10 · team of 5 · PO: Dara · SM: Sokha
Sprint Goal: a student can see upcoming events, register for
             one and cancel again, from the command line.

Capacity: 5 people x 8 days x 2 h focus          = 80 h
          minus events, PO work, midterm review  = -12 h
          available                              = 68 h
Forecast: 13 points (no history: take the lower bound)

ID     Story                                        Points
US-04  As a student I list upcoming events             3
US-05  As a student I register for an event            5
US-06  As a student I cancel my registration           5
                                          Total       13
Tasks for US-05 (hours)
  [ ] test: cannot register twice                      1
  [ ] test: capacity cannot be exceeded                1
  [ ] RegistrationService::registerStudent             3
  [ ] CSV RegistrationRepository (save, findByEvent)   3
  [ ] CLI command: register <eventId> <studentId>      2
  [ ] review, fix, update README                       2
```

Daily scrum, sprint review and a Start/Stop/Continue retrospective

Daily scrum, day 4

15 minutes, same time and place, for the Developers. Any format that inspects progress toward the goal works; many teams use three prompts:

```
Rithy  yesterday: duplicate test red,
                  then green (US-05)
       today:     capacity rule test
       blockers:  none
Malis  yesterday: CSV parser for events
       today:     list-events CLI
       blockers:  date format unclear
                  -> ask PO after daily
```

Problems are *named* in the daily and *solved* right after it by the people involved.

Sprint review, day 10

- A working session with the client, not a slide show: run `clubhub list-events`, register, cancel.
- Show only what meets the definition of done; say openly what is not done and why.
- Collect feedback as new or changed backlog items: "show remaining seats", "waiting list when full".
- Discuss what to do next; the PO updates the backlog order.

A 2-hour review for a 2-week sprint is the maximum, not the target: 30 to 45 minutes is typical in class.

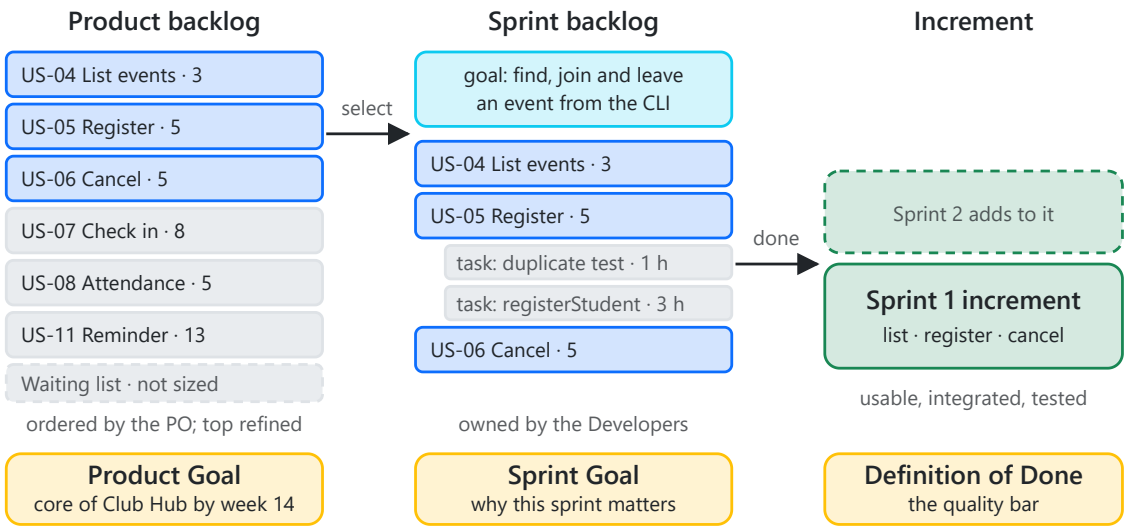
Retrospective, day 10

Start	writing the test before the code on every story; pairing on the CSV code
Stop	starting a new task while two are waiting for review; merging without running `ctest`
Continue	8:00 dailies that end on time; small pull requests

Two concrete actions for Sprint 2

1. Review column limited to 2 items; SM Vicheka checks it at each daily.
2. Add "ctest passed" to the pull request template; owner Rithy, by Monday of week 11.

Three artifacts, each with a commitment



- **Product backlog**: the single ordered list of everything the product might need. Items near the top are small and clear (refined); items at the bottom are coarse. The PO is accountable for its order.
- **Sprint backlog**: the Sprint Goal, the items selected for the sprint and the plan to deliver them. It changes daily as the Developers learn.
- **Increment**: a concrete, usable step toward the Product Goal. It adds to all earlier increments and is verified to work with them. Several increments may be produced within one sprint.
- **Commitments** make each artifact measurable: the Product Goal (long-term target), the Sprint Goal (this sprint's target) and the Definition of Done (when an item becomes part of the increment).

A definition of done for a C++ team

```
# Definition of Done · Team Angkor · v1.0
An item is Done when every box is ticked:
- [ ] Builds with GCC and MSVC with zero warnings
      (-Wall -Wextra -Werror or /W4 /WX)
- [ ] Each acceptance criterion has a GoogleTest
      test; ctest passes on a clean build
- [ ] No raw new/delete; clang-format applied
- [ ] Reviewed and approved in a pull request by
      another developer, then merged to main
- [ ] Public functions documented in the header;
      README and CLI help updated
- [ ] PO has seen it work against the criteria
Sprint 2 adds: CI pipeline green on main (Lab-08)
```

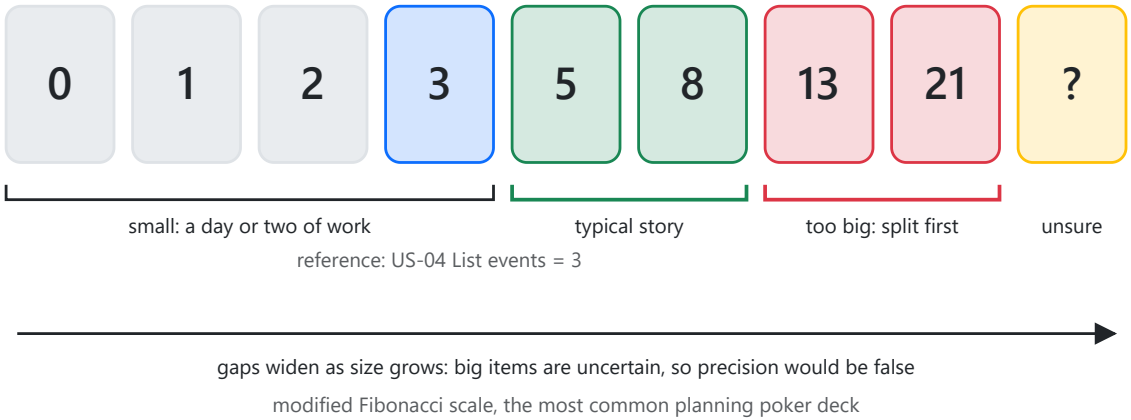
```
# CMakeLists.txt: make the DoD enforceable
if(MSVC)
  target_compile_options(clubhub_core PRIVATE /W4 /WX)
else()
  target_compile_options(clubhub_core PRIVATE
    -Wall -Wextra -Werror)
endif()
enable_testing()
include(GoogleTest)
gtest_discover_tests(clubhub_tests)
```

- The **DoD** applies to *every* item; **acceptance criteria** are specific to *one* story. An item needs both to count.
- An item that does not meet the DoD at the end of the sprint is not shown as done: it returns to the product backlog.
- Automate what you can (compiler flags, tests); keep the rest as a short checklist in the pull request template.

The DoD grows with the team: CI arrives in Chapter 08, coverage targets in Chapter 09.

TOPIC 5

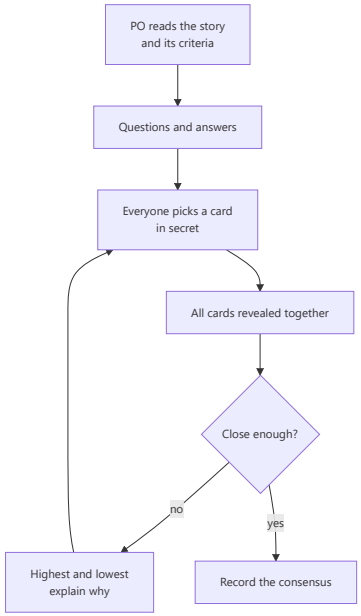
Story points: relative size, not hours



- A **story point** is a unit of relative size that blends effort, complexity and uncertainty. "US-05 is about as big as US-06 and bigger than US-04."
- Pick a **reference story** everyone understands (US-04 = 3) and size others against it. People compare much better than they predict hours.
- Points belong to *one team*: 5 points in team A says nothing about team B.
- The **whole team** estimates, because the work includes coding, testing and review.
- Hours are still useful for *tasks* inside a sprint (capacity check), not for backlog items.

⚠ A story of 13 or 21 does not fit comfortably in a two-week student sprint. Split it (for example "reminder by email" and "reminder in the CLI") before planning.

Planning poker: divergent estimates are the point



Revealing at the same time avoids *anchoring*: nobody copies the first number spoken aloud. Technique popularised by Mike Cohn (2005).

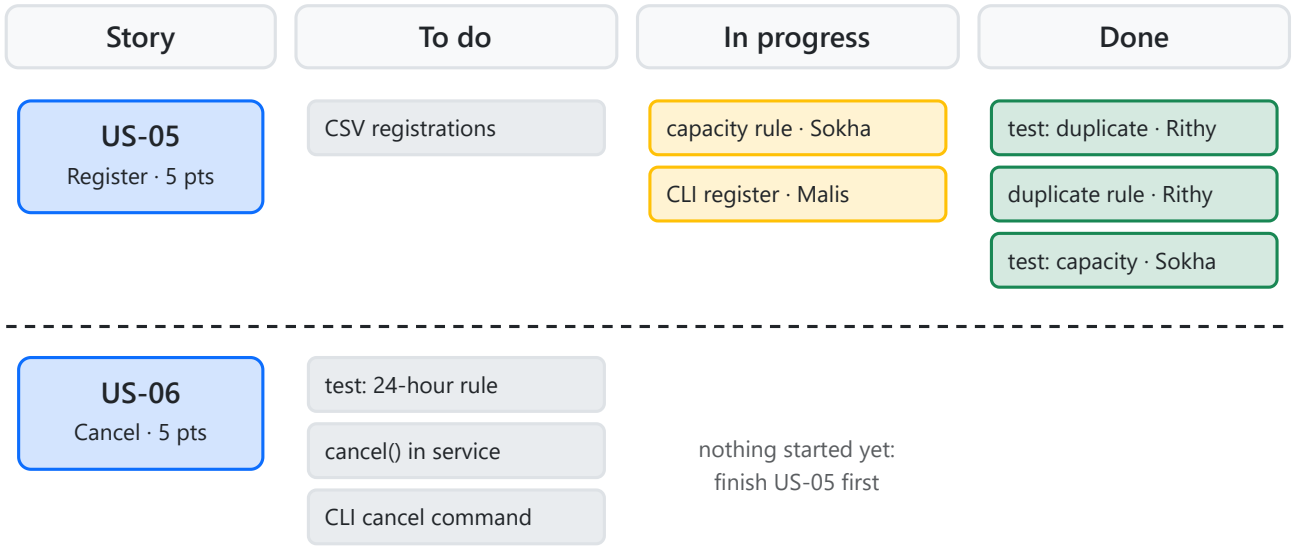
US-06 Cancel my registration	Dara	Sokha	Vicheka	Rithy	Malis
Round 1	3	3	5	13	2
Round 2	5	5	5	8	5
Consensus	**5 points**				

Discussion after round 1

- **Malis (2)**: "Cancel is one line: set the status to `Cancelled`."
- **Rithy (13)**: "The 24-hour rule needs the event start and the current time, and the CSV file must be rewritten, not appended. We also have no test for time yet."
- The team agrees to inject the current time as a parameter (easy to test) and to reuse the CSV writer from US-05. Round 2 converges.

The estimate improved because hidden work (time rule, file rewrite) became visible. Record it as tasks.

The task board makes the sprint backlog visible

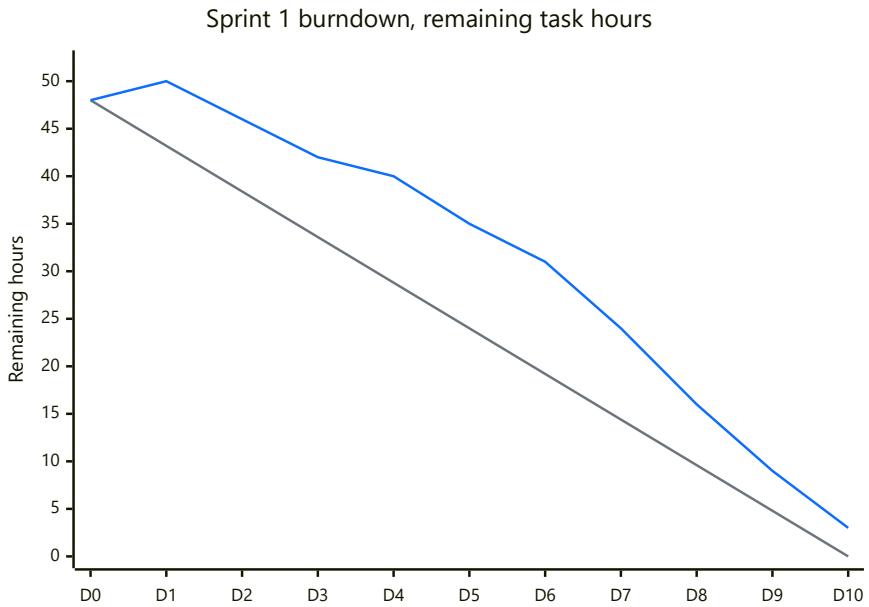


one swimlane per story · one sticky note per task · a name on every card in progress

- A **task board** shows every sprint backlog item and its tasks in columns by state. Anyone can see the sprint at a glance: transparency.
- Cards move **left to right**; the developer who takes a task puts their name on it.
- Update the board **before** the daily scrum so the daily discusses reality.
- Work on one story at a time (*swarming*): three half-finished stories are worth zero points at the review.
- Tools: a whiteboard with sticky notes, or GitHub Projects (Lab-07), Jira, Trello.

💡 Add a *Review* column when pull requests wait for a reviewer: the queue becomes visible.

The sprint burndown: remaining work per day

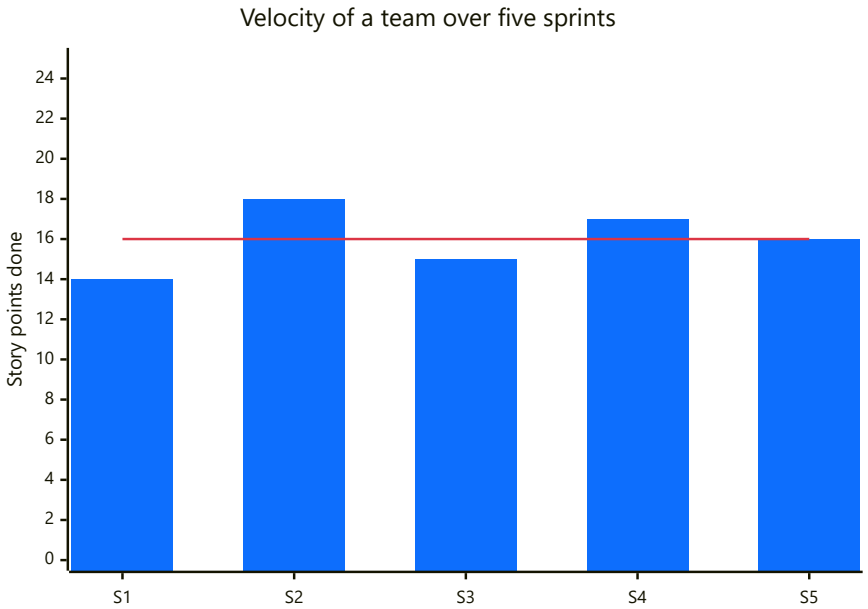


Grey straight line: ideal burn (48 h / 10 days = 4.8 h per day). Blue line: actual remaining hours taken from the board every evening.

- **How to build it**: after sprint planning, sum the task estimates (48 h). Each day, sum the hours still open on the board and plot the point.
- **Day 1 goes up** (48 to 50 h): the team discovered a task, the CSV file must be rewritten on cancel. New work found is normal and should be visible.
- **Actual above ideal** = behind. On day 5 the ideal is 24 h, the actual 35 h: 11 h late. The daily scrum should re-plan now, not on day 9.
- **Day 10 ends at 3 h**: the CLI cancel command is unfinished, so US-06 is not done. It returns to the product backlog; the sprint delivers 8 of 13 points.

⚠ A burndown is a conversation tool for the team, not a performance report. Never "correct" the data to follow the ideal line.

Velocity: how many points a team finishes per sprint



Bars: points of stories that met the DoD. Red line: the average, 16 points.

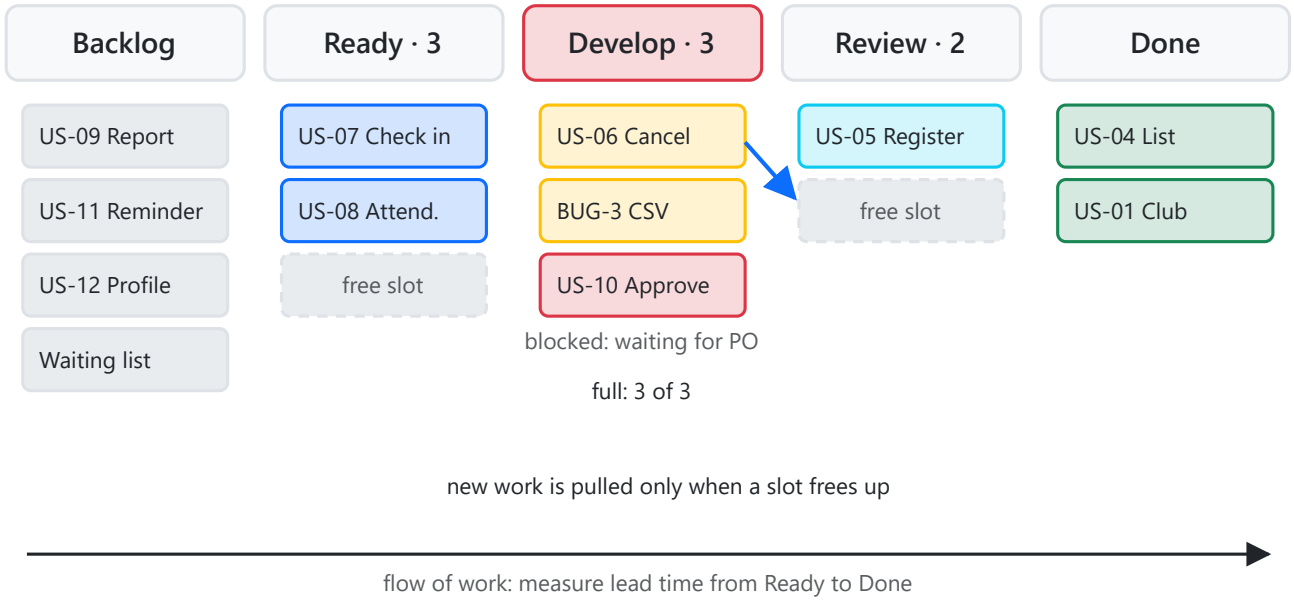
```
Remaining product backlog      60 points
Velocity range (last 5 sprints) 14 to 18 points

Best case   60 / 18 = 3.3  -> 4 sprints
Worst case  60 / 14 = 4.3  -> 5 sprints
Forecast    4 to 5 sprints = 8 to 10 weeks
            (2-week sprints; round up, a
             partial sprint is still a sprint)
```

- **Velocity** counts only *done* stories; partly finished work counts zero.
- Forecast with a **range**, never a single number; the range narrows as history grows.
- Use it for the team's own planning. Comparing teams by velocity, or setting it as a target, makes points inflate.

💡 Club Hub has only two sprints. If Sprint 1 finishes 8 points, the Sprint 2 forecast is about 8 to 12 points: the PO must choose which Must stories fit.

Kanban: visualise the flow and limit work in progress



- **Visualise** every step of the workflow as a column.
- **Limit WIP**: the number on each column is the maximum number of cards in it. When Develop is full, a developer helps finish or unblock a card instead of starting a new one.
- **Manage flow**, make **policies explicit** (what "Ready" means), use **feedback loops** and **improve** step by step.
- No sprints and no prescribed roles: work is *pulled* continuously.

```
Little's Law: lead time = WIP / throughput
WIP 6 cards, 3 cards done per week -> 2 weeks
WIP 3 cards, same throughput       -> 1 week
```

Scrum and Kanban compared

Aspect	Scrum	Kanban
Cadence	Fixed-length sprints (1 to 4 weeks)	Continuous flow; cadences (for example weekly replenishment) are optional
Roles	Product Owner, Scrum Master, Developers	None required; keep your current roles
Limit on work	The sprint forecast limits WIP per sprint	Explicit WIP limit per column
Changing priorities	New items wait for the next sprint (the goal protects focus)	Any time: the next free slot pulls the top item
Key metrics	Velocity, sprint burndown	Lead time, cycle time, throughput, cumulative flow diagram
Board	Reset every sprint	Persistent
Fits best	New product development toward a goal, with regular reviews	Support, maintenance, operations: unpredictable arrivals of small items
Club Hub use	Sprints 1 and 2 (weeks 9 to 12)	Bug fixes after the week-14 demo; the WIP-limit experiment in Lab-07

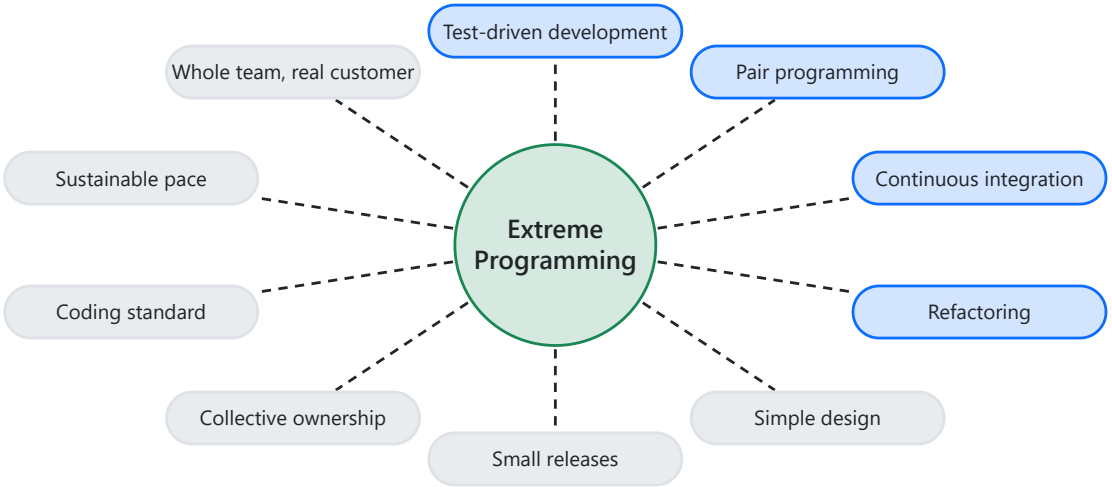
Scrumban

Many teams combine both: Scrum's roles, sprints and review, plus Kanban's WIP limits and flow metrics on the board. The Scrum Guide allows any complementary practice.

Lead time: from request to done (what the customer feels). **Cycle time**: from work started to done (what the team controls).

⚠ A board with columns but no WIP limits is not Kanban, just a to-do list.

XP: engineering practices that make short iterations safe

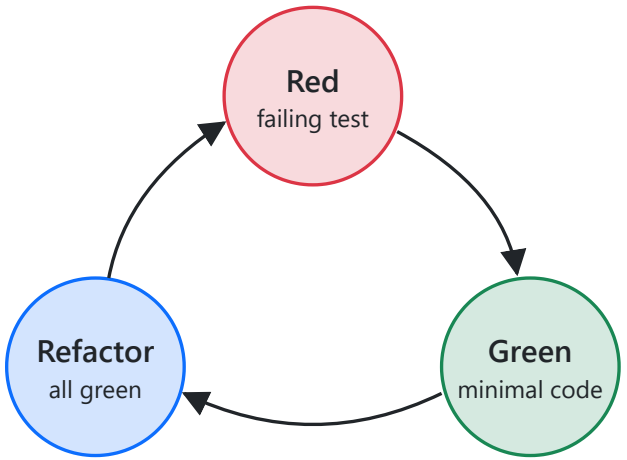


Practice	In one line
Pair programming	Two people, one keyboard: a **driver** types and handles the detail, a **navigator** reviews each line, thinks ahead and spots missing tests. Swap every 20 to 30 minutes.
TDD	Write a failing test, make it pass, clean up (next two slides).
Continuous integration	Merge small changes to main at least daily; an automated build and test run checks every merge (Chapter 08).
Refactoring	Improve the structure without changing behaviour, protected by the tests.

```
git commit -m "Add capacity rule" \
  -m "Co-authored-by: Sokha <sokha@example.com>"
```

The co-author trailer keeps both partners visible in the Git history.

Red: write a test for "cannot register twice" first



a few minutes per cycle

- The test states the rule from the acceptance criterion before any code exists.
- Run it and **watch it fail**: that proves the test can detect the missing behaviour.
- Commit the red test: `test: cannot register twice (red)`.

```cpp
// tests/registration_service_test.cpp
#include <gtest/gtest.h>
#include "InMemoryRegistrationRepository.hpp"
#include "clubhub/RegistrationService.hpp"

using namespace clubhub;
using clubhub::testing::InMemoryRegistrationRepository;

TEST(RegistrationService, StudentCannotRegisterTwice) {
    InMemoryRegistrationRepository repo;  // test double, no file
    RegistrationService service{repo};
    Event event;
    event.id = "E1";
    event.capacity = 30;
    service.registerStudent(event, "S001");
    EXPECT_THROW(service.registerStudent(event, "S001"), DuplicateRegistration);
}
```

```
$ ctest --test-dir build --output-on-failure
[ RUN      ] RegistrationService.StudentCannotRegisterTwice
registration_service_test.cpp:16: Failure
Expected: service.registerStudent(event, "S001") throws an exception
  of type DuplicateRegistration.
  Actual: it throws nothing.
[  FAILED  ] RegistrationService.StudentCannotRegisterTwice
```

Green with the simplest code, then refactor under green tests

Green just enough code to pass; commit `feat: reject duplicate registration`

```
Registration RegistrationService::registerStudent(
    const Event& event, const std::string& studentId) {
    for (const auto& r : registrations_.findByEvent(event.id)) {
        if (r.studentId == studentId) {
            throw DuplicateRegistration{"already registered"};
        }
    }
    Registration reg{event.id, studentId,
                     RegistrationStatus::Registered};
    registrations_.save(reg);
    return reg;
}
```

💡 Next red test from the same story: "capacity cannot be exceeded", then "a cancelled student may register again". Each rule gets its own cycle.

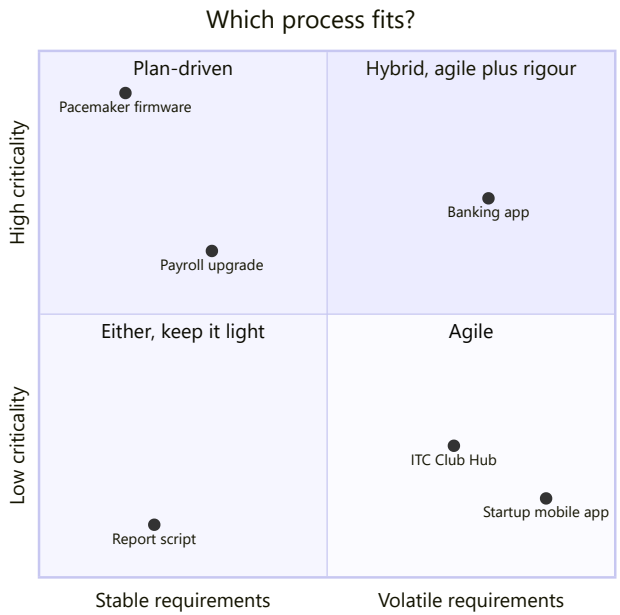
Refactor same behaviour, clearer code; commit `refactor: extract isRegistered`

```
bool RegistrationService::isRegistered(
    const Event& event, const std::string& studentId) const {
    return std::ranges::any_of(
        registrations_.findByEvent(event.id),
        [&](const Registration& r) {
            return r.studentId == studentId;
        });
}

// registerStudent now reads like the business rule
if (isRegistered(event, studentId)) {
    throw DuplicateRegistration{
        studentId + " is already registered for " + event.id};
}
```

- Run `ctest` after every small change; if a test turns red, undo the last step.
- Refactoring never adds behaviour. New behaviour always starts with a new red test.

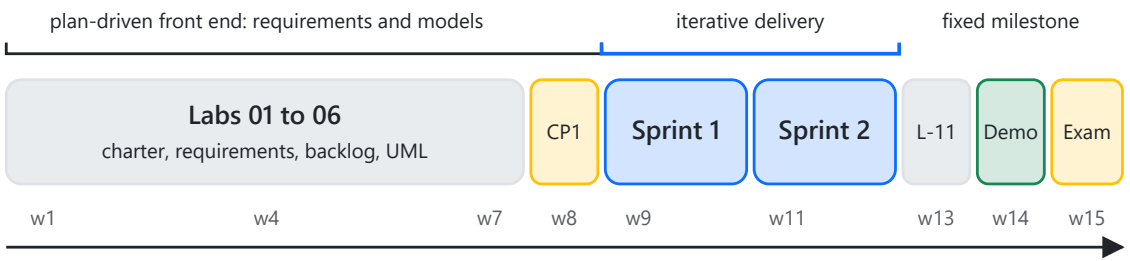
When does agile fit, and when does a plan-driven process fit?



Factor	Agile home ground	Plan-driven home ground
Criticality	Failure costs comfort or money that can be recovered	Failure can cost lives or large sums
Dynamism	Requirements change often	Requirements are stable and known
Size	Small co-located teams	Many teams, many sites
Personnel	Experienced people who can decide	Mixed experience, need written guidance
Culture	People like freedom and change	People like order and clear roles

Also check contracts and regulation: a fixed-price tender or a certification standard (medical, aviation) may demand upfront documents and formal approval.

Hybrid approaches: most real projects mix both



CP1 = project checkpoint 1 in the midterm week: requirements and models reviewed

the scope inside the sprints flexes; the week-14 date and the quality bar do not

Hybrid	How it mixes
Water-Scrum-Fall	Upfront requirements and a formal release phase, sprints in the middle. Common in large organisations.
Stage gates + sprints	Management approves at fixed gates (budget, go-live); teams work in sprints between gates.
Scrumban	Scrum events with Kanban WIP limits and flow metrics.
Scaled frameworks	SAFe, LeSS, Disciplined Agile coordinate many teams; covered in the IT Project Management course.

⚠ A hybrid is a deliberate choice for a reason (regulation, contract, fixed date). "We kept the old process and renamed phases to sprints" is an anti-pattern, not a hybrid.

TOPIC 10

Common agile anti-patterns and how to fix them

Anti-pattern	Symptom	Fix
Scrum-but	"We do Scrum, but we skip the retrospective / have no definition of done / run 6-week sprints." The removed part is usually the one that exposes a problem.	Keep every event and artifact; shorten an event rather than drop it. Ask what problem made the part feel useless and fix that problem.
Sprint as mini-waterfall	Days 1 to 3 analysis, 4 to 7 coding, 8 to 10 testing. Nothing is done until the last day; the burndown is flat and then falls off a cliff; bugs roll into the next sprint.	Slice stories vertically (CLI + service + file for one small rule), write tests first, swarm on one story at a time, add a WIP limit.
No real Product Owner	Nobody can say what matters most; developers guess; a committee reorders the backlog each week; the review has no one to accept the work.	Name one PO with authority over the order. In Club Hub the PO meets the client (instructor) every week and accepts stories against their criteria.
Daily scrum as status report	Everyone reports to the Scrum Master; nobody listens to each other; blockers stay unsolved for days.	Talk to the team about the Sprint Goal; walk the board right to left; solve problems right after the daily.
Velocity as a target	"Next sprint you must do 20 points." Estimates inflate, quality drops, velocity stops meaning anything.	Use velocity only for the team's own forecast. Measure outcomes (working features, client feedback) instead.
Hardening sprint	A special sprint "to test and fix everything" before release: the DoD was too weak.	Strengthen the DoD so every increment is releasable; fix defects inside the sprint that created them.

💬 Test for any practice: does it still let the team *inspect* real working software and *adapt* within a sprint? If not, it has drifted from agile.

Making Sprint 1 work in a student team

1 Goal first, stories second

Write the Sprint Goal as an outcome a client can see. Mistake: a goal that is only a list of story ids.

2 Small, vertical stories

Each story delivers a thin working slice through CLI, service and storage. Mistake: stories such as "write all the headers".

3 Plan with honest capacity

Count real hours per person and subtract exams and events. Mistake: committing to the whole Must list in Sprint 1.

4 Done means the DoD

Only items that build without warnings, pass their tests and are reviewed count. Mistake: "done except testing".

5 Keep the board true

Move cards when work moves, before the daily. The burndown is built from this data. Mistake: updating the board once, the night before the review.

6 Retrospective actions with owners

Two concrete actions with an owner and a date beat ten vague wishes. Mistake: "communicate better" as an action.

Chapter quiz 10 questions

1. Which pair is one of the four values of the Agile Manifesto, in the right order (valued more over valued less)?

- Comprehensive documentation over working software
- Responding to change over following a plan
- Processes and tools over individuals and interactions
- Contract negotiation over customer collaboration

2. In Scrum, who is accountable for ordering the product backlog?

- The Scrum Master
- The Developers, by vote
- The Product Owner
- The client or instructor

3. What is the timebox of the daily scrum?

- 5 minutes
- 15 minutes
- 30 minutes
- 1 hour for a one-month sprint

4. At the end of a sprint a story is coded and works, but it has no tests although the definition of done requires them. What happens?

- It is shown as done and tested next sprint
- It counts as half its points
- It is not done: it returns to the product backlog
- The Scrum Master extends the sprint by two days

5. In planning poker for a story, the votes are 3, 3, 5, 13 and 2. What should the team do next?

- Take the average, 5.2, and round to 5
- The Product Owner decides the number
- Ask the people who voted 13 and 2 to explain, then vote again
- Pick 13 to be safe

6. A team has 60 points left in the backlog and a velocity between 14 and 18 points per sprint. What is a sound forecast?

- Exactly 3.75 sprints
- 3 sprints
- 4 to 5 sprints
- 6 to 8 sprints

7. On day 5 of a sprint burndown the ideal line shows 24 remaining hours and the actual line shows 35. What does it tell the team?

- They are ahead of plan
- They are about 11 hours behind and should re-plan at the daily scrum
- The ideal line must be redrawn to match
- Nothing: burndowns are read only at the end

8. A Kanban board has a WIP limit of 3 on "Develop" and it holds 3 cards. A developer has finished a review. What should they do?

- Pull a new card into Develop anyway
- Help finish or unblock a card already in Develop
- Raise the WIP limit to 4
- Wait for the next sprint planning

9. In TDD, you wrote the test EXPECT_THROW(service.registerStudent(event, "S001"), DuplicateRegistration) and it fails. What is the next step?

- Write the simplest code that makes this test pass
- Refactor the whole RegistrationService first
- Delete the test because it fails
- Write five more tests before any code

10. Which project is the strongest candidate for a plan-driven (or heavily plan-driven hybrid) process?

- A startup mobile app still searching for its users
- A student club app built by a team of five
- Pacemaker firmware with stable requirements and certification
- A small internal report script

WRAP-UP

Summary

- The Agile Manifesto values people, working software, customer collaboration and responding to change more than their counterparts, which still have value.
- Scrum has three accountabilities: the Product Owner orders the backlog, the Scrum Master makes Scrum work, the Developers build the increment.
- Five events (sprint, planning, daily scrum, review, retrospective) are inspect-and-adapt points with fixed timeboxes.
- Product backlog, sprint backlog and increment carry the commitments Product Goal, Sprint Goal and Definition of Done.
- Story points size work relatively; planning poker surfaces hidden work when estimates diverge.
- Task boards, burndowns and velocity make progress visible; forecast with a range, never with a single number.
- Kanban limits work in progress so work flows: stop starting, start finishing.
- XP practices (TDD, pair programming, CI, refactoring) keep short iterations safe; in TDD the red test comes first.
- Choose agile, plan-driven or a hybrid by criticality and volatility, and watch for anti-patterns such as Scrum-but and mini-waterfalls.

Open Lab-07: 5 tasks + 1 challenge

Next chapter: 08 · DevOps Practices