

InfSci 2510
Information System Analysis and Design

Lecture 9
11/4/2002

Reading

- *Chapters 20*
Quality Assurance through Software Engineering
- *Chapters 8*
Prototyping and Rapid Application Development

Topics

- **ISAD in System Development Life Cycle**
- Information System Development
- Structure Charts and Architecture Patterns
- Software Engineering in Modular Development
- Prototyping

System Development Life Cycle

- Identify: problems or opportunities
- Determine: the requirements
- **Analyze: the existing system**
- **Design: a solution – the intended system**
- ***Develop*: the new system**
- ***Test*: functional correctness and performance**
- ***Implement*: deploy the system**
- Maintenance/Evaluation: fixes/enhancements

Topics

- ISAD in System Development Life Cycle
- **Information System Development**
- Structure Charts and Architecture Patterns
- Software Engineering in Modular Development
- Prototyping

Information System Development

How to maintain quality in system development?

TQM - Total Quality Management

The primary elements of TQM are meaningful only when occurring in an organizational context that supports a comprehensive quality effort.

-Dean and Evans (1994)

- Early commitment to quality
- Organizational support (from the top, and the team...)
- Disciplined structured approach

The analyst/designer must be aware of the factors driving the interests in quality.

Information System Development

Structured Approach to ISAD:

- **Top-Down Design**

...maintaining the big picture so that design work of each component will not lose sight of the system goal.

- **Modular Development**

...breaking down the programming task into logical, manageable modules, with strong emphasis on the interface between modules.

- **Bottom-Up Integration**

...beginning with the bottom level, integrate modules after unit testing and move up the functional hierarchy, so that system integration works from the bottom up.

Information System Development

- **Top-Down Design:** identifying the overall goal first and decomposing into sub-goals; develop smaller parts and sub-systems to solve sub-goals which will then integrate into the overall system.
 - overall organizational objectives maintained throughout the development effort; compatible with the DFD analysis and design approach.
 - modular development and bottom-up integration can then afford parallel effort on different parts.

Top-Down Approach to IS Development

Organizational Objectives Level

Coordinating the system to meet company objectives

Functional Systems Level

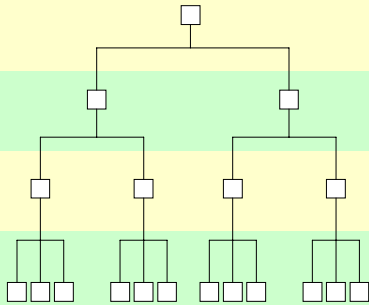
such as: payroll, accounting, customer care and services

Operational System Level

such as: databases, file access, user administration...

Program Module Level

such as: input data capture, sorting records, printing...



Information System Development

Modular Development

- Concurrent effort.
- For the processes to be automated, partition the overall system into sub-systems.
- Develop **structure chart** for each sub-system.
- Apply top-down design and maintain the hierarchical relationship between modules.
- Keep each module to a manageable size, but minimize the number of modules that would be affected when making changes.
- Pay special attention to the **critical interfaces** (note the control flags and data couples passed in-between).

Information System Development

Bottom-Up Integration

- **Unit test** each module, using *stubs* and *drivers*.
- Related modules are tested together – in order to **test the interfaces** between these modules.
- When modules are integrated into a component, **test the integrated sub-system**.
- Testing should be applied to each integration, **from bottom up the hierarchy**.

Information System Development

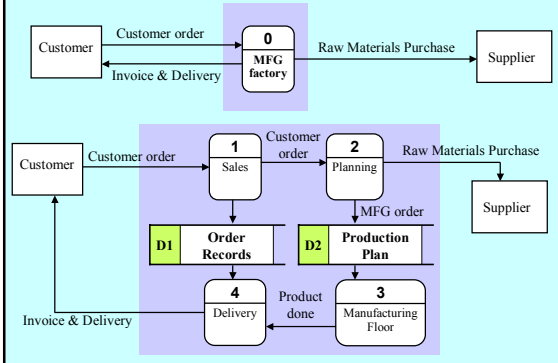
Pitfalls of the top-down design approach

- Wrong sub-systems: mistake in hierarchical functional decomposition – watch out for overlapping needs and sharing of components.
- Neglected interface design: it usually requires **expertise** to see the ramifications in the interfaces.
- Impact of changes: it may be difficult to track how changes in one component may affect the others.
- Waterfall model in effect: the lack of **user involvement** during the development time.

Topics

- ISAD in System Development Life Cycle
- Information System Development
- **Structure Charts and Architecture Patterns**
- Software Engineering in Modular Development
- Prototyping

Physical DFD Example...



Structure Charts

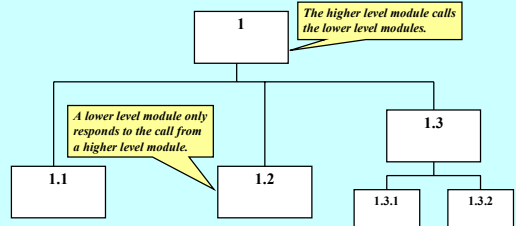
Applying physical DFD to software design and development...

- A software sub-system is identified by a certain process in the DFD which is to be automated.
- The processes derived from it should also be automated, being lower in the hierarchy.
- The **structure chart captures this hierarchy** to facilitate top-down design of all the modules in the software sub-system; structure chart shows the control hierarchy of program modules.
- There should be a structure chart for each software sub-system.

Structure Chart

Recommended tool for modular, top-down system design.

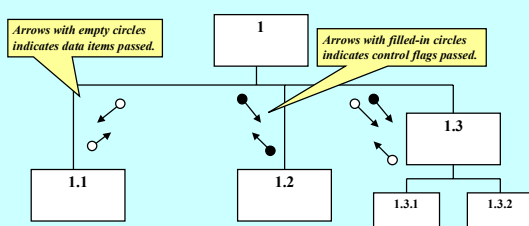
- A **tree** of rectangles ... (an *acyclic* graph)
- Each rectangle generally is a process, except perhaps the lower level ones.



Structure Chart

Design the interfaces between these modules...

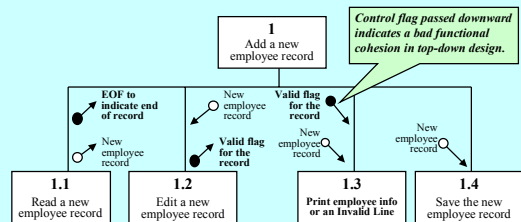
- **Data Couples:** data items passed between them;
- **Control Flags:** control signals passed between them.



Structure Chart Example

To add a new employee record...

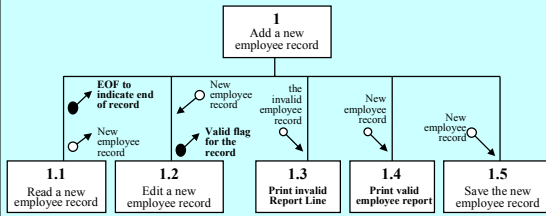
- Process 1 (Add a new employee record) has four subordinate processes: 1.1, 1.2, 1.3, 1.4.



Structure Chart Example

Do *not* pass control information downward the hierarchy.

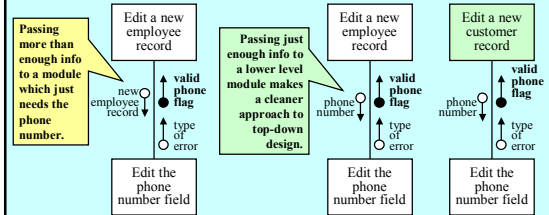
- **Improved structure chart:** using modules with well defined functions only, will NOT be passing control flag downwards.



Structure Chart Example

To edit the phone number in a new employee record...

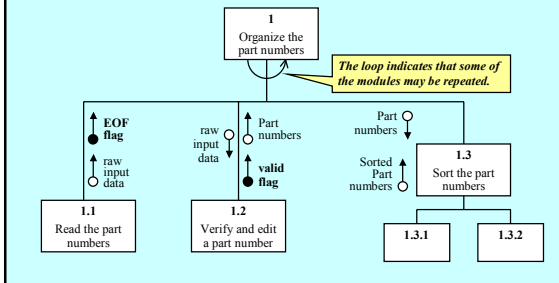
- **Do NOT pass more than enough information downward.**
- So that lower level modules remain functionally pure – a better approach to top-down design.



Structure Chart: the Loop

Depicting iterative top-down control...

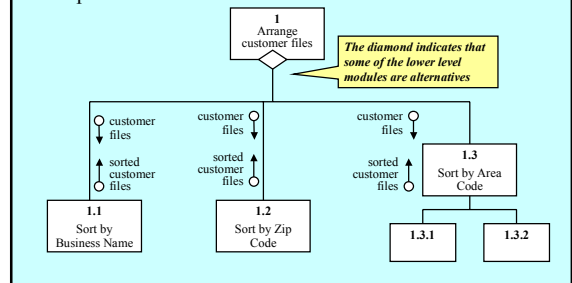
- **The Loop:** some of the modules may be repeated.



Structure Chart: the Diamond

Depicting alternatives in top-down control...

- **The Diamond:** some but not all of the modules may not be performed.



Drawing a Structure Chart

Exercising top-down control...

- Start with the **physical DFD**: with automated processes partitioned into software sub-systems.
- One Structure Chart for each sub-system.
- The **top level process** of the sub-system is the **top level module** – identifying the goal of the sub-system.
- Lower level DFDs will show the processes for the lower level modules.
- The lowest level modules may need further subordinate modules (beyond the DFDs).

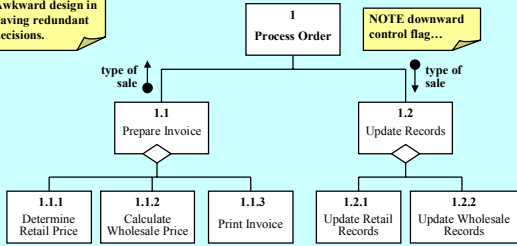
Design using a Structure Chart

Some principles to follow...

- Minimize the use of control flags – this results in cleaner interfaces between the modules. The data couples are translated from the data flows.
- Avoid using any **downward flow of control flags**.
- Prevent **redundant decisions** (probably indicated by a downward control flag somewhere).
- Make use of the **sharing of resources** by sharing the use of lower level modules.
- Note the presence of **improper subordination** and make corrections.

Prevent Redundant Decisions

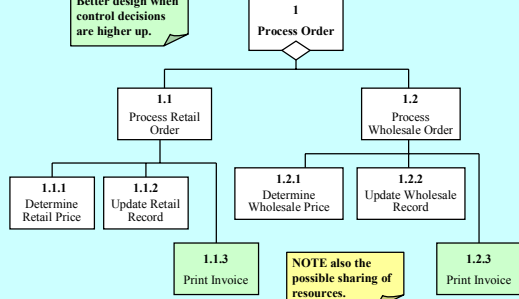
Awkward design in having redundant decisions.



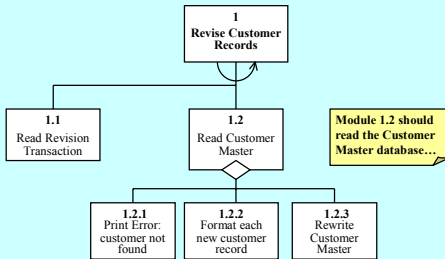
NOTE downward control flag...

Prevent Redundant Decisions

Better design when control decisions are higher up.

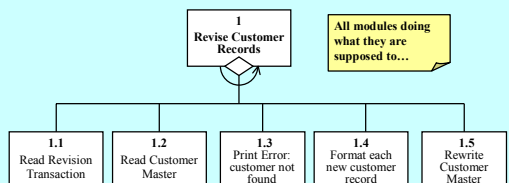


Improper Subordination



Module 1.2 should read the Customer Master database...

Improper Subordination: corrected



All modules doing what they are supposed to...

Structure Chart: types of modules

Hierarchical Approach...

- **Control Module:** found near the top, contains business logic to control lower level modules. Often represented in DFD as a process.
- **Transformational Module:** found in the mid level, performs one task noted in the business terminology (e.g. Sort the Customer files).
- **Functional Module:** aka Specialized modules, found in the lowest level, performs a task in the data processing terminology (e.g. Copy a data record). May or may not be found in the DFD.

Structure Chart: architectural patterns

- **Transform-centered:** the system is primarily for data processing – transforming data flows in the system (from one form to another); DFD showing generally a linear path.
- **Transaction-centered:** decision is made at the high level to control which data path to take; DFD showing generally multiple parallel paths.
- **Open Platforms:** general purpose system – with a collection of tools available.
 - toolkit
 - framework
 - workbench

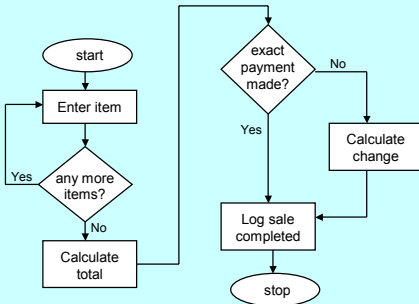
Topics

- ISAD in System Development Life Cycle
- Information System Development
- Structure Charts and Architecture Patterns
- **Software Engineering in Modular Development**
- Prototyping

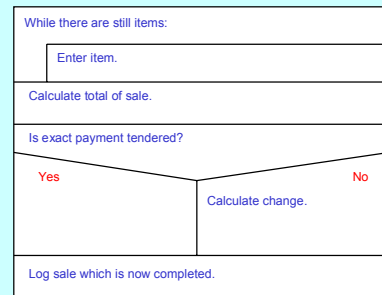
Detailed Specification for each module

- Specify the interface for each module.
 - business control logic: decision tree or decision table;
 - user interface: story boards for each dialog path;
 - batch processing: input/output formats;
- Specify what each module should do.
- Specify how each module should be implemented.
 - Flow Chart – for traditional procedural programming
 - Nassi-Schneiderman Chart – for structured programming
 - Pseudo Code – program oriented
 - Procedure Manual – human reading oriented
 - Folklore Method (informal)

Flow Chart: POST example



Nassi-Schneiderman Chart: POST example



Information System Development

Pushing for Quality in module development...

- **TQM** applied in software engineering teams:
 - Coding Standard
 - Code Inspection
 - Structured Walkthroughs
 - Ego-less Peer Review
 - Competitive Parallel Development
 - Extreme Programming

Topics

- ISAD in System Development Life Cycle
- Information System Development
- Structure Charts and Architecture Patterns
- Software Engineering in Modular Development
- **Prototyping**

Prototyping vs the SDLC

The *Waterfall* model... **one stage after another!**

- Identify: problems or opportunities
- **Determine: the requirements**
- **Analyze: the functions needed**
- **Design: a solution – the intended system**
- **Develop: the new system**
- Test: functional correctness and performance
- Implement: deploy the system
- Maintenance/Evaluation: fixes/enhancements

Prototyping vs the SDLC

The problems in SDLC...

- **Lack of end user involvement** until the end – mistakes in the process will be difficult to discover until it becomes too costly to fix.
- The time span in the long SDLC duration allow too much for **changes in user requirements** – the project is actually shooting a moving target!

Why Prototype?

Seeking information early in the SDLC...

- **User Reactions**
- **User Suggestions**
- **Test Innovations** (prototype as a test bed)
- **Revision of Plans...**

Risk Areas and Risk Control

- **Risk areas:** parts of the system which we have no experience, and/or insufficient knowledge about how it should operate.
- Prototype can be design to tackle the risk area.
- **Feasibility Test** – to assess the level of risk:
 - technical feasibility: possible to work?
 - operational feasibility: serve the business purpose?
 - economic feasibility: too costly?

Approaches to Prototyping

- **Patched-Up Prototype:** a system quickly put together – functional but not robust: need to use with caution.
- **Non-Operational Prototype:** not fully functional, but meant to test a particular layer. (Horizontal)
- **First-of-a-Series (pilot) Prototype:** intended for systems with multiple installations planned.
- **Selected Features Prototype:** functional but not covering all the features. (Vertical)

Patched-up Prototype

- System would work properly in controlled test situations.
- Most error situations may not be handled, and may result in serious problem.
- Certain less important functional features may not be available.
- System performance not guaranteed.

Horizontal Prototype

- To test the features of a specific layer of functionalities in the system.
- Example: story boards to test the Graphical User Interface layer...
- Not operational, but covers most of the features (to solicit user opinion).
- May or may not be re-used.

Vertical Prototype

- To test a certain segment of functionalities with some sub-system, and performance.
- Example: database access functionalities.
- Operational for the available functionalities, but not all the relevant features are there.
- Often can be re-used – developed in the system for use later on.

First of a series - Prototype

- To try it out before committing to develop more of the same.
- Pilot System.