Software Design Concepts (Chapter 13)

- Starting point
  - Specifications & preliminary analysis
  - Entity, process flow & control flow diagrams
  - Any preliminary architectural, data, or interface designs.
  - Any PDL, Formal spec. (pre & post conditions), etc.

- Software design
  - Prelude to programming and coding of components
  - Provide high quality software
  - Meet the requirements and specifications (revised)
  - Address the data, function and control software

History of Software Design

- Beginning with Dykstra – simple
  - Functions and data equals program
  - Structured programming
  - Top down – decomposition
    - Didn't always work – sometime needed bottom-up

- Languages
  - Fortran – evolved and improved
  - Algol – fore runner of Pascal, etc.
  - C & Smalltalk – fore runners of object oriented systems,
    - C++ & Java
  - Others – Lisp, Perl, and many others
  - Significant improvements in debugging, error analysis, etc.

Design Principles

- Traceable to “analysis” models – specification
- Don’t reinvent the “wheel”
  - Reusable software, libraries, etc.
- Exhibit, uniformity, integration and modularity
  - Design to accommodate change
  - Design is prelude to actual coding
  - They are not usually caught in debugging or are hard to find
Design Concepts

Abstraction – partially covered previously
Modeling - covered
⇒ Mathematical models and analysis
⇒ Data and functional abstraction
⇒ Object abstraction
⇒ Base & derived object

Refinement - covered
⇒ Top down strategies – design upper level components first
⇒ Complements abstraction

Modularity - new
⇒ Isolation of activity
⇒ Minimize interfaces – keep related things together; unrelated things apart

Modularity

Size not important - what is?

⇒ Decomposability & Composability
⇒ Partition into subproblems
⇒ Group related activity into related “sub” problem

⇒ Continuity
⇒ Changes map to changes in a module(s) instead of across system
⇒ Reduce side effects

⇒ Protection
⇒ Ability to confine information and hide it if necessary

Effective Modular Designs

⇒ Functional Independence
⇒ Single-minded with minimal interactions

⇒ Cohesion
⇒ Module does a “single” task - procedural cohesion
⇒ Module uses specific data block(s) - communicational cohesion

⇒ Coupling
⇒ Passage of control between modules
Heuristics for Effecting Modularity

- Evaluate program module activity
  - Group (implode) & separate (explode) modules
  - Examine for cohesion & coupling
- Evaluate program "fan-out"
  - If "high" fan-out determine if activity can be reduced
  - Keep scope of control confined where possible
- Evaluate module entry and exit
  - Make modules predictable - black box
  - Avoid modules which use internal "memory"

Architectural Design (Chapter 14)

Data Architecture Design Heuristics

- Define all data structures and activity
- Data dictionary
  - Naming, attributes, keys, relationships, etc.
- Data access - to components which need it, i.e., hiding
- Data libraries
  - Group data access and actions
  - Use available database systems (Oracle, FoxPro, etc.) and SQL type languages
Data Architectural Styles (cont’d)

- Data flow
  - Call and return - main and subprogram structure
  - Object-oriented structure - data and operations integrated
    - Library structure - dll’s, etc.
  - Layered structure
    - Outer layer - user interface
    - Inner layer - data operations
    - Intermediate layer - utility services and application software interface

Transform Mapping Steps

- Review system model
- Refine data flow software diagrams
- Determine whether DFD has transform or transaction character
- Isolate transform centers - data I/O
- Factor into top-down and sublevels of control
- Study sections 14.6 - 14.8 for examples