A Sketchy Evolution of Software Design

- 1960s
  - Structured Programming
    - ("Goto Considered Harmful", E.W.Dijkstra)
    - Emerged from considerations of formally specifying the semantics of programming languages, and proving programs satisfy a predicate.
    - Adopted into programming languages because it's a better way to think about programming
- 1970s
  - Structured Design
    - Methodology/guidelines for dividing programs into subroutines.
- 1980s
  - Modular (object-based) programming
    - Ada, Modula, Euclid, …
    - Grouping of sub-routines into modules with data.
- 1990s
  - Object-Oriented Languages started being commonly used
  - Object-Oriented Analysis and Design for guidance.

Three Papers by David Parnas

- "On the Criteria To Be Used in Decomposing Systems into Modules"
  - *Comm. ACM 15, 12* (Dec. 1972), 1053-1058
- "On a 'Buzzword': Hierarchical Structure"
  - *IFIP Congress '74.*
    North Holland Publishing Company, 1974 pp. 336-339
- "On the design and development of program families"
  - *IEEE Trans. On SE.,* vol. SE-2, pp.1-9, Mar. 1976

Module Structure

- David Parnas
  - "On the Criteria To Be Used in Decomposing Systems into Modules"
    - *Comm. ACM 15, 12* (Dec. 1972), 1053-1058
- Discusses "modularization"
  - Module = a collection of subroutines and data elements
  - Critique of Procedural Design
    - Pointing the way to object-based and OO design.
- Describes two ways to modularize a program that generates KWIC (Key Word in Context) indices.
  - Modularization 1
    - Based on the sequence of steps to perform
  - Modularization 2
    - Based on the principle of "information hiding"

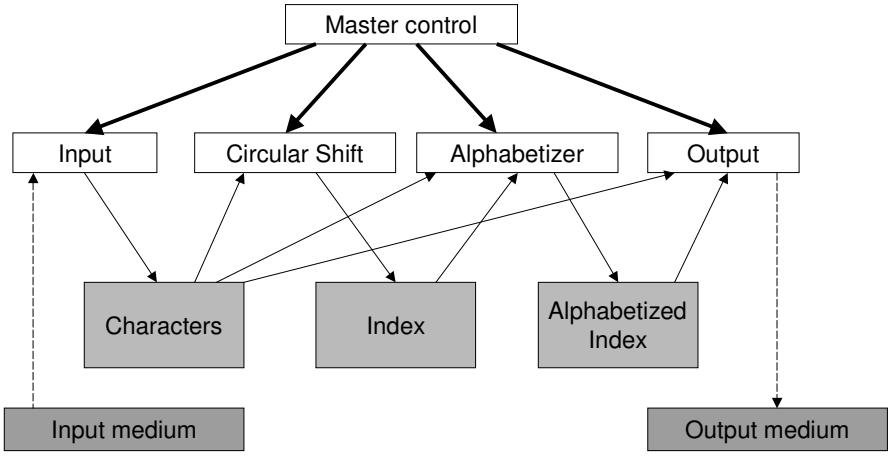
KWIC

- Input

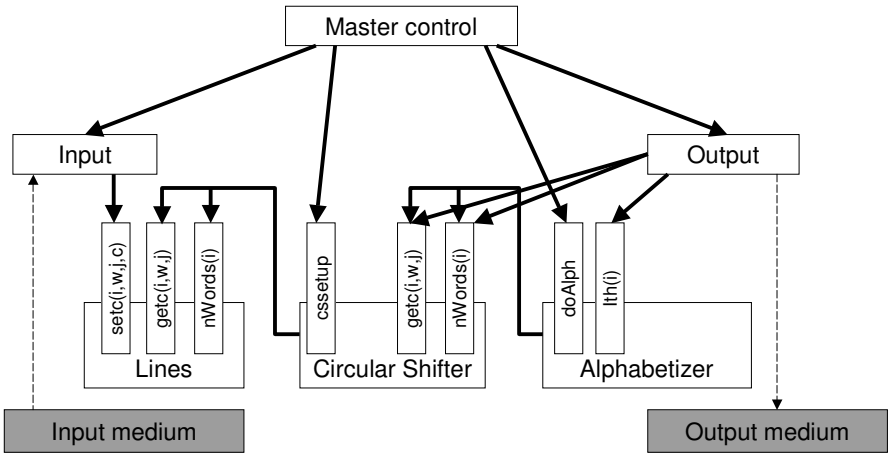
  Designing Software for Ease of Construction
  Figs are Good
- Output

  are Good Figs
  for Ease of Construction Designing Software
  of Construction Designing Software for Ease
  Construction Designing Software for Ease of
  Designing Software for Ease of Construction
  Ease of Construction Designing Software for
  Figs are Good
  Good Figs are
  Software for Ease of Construction Designing

KWIC Modularization 1



KWIC Modularization 2



Criteria for decomposition

- Modularization 1
  - Each major step in the processing was a module
- Modularization 2
  - Information hiding
    - Each module has one or more "secrets”
    - Each module is characterized by its knowledge of design decisions which it hides from all others.
  - Lines
    - how characters/lines are stored
  - Circular Shifter
    - algorithm for shifting, storage for shifts
  - Alphabetizer
    - algorithm for alpha, laziness of alpha

General Comparison

- General
  - Note: both systems might share the same data structures and the same algorithms
  - Differences are in the way they are divided into work assignments
  - Systems are substantially different even if identical in the runnable representation
    - Possible because the runnable representation is used only for running
    - Other representations are used for
      - Changing
      - Documenting
      - Understanding
      - …

Changeability Comparison

- Design decisions that may change
  - Input format
    - (1, 1)
  - All lines stored in memory
    - (all, 1)
  - Pack characters 4 to a word
    - (all, 1)
  - Make an index for circular shifts rather than store them
    - (3,1)
  - Alphabetize once, rather than either
      - Search for each item as needed
      - Partially alphabetize, partially search
    - (3,1)

Independent Development

- Modularization 1
  - Must design all data structures before parallel work can proceed
  - Complex descriptions needed
- Modularization 2
  - Must design interfaces before parallel work can begin
  - Simple descriptions only
- Comprehensibility
  - Modularization 2 is better
    - Parnas subjective judgment

Hierarchical Structure

- David Parnas
  - "On a 'Buzzword': Hierarchical Structure"
    - *IFIP Congress '74.*
      North Holland Publishing Company, 1974 pp. 336-339
- Earliest abstract discussion of what has become known as "software architecture"
  - Debunks prevalent notion **Hierarchical == good**
  - Seeks to demonstrate that the term is used for many different things, often at the same time.
  - Context is OS design
    - T.H.E, MULTICS, RC4000

Types of Hierarchy

- Program Hierarchy
  - Calls hierarchy
  - Only useful when humans are working with the system
    - E.g., inlines as semantically equivalent to calls
- Process Hierarchy
  - Often mixed-up with other hierarchies
  - E.g., "Give works to" in the T.H.E system.
- Resource Allocation Hierarchy
  - RC4000
- Protection Hierarchy
  - MULTICS
- Module Hierarchy
  - Procedure or module **part-of** a higher-level module

Program Families

- David Parnas
  - "On the design and development of program families"
    - *IEEE Trans. On SE.,* vol. SE-2, pp.1-9, Mar. 1976
- Family of Programs
  - When it is worthwhile to
    - First study common properties
    - Then determine special properties of each family member
- Basis for
  - Designing for change
  - Reuse libraries
  - OO Design

Methods

- Classical method: Sequential completion
  - One member of the family developed completely
  - Modify to get the next
  - And so on
- New techniques
  - Develop to an intermediate stage
  - Different family members will proceed with different design decisions from the intermediate stage onwards
  - Must represent the intermediate stages:
    - Stepwise refinement based
      - Postpone implementation of operand types and operators until later.
    - Module based
      - Specify module interface
      - Substitute different implementations