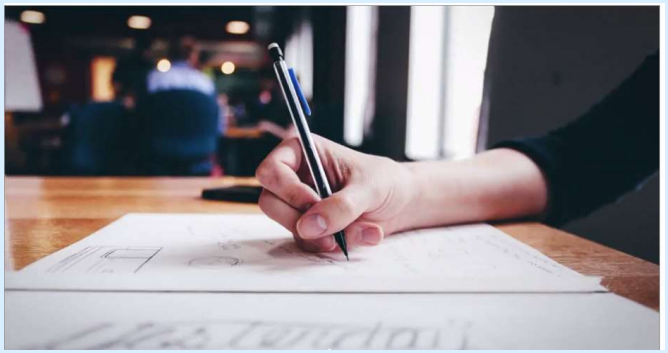
FARABI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Object-Oriented Analysis and Design

Lecture 3
Design in the Software Process



Dr. Bazargul Matkerim, Professor-Researcher

Learning Outcomes

Distinguish

Explain conceptual and technical design.

Refine

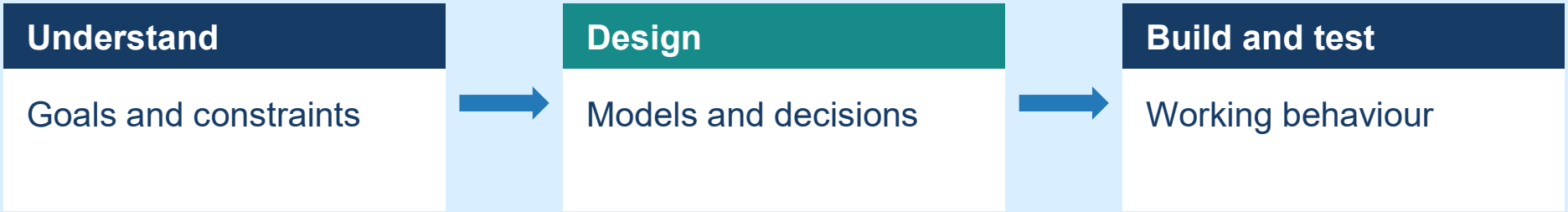
Translate requirements into object responsibilities.

Evaluate

Use scenarios and tests to review a design.

Learning task: develop a course-search feature from requirement to design.

Design in an Iterative Software Process



Feedback from users, prototypes and tests may change requirements and design decisions during the next iteration.

Design continues as the team learns more about the problem.

Requirements Begin with Questions



- Who will use the system, and for what goal?
- Which behaviours and constraints are essential?
- What would count as an acceptable result?

Elicitation turns assumptions into questions that stakeholders can answer.

Conceptual and Technical Design

Conceptual design

Describe the proposed solution
from the stakeholder's perspective.

Artifacts: sketches, mockups,
major components and connections.

Technical design

Explain how the solution will work.

Artifacts: classes, interfaces,
interaction diagrams and
data-access decisions.

These views inform each other; a technical constraint can change a concept.

House Design: Components and Connections



- Components: kitchen, bedrooms, living room.
- Connections: doors and passages between rooms.
- Responsibilities: cooking, sleeping, shared activities.

Software components also need clear purposes and connections.

Technical Details Can Reveal Conflicts



Desired concept

A gym above a wide,
open basement.

Technical concern

Extra supports may reduce
open space or headroom.

Explore feasible alternatives and discuss their consequences with the client.

Stakeholder Feedback Refines the Design



- Show the proposed experience using a mockup.
- Ask stakeholders to walk through a realistic task.
- Record decisions and revise unclear or missing details.

Early feedback can reveal misunderstandings before substantial implementation.

Case Study: Searching for Courses

“As a learner, I want to search for courses by keyword so that I can find a relevant course.”

Scope for this example

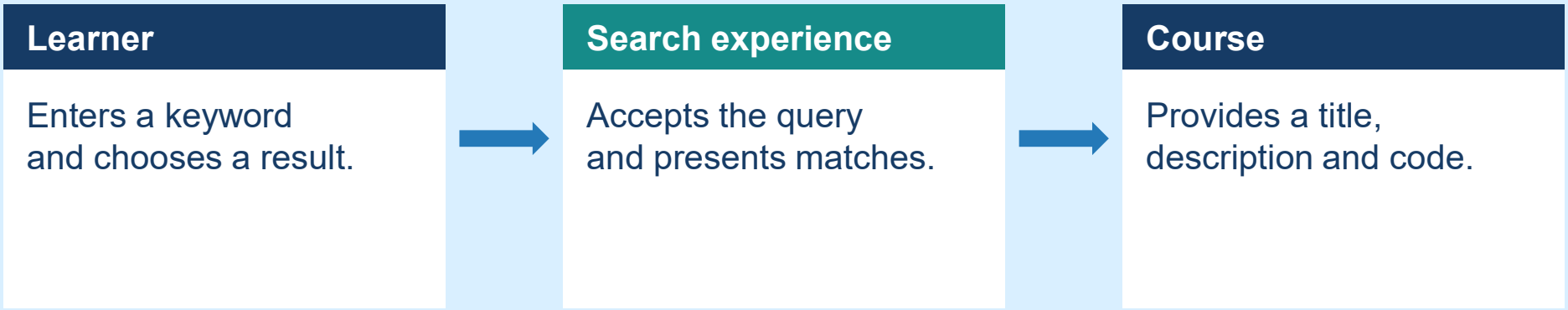
Search course titles and descriptions.
Show matching course summaries.

Rules to confirm

What happens for an empty query?
How are results ordered?
Which courses are visible?

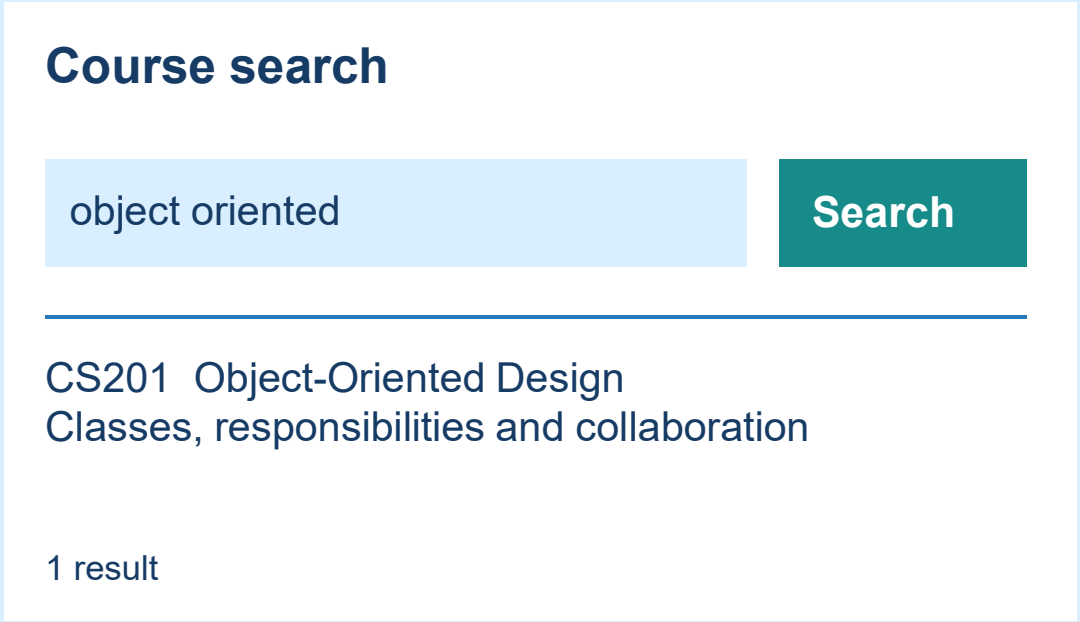
A short requirement starts the discussion; it does not settle every design rule.

A Conceptual Model of Course Search



Conceptual design focuses on what the user sees and needs to accomplish.

A Mockup Makes Interaction Concrete



- Input: where is the keyword entered?
- Action: how does the learner start the search?
- Output: how are matching courses displayed?

A mockup validates the interaction without committing to implementation details.

Alternative Outcomes Need Design Too

Empty query

Prompt the learner to enter a keyword.

No matches

Show a clear message and allow another search.

Service unavailable

Explain that search failed and offer a retry.

Proposed behaviours: confirm these rules with stakeholders before implementation.

From a Mockup to Object Responsibilities

SearchPage

Collect input
Display results and errors

CourseSearchService

Validate the query
Coordinate the search

CourseRepository

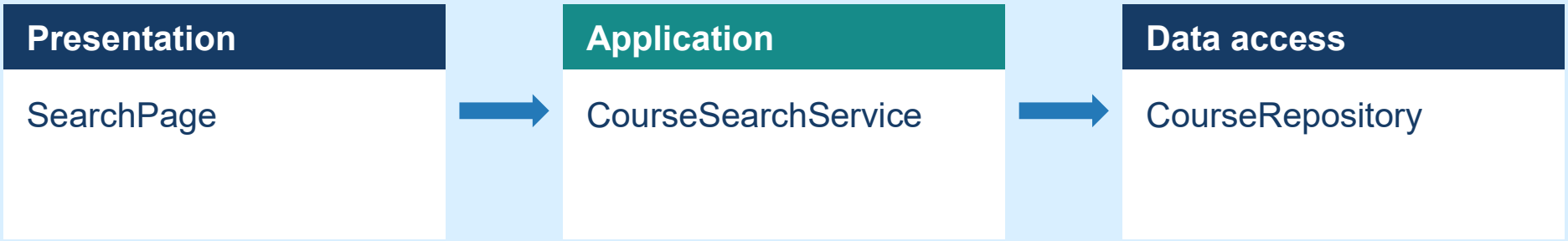
Retrieve matching
course data

Course

Represent course identity
and descriptive data

The page delegates search work; each object has a focused responsibility.

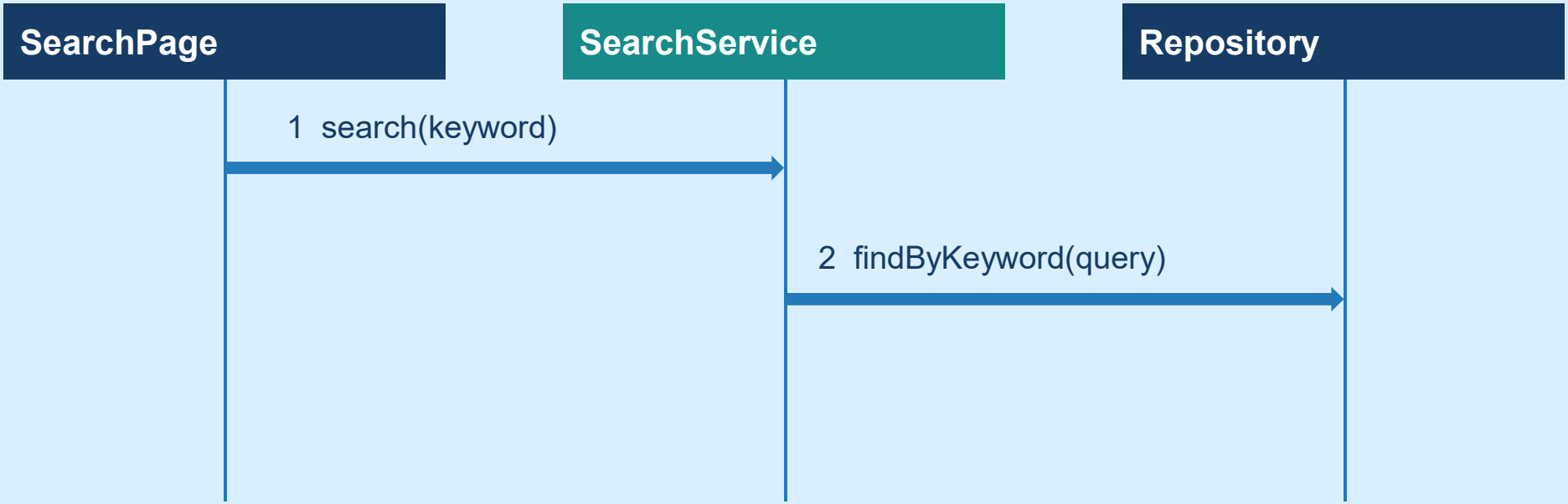
A Technical View of Course Search



Request flows to the repository; matching course data returns to the page.

This is one possible design. The repository hides data-access details.

A Search Interaction



3 Return matching courses; the page renders the result list.

Interfaces Make Collaboration Explicit

CourseSearchService

+ search(keyword: String): List<Course>

CourseRepository

+ findByKeyword(query: String): List<Course>

Input: a keyword string.

Output: a list of matching courses.

Contract: define empty-query and failure behaviour explicitly.

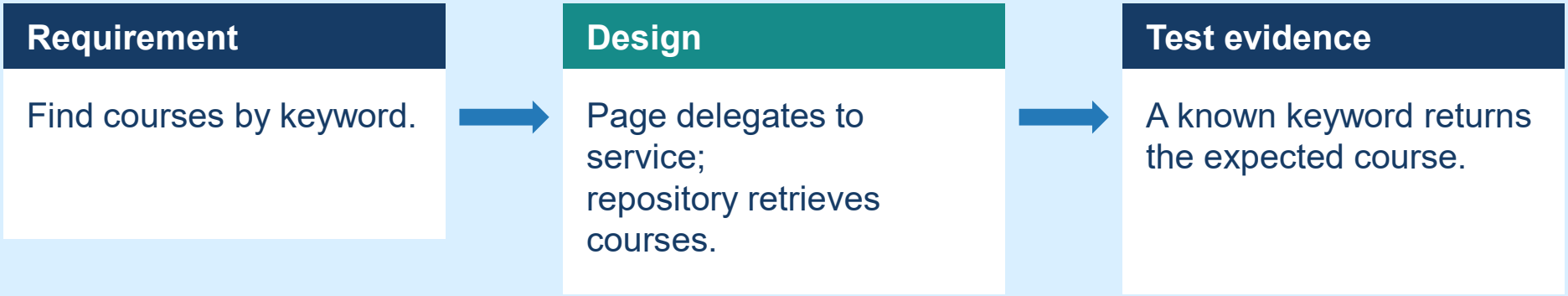
Operation signatures describe collaboration; contracts clarify the expected behaviour.

Design Decisions Have Consequences

Decision to make	Consequences to consider
Local catalogue or external service?	Data ownership and availability
Keyword matching or full-text search?	Search quality and complexity
Live retrieval or cached results?	Response time and data freshness

Record the chosen option, its rationale, and the assumptions behind it.

Traceability Connects Design and Tests



Also check empty input, no matches and repository failure.

Validation asks whether the feature meets the need; verification checks its specification.

Practice: Add a Course Filter

New requirement: learners can filter search results by course level.

Revise the concept

Add the filter to the mockup.
Explain how it affects results.

Refine the technical design

Update the service and repository operations, then specify two tests.

Discuss what changes, what stays stable, and which assumptions need confirmation.

Review and Key Ideas

Requirements express goals and constraints.
Conceptual models explain the proposed experience.
Technical models allocate work to classes and interfaces.
Feedback and tests help the design evolve.

Exit question: why can a convincing mockup still hide an incomplete technical design?