ECE450 – Software Engineering II

Today: **Key Principles of Software Architecture and Design (I)**

Broad history of architecture and design

- 1960's: Structured Programming
  - Response to spaghetti code
  - "GOTO Considered Harmful", Dijkstra
    - Why?
  - It took a while to convince people that coding without GOTOs was
    - Possible
    - Desirable
- 1970's: Structured Design
  - There are good ways and bad ways to divide programs in subroutines
  - See coming KWIC example
- 1980's and 1990's: Modular Programming and Object Orientation
  - Move away from structured languages and into object-oriented programming
  - Object-oriented analysis and design as a way to make sense of the world

Broad history of architecture and design (cont)

- 2000's: Several current trends. Three examples:
  - Aspect-orientation
    - Identify "aspects" of the system, and work on them separately
    - Common example: logging
    - Still mostly experimental
  - Agent-orientation
    - Think of "agents", each trying to accomplish its particular goals
    - Allows for better design of autonomic and flexible systems
  - Web Services architectures
    - REST
    - AJAX

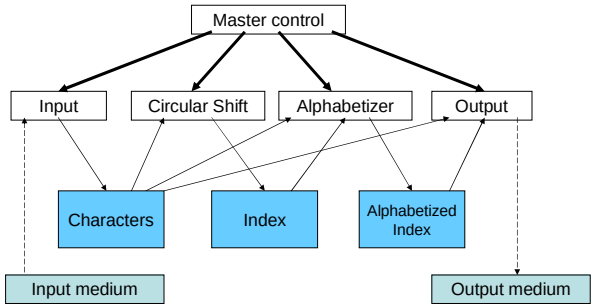
Module Structure

- Classic paper:
  - "On the Criteria to be Used in Decomposing Systems into Modules", by David Parnas
    - Discusses modularization (where "module" = collection of subroutines and data elements)
    - Critiques design by flowchart
    - Hints towards object-oriented approaches to design
    - Key lessons still overlooked by most of us!
- KWIC = Key word in context
  - Input is a set of lines of text
    - Software Engineering II
    - Mary had a little lamb
  - Output:
    - Engineering II Software
    - II Software Engineering
    - Mary had a little lamb
    - Software Engineering II
    - a little lamb Mary had
    - had a little lamb Mary
    - lamb Mary had a little
    - little lamb Mary had a

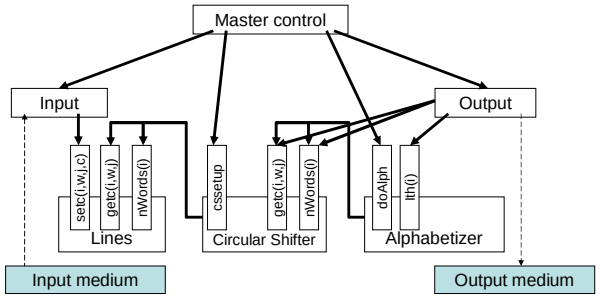
Define its modules!

- How would you modularize it?
- Again:
  - Input is a set of lines of text
    - Software Engineering II
    - Mary had a little lamb
  - Output:
    - Engineering II Software
    - II Software Engineering
    - Mary had a little lamb
    - Software Engineering II
    - a little lamb Mary had
    - had a little lamb Mary
    - lamb Mary had a little
    - little lamb Mary had a

KWIC Modularization 1



KWIC Modularization 2



Analysis of the modularizations

- Modularization 1
  - Based on the sequence of steps of the program
    - Each major step in the processing was a module
- Modularization 2
  - Based on the principle of information hiding
  - Each module has one or more “secrets”
    - Lines: How characters and lines are stored
    - Circular shifter: Algorithm for shifting and storage of shifts
    - Alphabetizer: Algorithm for alphabetization, when to alphabetize
- Note how both systems might share the same data structures and algorithms
  - They might be identical in their executable representations!
- The difference is in how to divide them in work assignments
  - There are many representations other than the executable one:
    - For documentation
    - For understanding
    - For modifying
    - …

Comparison of changeability

- What design decisions might change? How do they affect each modularization?
  - Format of the input
    - Affects one module in each
  - All lines stored in memory?
    - Affects ALL modules in Mod 1, one module in Mod 2!
  - Pack characters 4 to a word
    - Affects ALL modules in Mod 1, one module in Mod 2!
  - Make an index for circular shifts, rather than store them
    - Affects three modules in Mod 1, one module in Mod 2
  - Alphabetize once, rather than searching for items as needed (or partially alphabetize)
    - Affects three modules in Mod 1, one module in Mod 2

Other comparisons

- Independent development?
  - Modularization 1
    - Must design all data structures before parallel work can proceed
    - Complex decisions if you're new to the problem domain
  - Modularization 2
    - Must design interfaces before parallel work can begin
    - …which need only be simple descriptions
- Efficiency?
  - Modularization 1 might actually be more efficient
    - Less method calls
    - Depends on the decisions at each module in Modularization 2
- Comprehensibility?
  - According to Parnas, Modularization 2 is better
    - You may have a different judgment
    - …but remember that each module only needs to "know" very little about the others