Outline: Software Design

- Goals
- History of software design ideas
- Design principles
- Design methods

  Life belt or leg iron? (Budgen)

A Little History ...

- At first, struggling with programming languages, small programs, math algorithms.
  - Worried about giving instructions to machine (efficiency)
  - "Think like a computer"
- Found that life cycle costs depend far more on how well communicates with people than how fast it runs.
- Separated the two and more emphasis began on
  - How to write software to communicate algorithms and structure to humans
  - How to structure design process itself.

Structured Programming

- Goal: mastering complexity

- Dijkstra, Hoare, Wirth:
  - Construction of correct programs requires that programs be intellectually manageable
  - Key to intellectual manageability is the structure of the program itself.
  - Disciplined use of a few program building blocks facilitates correctness arguments.

Structured Programming (2)

- Restricted control structures
- Levels of abstraction
- Stepwise refinement
- Program families
- Abstract data types
- System structure:
  - Programming-in-the-large vs. programming-in-the-small
  - Modularization
  - Minimizing connectivity

Restricting Control Structures

- Dijkstra: 3 main mental tools
  - Enumerative reasoning
  - Mathematical induction
  - Abstraction (e.g., variable, procedure, data type)

1. Restrict programs to constructs that allow us to use these mental aids.
   - Sequencing and alternation (enumeration)
   - Iteration and recursion (induction)
   - Procedures, macros, and programmer-defined data types
   - SESX
   - Small procedures

2. Make program structure fit problem structure.

Levels of Abstraction

- 1968: Dijkstra paper on his experiences with T.H.E. Multiprograming system

- Designed using "levels of abstraction"
  - System design described in layers
  - Higher levels could use services of lower levels
  - Lower levels could not access higher levels

- Lowest level implemented first
  - Provided a "virtual machine" for implementation of next level
  - Process continued until highest level completed.

- A "bottom up" technique

Stepwise Refinement

Wirth (1971): "Divide and conquer"

- A top-down technique for decomposing a system from preliminary design specification of functionality into more elementary levels.
- Program construction consists of sequence of refinement steps.
- Use a notation natural to problem as long as possible.
- Refine function and data in parallel.
- Each refinement step implies design decisions. Should be made explicit.

Prime Number Program

```
begin var table p;
    fill table p with first 1000 prime numbers
    print table p
end
```

- Assumes type "table" and two operators
- Design decisions made:
  - All primes developed before any printed
  - Always want first 1000 primes
- Decisions not made:
  - Representation of table
  - Method of calculating primes
  - Print format

Program Families

- Basic premise: Software will inevitably exist in many versions
  - Different services for slightly different markets
  - Different hardware or software platforms
  - Different resource tradeoffs (speed vs. space)
  - Different external events and devices
  - Bug fixes
- Think of development as a tree rather than a line
  - Never modify a completed program
  - Always begin with one of intermediate forms
  - Continue from that point making design decisions
- Order of decisions important in how far have to back up.
  - Make early decisions only those that can be shared by all family members
  - Put off decisions as long as possible.

Abstract Data Types

- Defines a class of objects completely characterized by operations available on those objects.
- Really just programmer-defined data type
  - Built-in types work same way
  - Allows extending the type system
  - Pascal, Clu, Alphard, Ada
- Want language to protect from foolish uses of types (strong typing or automatic type conversion)
- Criteria:
  1. Data type definition must include definitions of all operations applicable to objects of the type.
  2. User of ADT need not know how objects of type are represented in storage
  3. User of ADT may manipulate objects only through defined operations and not by direct manipulation of storage representation.

System Structure

- DeRemer and Kron (1976):
  - Structuring a large set of modules to form a system is an essentially distinct and different intellectual activity from that of constructing the individual modules (programming in the large, MILs)
  - Activity of producing detailed designs and implementations is programming in the small.
- Modularization
  - Want to minimize, order, and make explicit the connections between modules.
  - Combining modularity with hierarchical abstraction turned out to be a very powerful combination (part-whole and refinement abstractions)

Module Specification

- Started to distinguish between design and "packaging"
  - Design is process of partitioning a problem and its solution into significant pieces.
  - Packaging is process of clustering pieces of a problem solution into computer load modules that run within system time and space constraints without unduly compromising integrity of original design.
  - Optimization should only be considered in packaging and care should be taken to preserve design structure.
- Reuse
  - Assumed hundreds of reusable building-block modules could be abstracted and added to program libraries.
    Why didn't happen?

Stepwise Refinement vs. Module Specification

SR: Intermediate steps are programs that are complete except for implementation of certain operators and operands.

MS: Intermediate stages are not programs. Instead they are specifications of externally visible collective behavior of program groups called modules.

- Similarities
  - Precise representation of intermediate stages in program design.
  - Postponement of decisions: Important decisions postponed until late stages or confined to well-delineated subset of code.

Stepwise Refinement vs. Module Specification (2)

- Differences
  - Decision Making

    SR: Decision-making order critical. May have to backtrack more than really want. Sequencing decisions made early because intermediate reps are programs.

    MS: May be easier to reverse decisions without repeating so much work. Sequencing decisions made last.

  - Effort

    SR: Less work than either classical approach (because keeps complexity in control) or MS.

    MS: Significant amount of extra effort because only works if external characteristics of each module sufficiently well specified that code can be written without looking at implementation of other modules. In return, get independent development potential.

Minimizing Connectivity

- Yourdan; Constantine and Myers
  - Cohesion: relationship between functions a module provides
  - Coupling: relationship between modules, intermodule connections
- Intermodule Friction
  - Smaller modules tend to be interfaced by "larger surfaces"
  - Replacement of module with large interface causes friction, requiring rewrites in other modules.
- Uses relationship
- Primary goal: locality of visibility

Minimizing Connectivity (2)

- Advantages of reducing connectivity (coupling)
  - Independent development (decisions made locally, do not interfere with correctness of other modules).
  - Correctness proofs easier to derive
  - Potential reusability increased.
  - Reduction in maintenance costs (less likely changes will propagate to other modules)
  - Comprehensibility (can understand module independent of environment in which used).
  - Some studies show less error-prone.