Common software lifecycle

- **Requirements**: identify precisely what user needs from product?
- **Planning**: identify what needs to happen and when
- **Design**: based on user requirements, evaluate and choose between possible approaches to creating a solution
- **Implementation**: write actual code matching the design
- **Testing**: determine if the implemented software actually fulfills the user requirements
- **Deployment**: get the product to the user, put in operation
- **Maintenance**: fix and refine product as necessary for the rest of its operational lifespan

CSCI 161 focus

- In this course we focus largely on the three middle stages: design, implementation, and testing
- The requirements and planning portions for 161 are largely dictated by the instructor, and no real deployment or maintenance carried out

Design

- requirements, deadlines, some design details usually dictated by instructor, so need to consider possible solution approaches:
  - identify decomposition of problem into core subproblems
    - define modules or subsystems for each subproblem, and precise responsibilities for each: what operations/types do they provide that can be called/used by other modules/subsytems
  - based on the information we need to store/manipulate:
    - pick which data structure(s) we should use (e.g. linked lists, trees, stacks, queues, lookup tables, etc)
  - for the key operations we need to perform:
    - pick which algorithms we should (e.g. for insertions, searches, sorting, removals, traversals, etc)

Design/specifications

- Document and formalize each aspect of the design, creating specifications
  - like requirements, but at the design level
  - needs to be written/checked to ensure everyone understands
- Can take weeks/months for larger/team projects
- Designs are rarely perfect on first draft:
  - issues encountered during implementation/testing may force us to revisit and update the designs

Specifications in 161

- won't require specific/separate design documents in 161
  - generally we're still developing small products, written solo, and with no long term maintenance plans
  - the data structures and algorithms largely dictated by instructor
- *will* expect that any other crucial design decisions are noted/explained in the code comments

Implementation

- implementation is the phase in which we write our actual C++ code based on the supplied design
  - identify which components belong in which files
  - implement all the data types and operations
- if design flaws are detected then we'll need to revise our design
- if errors are found in subsequent testing then we'll need to revise our implementation

Testing

- based on the user requirements (not the design), develop sets of test cases that can check if every requirement has been fully met
- each test case should have
  - a specific purpose: one precise requirement that it checks
  - the user input (data entered and/or input files) to be used
  - notes on the expected behaviour/output for the test case
- ideally, store test case input in files, automate with small scripts
  - avoids possibility of tester making mistakes when manually running tests

Test case development

- Want to check it works correctly in “normal” use
- Also want to check it correctly deals with any error checking as specified
- Nearly every sentence of requirements should give ideas for more test cases
- Example “should handle values in range 0-59”:
  - test cases for just in bounds (0, 59)
  - test cases for just out of bounds (-1, 60)
  - at least one test case somewhere inside the range (1-58)