

Exercise walk-through

12.2

Architecture and design

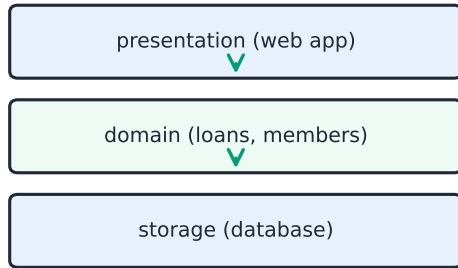
University Computer Science

Architecture: layers pointing inward

- 1 **Layers** Presentation (web), domain (loans), storage (database); dependencies point inward, never outward.
- 2 **Counts** Loan-rule change: 1 module, 2 files; smeared across presentation: 3 **layers**, 5 files - the count IS the metric.
- 3 **Sequence** Presentation calls domain.borrow; domain checks limit and status; storage records with dates - four participants, five labelled messages.

Architecture and sequence diagrams

layers, inward arrows



dependencies point inward

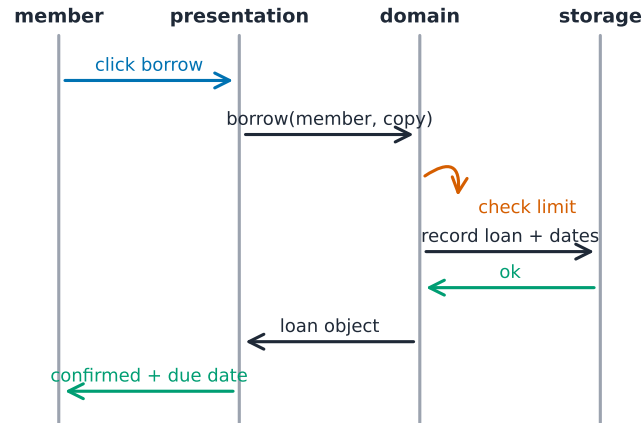
the domain knows no framework

loan-rule change: 1 module, 2 files

smeared version: 3 layers, 5 files

Notifier sits behind an interface

the borrow sequence



Three **layers** with the inward rule; the borrow sequence labelled.

Exercises 2.1-2.3

2.1 Which layer owns the limit rule, and why there? [3]

2.2 What does “the domain knows no framework” buy? [3]

2.3 Name the four participants of the borrow sequence. [3]

Exercises 2.1-2.3

2.1 Which layer owns the limit rule, and why there? [3]

2.1

- The domain – the limit is a loan rule, not a screen concern ✓.

2.2 What does “the domain knows no framework” buy? [3]

2.2

- Testability without browser or database, and swap of outer layers ✓.

2.3 Name the four participants of the borrow sequence. [3]

2.3

- Member, presentation, domain, storage ✓.

Exercise 3.1

3.1 A teammate proposes moving the limit check into the presentation layer “because the button is there”.

State which principle is violated and the cost, using the file counts. Give the layered design (one sentence per layer’s job). Redraw the sequence in words if the check moves: name the new message flow and what storage must now validate. [8]

Exercise 3.1

3.1 A teammate proposes moving the limit check into the presentation layer “because the button is there”.

State which principle is violated and the cost, using the file counts. Give the layered design (one sentence per layer’s job). Redraw the sequence in words if the check moves: name the new message flow and what storage must now validate. [8]

3.1

- Violated: the dependency rule (dependencies point inward) and cohesion – the loan rule leaves the loans module ✓.
- Cost: the same rule now touches 3 layers, 5 files instead of 1 module, 2 files – every change multiplies ✓.
- Layered design: presentation renders and collects input; domain decides (limits, statuses, dates); storage persists and reads ✓.
- Moved check: presentation calls storage directly; storage must now validate the limit (rules leak into the database layer) – the sequence gains a presentation-storage arrow and the domain is bypassed ✓.

Exercise 3.2

3.2 Reminder delivery today is email; the product wants push for mobile users.

State where the Notifier boundary sits and why the domain never names email or push. Give the swap's cost and the suite's proof that FR-3 still holds. Name the design metric this improves. [7]

Exercise 3.2

3.2 Reminder delivery today is email; the product wants push for mobile users.

State where the Notifier boundary sits and why the domain never names email or push. Give the swap's cost and the suite's proof that FR-3 still holds. Name the design metric this improves. [7]

3.2

- The Notifier interface sits at the domain's edge; delivery implementations live outside – the domain calls the interface only ✓.
- The domain naming email would couple it to a channel that changes; naming nothing keeps the rule pure ✓.
- Swap cost: one configuration line (register push as the implementation); zero domain edits ✓.
- The suite's proof: swap in a test double notifier, run FR-3's acceptance test (reminder count = overdue days) – still green ✓.
- Metric: coupling down (the domain depends on an abstraction, not a channel) ✓.