Lecture 6.2
Design of processes: Structured Design

**Physical process design**
- The data flow diagrams and process descriptions produced in the systems analysis phase need to be translated into processes (programs and procedures) during the systems design stage.
- These need to be designed with quality criteria in mind: maintainability etc.
- Programmers may be more concerned about whether the code works or is elegant.

**Structured Design**
- A lot of software resources are spent on maintenance
- To help avoid this we need to be able to:
  - deliver error free designs
  - design easy to maintain systems
  - design software systems that are easy to test and validate

**Structured Design**
- Partition the system into "black boxes"
- Organise the "black boxes" into hierarchies suitable for computer implementation
- Use diagrams to make the system structure easy to understand
- Use a set of strategies for developing a design solution from a well defined statement of a problem (e.g. a DFD)
- Use a set of criteria for evaluating the quality of a given design solution
- Produce systems that are easy to understand, reliable, flexible, and efficient to maintain

**Structure Charts**
A system is easier to write and test if we divide it into modules

- Each of these modules is coded separately
- GET VALID TRANSACTION

**Structure charts**
- USE A HIERARCHY:
- Start and end here
- Top members (managers) co-ordinate and control
- Subordinates process
- A manager should have no more than 7±2 immediate subordinates reporting directly to it
Structure Charts

- manager calls worker
- worker executes
- manager resumes

- each module does something for its boss
- it may appoint subordinates

Module connections

A reference to the name of another module
A calls B

A reference to an identifier defined within another module.
A is making reference to a data item X that exists inside B. The reference is from A to B, but the data flows from B to A.

Modules for system building

A algorithm

- Sends info. from below up to its bosses
- Sends info. down to its subordinates

- Gets info. from boss, transforms it and sends new info. back to the boss
- Co-ordinates the communication of its subordinates

Deriving structure charts

- Each business process will generate its own structure chart, using a design strategy such as Transform Analysis.
- Each box on the structure chart will be a module with its own specification.
- Tie together all structure charts under a transaction monitor, so each structure chart can be 'fired' each time a transaction stimulus of its type arrives (or is selected from a menu).

Transform analysis

A design process which turns data flow diagrams into structure charts.

A transform-centred system has as its focus the derivation of new information from existing data

The input part of the system is known as the afferent part
The output part of the system is known as the efferent part

Transform analysis - an example
Design features

- Design features that lead to systems that are easier to maintain and modify:
  - Small module size
  - Modular independence (decrease coupling)
  - Black box characteristics
  - Module strength (increase cohesion)

Design guidelines

Design guidelines that lead to systems that are easier to maintain and modify:

- Factoring
  Decompose a system into smaller and smaller parts, organised hierarchically
- Span of Control
  No superordinate should control more than 7 subordinate modules
- Coupling
  Minimise the extent to which modules are dependent on each other (thereby minimising communication between modules)

Design guidelines cont.

- Reasonable Size
  Keep modules reasonably small
- Cohesion
  All activities within a module should pertain to the same process
- Shared Use
  Subordinate modules should be called by several superordinate modules where possible

Coupling and cohesion

- These are important design principles related to the concept of “black box” design
- Black Box Design
  - Defined outputs
  - Defined function
  - Unknown internal processing
  - (How?)
  We should be able to exploit a “black box” without knowing what’s inside it

Cohesion

Measure of the strength of association of elements within a module

- Strongest, best maintainability
  - Functional
  - Sequential
  - Communicational
  - Procedural
  - Temporal
  - Logical
  - Coincidental
  - Black box
  - Not quite so black box
  - Grey box
  - White box

Weakest, worst maintainability
Levels of cohesion

- **Functional**: the highest level of cohesion in which all instructions in the module pertain to a single function or task.
- **Sequential**: instructions are related by data rather than by the task being performed (i.e., the output of one instruction becomes the input to the next instruction).
- **Communicational**: each instruction in the module acts on the same data but without any sequential dependence.

Levels of cohesion

- **Procedural**: instructions are related through flow of control, and sequence is still important.
- **Temporal**: instructions are related through flow of control, although sequence is not important.
- **Logical**: instructions hardly related at all; execution is determined from outside the module.
- **Coincidental**: the lowest level, where instructions have no relationship to each other at all.

Levels of cohesion

- **Procedural**: instructions are related through flow of control, and sequence is still important.
- **Temporal**: instructions are related through flow of control, although sequence is not important.
- **Logical**: instructions hardly related at all; execution is determined from outside the module.
- **Coincidental**: the lowest level, where instructions have no relationship to each other at all.

Coupling

- **The data travelling between modules is called coupling**
- **Data**:
  - Highest form: modules are independent of each other and communicate through passing data elements.
  - Stamp: modules communicate through data records rather than data elements; more complex.
- **Control**:
  - Modules communicate through control flags.
- **Common**:
  - Modules communicate through common, shared data.
- **Content**:
  - Lowest form: modules refer to the inner workings of other modules.

The more we must know of module B to understand module A, the more closely coupled A is to B.

We are aiming for loosely coupled systems.

Types of coupling

- **Good, or loose**
  - Data: highest form: modules are independent of each other and communicate through passing data elements.
  - Stamp: modules communicate through data records rather than data elements; more complex.
  - Control: modules communicate through control flags.
  - Common: modules communicate through common, shared data.
  - Content: lowest form: modules refer to the inner workings of other modules.

- **Bad, or tight**
  - Data: highest form: modules are independent of each other and communicate through passing data elements.
  - Stamp: modules communicate through data records rather than data elements; more complex.
  - Control: modules communicate through control flags.
  - Common: modules communicate through common, shared data.
  - Content: lowest form: modules refer to the inner workings of other modules.
**Coupling**

Diagnosis Code

- AVOID WIRED-IN VALUES
- FORMAT ERROR MESSAGE
- CUSTOMER ERROR

ALL THE DATA MUST BE HONESTLY SHOWN

**Flexibility and efficiency**

- The more efficient a system, the harder it may be to modify.
- A system designed for flexibility may be slower in running time, and may take up more space.
- May have to compromise our modules in practice because of performance

**References**