Software Design

Prof. Clarkson
Fall 2016

Today's music: Top Down by Fifth Harmony
Review

Previously in 3110:
• architecture of large programs

Today:
• design of large programs
Review

• Architecture *is* a kind of design
  – focuses on highest level structure of system
  – based on principle of divide and conquer
• But architecture isn't about code per se
• As the *design process* proceeds, we get closer and closer to code
  – Design which modules will be part of system
  – Design the external specifications for those modules
  – Design the internal implementation of each modules
    • Which might involve iterating all the above for submodules
Review

Design criteria:

• **Simplicity:** easily understood
• **Efficiency:** uses minimal resources
• **Completeness:** solves the entire problem
• **Traceability:** every aspect of design is motivated by some requirement

...not independent

...simplicity by default trumps everything else
HIGH-LEVEL DESIGN
High-level design

• aka system design

• Goal is to decide what modules are needed, their specifications, how they interact

• Artifacts produced:
  • interfaces
  • design document
Interfaces

• In OCaml, could be .mli files
• Name of modules
• Definitions of exposed types
• Names of functions
• Specifications of functions
  – precondition, postcondition, exceptions raised, etc.
  – [next lecture]
Design document

• Capture what decisions designers made and why
• Help another programmer understand the design
• Describe important features that might not jump out from .mli files
• Mention tempting designs that were rejected and why they're problematic so that others don't make the same mistakes later
Design strategies

• **Top down:** move from abstract to concrete
  – "I know I need a module for processing inputs"
  – "What are the pieces of processing an input?"

• **Bottom up:** move from concrete to abstract
  – "I know have a module for parsing strings with regular expressions"
  – "How could I use that to process my inputs?"

• Nearly always combined
  – Design new modules from top down
  – Build on existing libraries bottom up
Top down design

• Start at top, most abstract level of hierarchy
• Proceed downwards, adding more detail to design as you deepen: stepwise refinement
• Eventually reach concrete enough design that it can be implemented

• Advantages of top down design:
  – get high-level design right
  – easier to design abstractions

• Disadvantages of top down implementation:
  – harder to test until program is complete
Bottom up design

• Start at bottom, most concrete level
• Proceed upwards, creating layers of abstraction
• Eventually reach powerful enough modules that they implement the desired system

• Advantages of bottom up implementation:
  – get low-level implementation right
  – always have testable code

• Disadvantages of bottom up design:
  – large-scale design flaws don't show up until too late
EVALUATING A DESIGN
1. What makes a design modular?

• **Partitioning:** modules are separated

• **Abstraction:** modules hide internal details

• Partitioning + abstraction yields...
  
  – *separation of concerns:* implement, maintain, reuse modules independently

  – changes to internals of one module don't require changes to other modules (even recompilation)
Partitioning

• Instance of **divide and conquer**: divide problem into smaller pieces, so that each piece can be solved separately

• Partitioning in software design is typically **hierarchical**: understanding can be deepened as necessary
  – at high level, code unit is library (at most a couple dozen modules)
  – at middle level, code unit is module (maybe a couple dozen functions)
  – at bottom level, code unit is function (maybe a couple dozen LoC)
Abstraction

• Interfaces describe the external behavior of a module, not the internal details that produce the behavior
• Design of one module can proceed with only abstractions of other modules
• Later, design proceeds from external behavior to internal details
• Abstraction enables:
  – Forgetting information
  – Treating different things as though they were the same
  – Example: animal kingdom...
Abstraction of the Camel

- **Domain**: Eukarya, Eubacteria, Archaea
- **Kingdom**: Animalia, Plantae, Fungi
- **Phylum**: Annelida, Arthropoda, Chordata
- **Class**: Aves, Mammalia, Reptilia
- **Order**: Artiodactyla, Carnivora, Primates
- **Family**: Camelidae, Giraffidae, Hippopotamidae
- **Genus**: Camelus, Loma, Vicugna
- **Species**: bactrianus, dromedarius
Computational Thinking

- Computational thinking is using abstraction and decomposition when... designing a large, complex system.
- Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction.

Jeanette Wing
Corporate VP,
Microsoft Research

https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf
2. What makes a design modular?

**Coupling:** strength of relationship between modules

- *highly coupled* modules have strong relationships with other modules
  - maybe they aren't strongly partitioned
  - maybe they share details about one another's internals hence aren't strongly abstracted

- *loosely coupled* modules have weak relationships with other modules [good modularity]
To reduce coupling...

• Keep external interfaces narrow:
  – hide representation types
  – hide helper functions
  – keep the number of functions small

• Keep external interfaces simple:
  – keep functions arguments few and their types small
  – don't let return values contain too much or too little information

• Pass data through interfaces but not control:
  – Passing control means telling the module what to do or how it should behave in the future
  – Passing data means just providing inputs that will be transformed into outputs
Coupling results from dependence

- A module *depends on* another if it uses a value, function, or type from it
- Module dependency diagram (MDD) depicts that relationship
Dependence

- **Fan out** of $M$: number of modules $M$ depends on
- **Fan in** of $M$: number of modules that depend on $M$
- both increase coupling
Question

Which of these MDDs exhibits weaker coupling?

A:
Module 1
Module 2
Module 3
Module 4

B:
Module 1
Module 2
Module 3
Module 4
Question

Which of these MDDs exhibits weaker coupling?

A:  
- Module 1
- Module 2
- Module 3
- Module 4

B:  
- Module 1
- Module 2
- Module 3
- Module 4
3. What makes a design modular?

**Cohesion:** strength of relationship within module

- *loosely cohesive* modules have weak relationships within module
  - maybe it tries to implement two unrelated pieces of functionality
  - maybe it's just a collection of utility functions

- *highly cohesive* modules have strong relationships within module [good modularity]
To increase cohesion...

- Reduce coupling
  - Strong coupling can be a sign that code is in the wrong place
  - Redesign to move it into a more cohesive module
- Make sure all parts of interface are at least logically related
- Better yet, make sure all parts of module contribute toward performing a single purpose
- Try writing a single sentence that fully and accurately describes purpose of module
  - conjunctions, commas, and multiple verbs all suggest lower cohesion ("This module implements stacks and queues.")
  - a lack of a single specific object of a verb suggests lower cohesion ("This module performs all output.")
DESIGN REVIEW
How to assess finished design

• Design review: inspection of design by a team
  – designers
  – those who produced requirements for system
  – programmers who will implement
  – independent reviewers
• Meeting is non-judgmental: focus is on improving design not on blaming for errors
• Ideally all prepare in advance by studying design, making notes, preparing questions
• All try to come to consensus about aspects of design...
How to assess finished design

• Is the design complete?
  – System requirements met by the design?
  – Specifications provided for all modules?
  – Are external dependencies (third party libraries) identified?

• Is the design high quality?
  – Simple?
  – Modular? (partitioning, abstraction, loose coupling, high cohesion)

• Does the design support implementation and testing?
  – Will modules be implementable and testable independently?
  – Can the integration of modules be tested?
Upcoming events

• [next week] A3 released

This is designed.

THIS IS 3110
Acknowledgment

Parts of this lecture are based on this book: