For personal use only,
please do not
distribute on
non-UBC domains.

09b - Modelling & Specification

CPSC 310

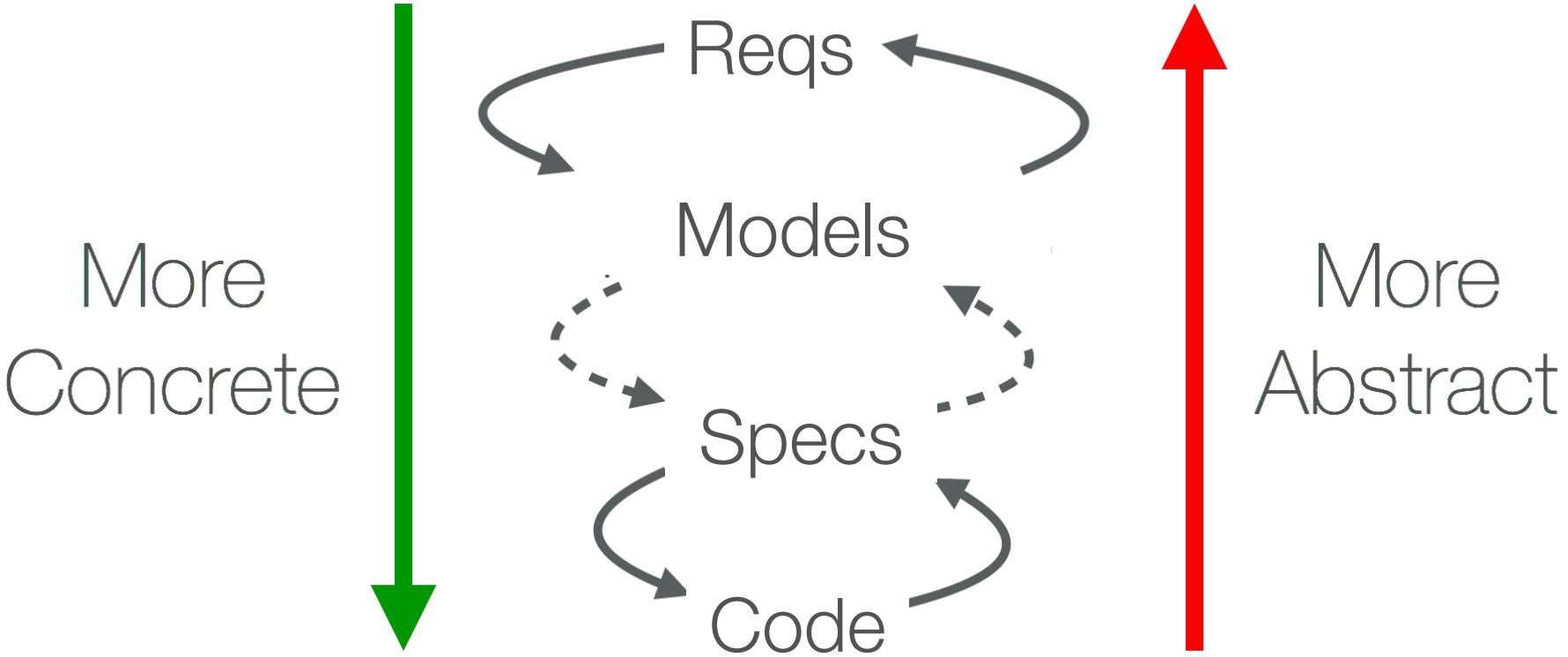
Nick Bradley & Kyle Chin

Learning goals

By the end of this lecture, you should be able to:

- Model components from a stakeholder's request.
- Check a model by walking through scenarios.
- Specify an operation's precondition, postcondition and exceptional cases.

Moving Between Representations



From Requirements to Specifications

- **Requirements:** stakeholder needs in natural language
  - Describe goals, constraints, and priorities
  - Checked by (non-technical) stakeholders
- **Models**: classes, operations, and rules the requirements imply
  - Decides what there is to specify, and finds what the requirements omit
  - Checked by walkthroughs
- **Specifications:** precise documentation and tests
  - What developers use to write the code and tests
  - Checks the code: the tests run against the implementation

What Models Capture

A model records:

1. **Structure**: classes and fields
2. **Operations**: what can be done to each class
3. **Rules**: what is always true and the classes and operations

It says what exists and what's allowed, not how anything works.

On paper, a model is a sketch: a box for each class, with its fields, operations, and rules. You check that it works by tracing through scenarios.

Modelling Procedure

Modelling is iterative:

1. **Extract**: identify candidate classes, fields, operations, and rules from the text.
2. **Shape**: determine ownership and encapsulation; assign each rule to a class or operation.
3. **Evaluate**: trace the model through scenarios.
4. **Revise or ask**: correct the model, or record unspecified cases as decisions or questions.

The model is complete when no scenario fails and every open question requires stakeholder's input.

Reading a request

The grammar hints at what to model:

In the text	**Usually becomes…**
A noun (or an adjective that classifies a noun)	An **class** or **field**
A verb someone does	An **operation**
a limit, condition, or consequence	A **rule**

Each section has a fixed number of seats. Once it's full, students can join the section's waitlist, which holds at most 50 students. A student can't join the waitlist for a section they're already enrolled in, and can't join the same waitlist twice. Students can leave the waitlist whenever they like, and can check their position on it at any time.

When a seat frees up, the first student in line is offered it and has 24 hours to accept. If they accept, they're enrolled and come off the waitlist. If they decline, or let the offer expire, they come off the waitlist and the seat goes to the next student in line.

Students' names and email addresses live in the Student Information System, not with us. The same student is often waiting for several sections at once.

Class, field, or operation?

For each noun: is it just a value of something else, or does it have data or rules of its own?

- Just a value (a count, a name, a Date): **field**
- Has its own data, or rules about how it changes: **class**
- Can always be computed from other state: **operation**

Seats: in a lecture hall they're interchangeable (so a count field).
At a concert each has a number, a price, and an occupant (class).

Ownership and Encapsulation

When class A has a field of class B, two questions arise:

1. **Ownership**: does A control B's lifetime?
   - Yes: A creates B directly (composition)
   - No: A accepts an instance of B created elsewhere (association)
2. **Encapsulation**: does a rule about A depend on B?
   - Yes: operations that could break the rule must be defined on A
     → That ensures A can enforce the rule
   - No: operations that change B can be defined on B directly

Adding Rules to the Model

Rules from the request become:

- **Invariants**: what's always true about an object between operations.
- **Preconditions**: what must be true before an operation is called (the caller's job).
- **Postconditions**: what an operation guarantees afterwards, including what doesn't change.
- **Exceptional cases**: what an operation checks, and the result it gives when it can't proceed

One sentence often gives both an invariant and rule(s) on the operation that maintains it.

Walkthroughs: testing models by hand

1. Create scenarios based on the request:

   "*Three students enrol into a full section, a seat becomes free and the first student on the waitlist accepts the offer.*"

2. Check that your model can handle each scenario:
   - Write the state after every call
   - Check that the scenario works only through the fields and operations in your sketch
   - Stuck? Either the model has a gap (change it), or the requirement doesn't say (decide behaviour, or ask).

Walkthrough example

"Three students enrol into a full section, a seat becomes free and the first student on the waitlist accepts the offer."

Operation	Waitlist	Offer
Add Ada	Ada	
Add Ben	Ada, Ben	
Add Cy	Ada, Ben, Cy	
A student drops	Ada, Ben, Cy	Ada (24 hrs)
Ada accepts (2h)	Ben, Cy	

Documentation: stating the rules

```
/**
* The waitlist for one section: students in the order they joined.
* @remarks
* Always true: no student is in line twice, and the line never exceeds its maximum length.
*/
export interface Waitlist {
   /**
    * Adds a student to the back of the line. Call only once the section is full.
    * @param id - A valid student id; not checked.
    * @returns `{ ok: true, position }`: the student is last in line; nobody else moves.
    *   Otherwise `{ ok: false, reason }`, unchanged, for the first that applies:
    *   `"already-enrolled"`, `"already-waiting"`, or `"full"`.
    */
   join(id: StudentId): JoinResult;
}
```

Invariant

Precondition

Postcondition

Exceptional

Unit tests: checking rules automatically

- Documentation states each rule in general; a unit test checks one instance of it.
- Each clause of the specification should have at least one test.
- Each walkthrough step becomes a test: an initial state (given), an operation (when), and the expected resulting state (then).
- Preconditions are not tested since behaviour is unspecified when violated.

Summary: Modelling & Specification

- Models determine what is specified; specifications determine what is tested.
- Modelling is iterative: extract, shape, evaluate, revise.
- Ownership and encapsulation determine structure and where rules are enforced.
- Invariants constrain classes; preconditions, postconditions, and exceptional cases constrain operations.
- Walkthroughs evaluate the model; unit tests make the specification executable.