

**Chapter
2**

**STRUCTURED
PROGRAM
DESIGN**

*Get on the
Fast Track!*



TM

**SYS-ED/
Computer
Education
Techniques, Inc.**

Objectives

You will learn:

- Modularity.
- Top-down design.
- Structure chart/hierarchy chart.
- HIPO diagram.
- Visual table of contents.
- Overview diagram.
- Detail diagram.
- Top-down coding.
- Top-down testing.

1 Modularity

The principles serving as the foundation of modularity are that:

- Any logical portion of a modular program can be changed without affecting the rest of the program.
- Each module, which in the COBOL programming language will be a subroutine or section, is independent of the other modules.
- Each module has one entry point and one exit point.
- One module must not be able to directly alter the code in another module.
- True modularity consists and is implemented with the three basic logic structures of structured programming.

The benefits of modularity include facilitating the:

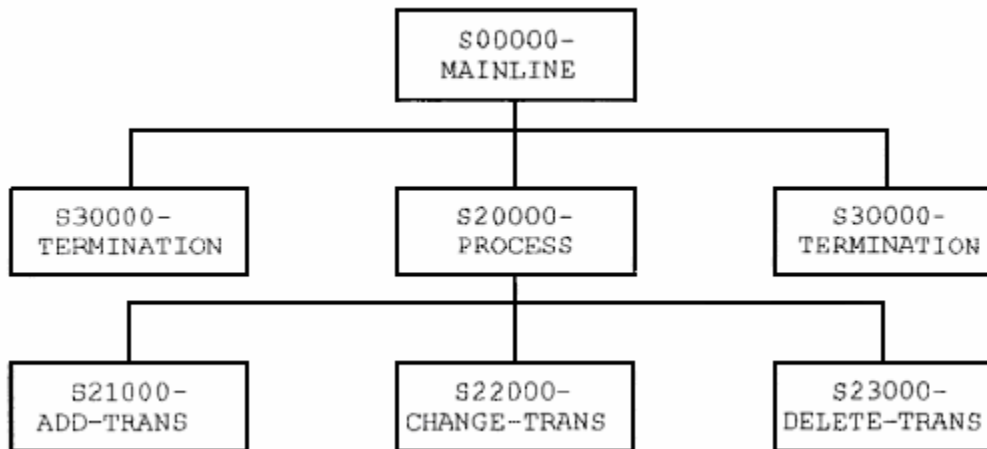
- Development of code.
- Maintenance and changing of programs.
- Testing and debugging of programs.
- Manager's task of allocating programming assignments.
- Organization and use of code for a return on investment.

2 Top-down Design

Top-down design starts with the highest levels of logic and major functions, and then incorporates the details and lesser functions.

The benefits of top-down design include:

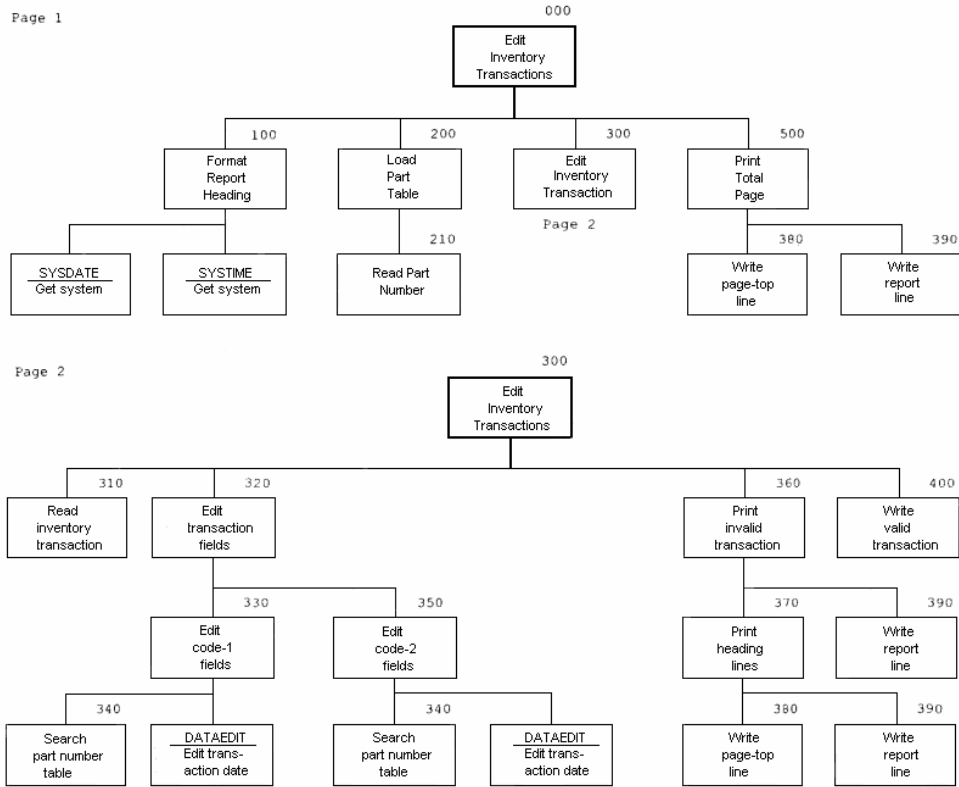
- A reduction in design complexity.
- Providing for an orderly development of logic.
- Avoiding simultaneous inconsistent interface definitions by different programmers working on the same interface at the same time.
- Subdividing and breaking down complicated problems into smaller and more manageable pieces.
- Stepwise refinement.

3 Structure Chart / Hierarchy Chart

A structure chart presents the function of each routine in a program and the relationship among functions.

- It indicates what is required for a problem solution in terms of functions, whereas a flowchart indicates how a program is implemented in terms of the procedures required.
- Each module can only be called by the module immediately above it, and must in turn return control to that module.

3.1 Example: Structure Chart



4 HIPO: Hierarchical Plus Input, Process, Output, Diagram

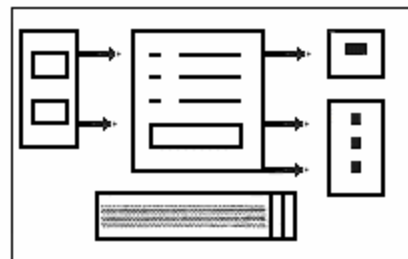
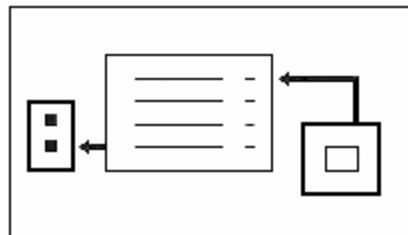
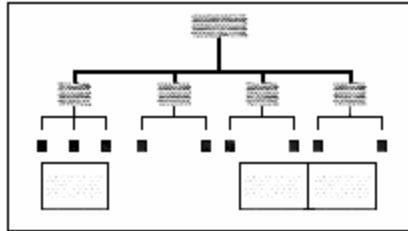
A HIPO: Hierarchy Plus Input, Process, Output Diagram:

- Specifies the input, the function and the output of each module in a program or system.
- Presents function, whereas flowcharts provide organization and logic.
- Provides a formal and standardized approach to the documentation of a top-down design.

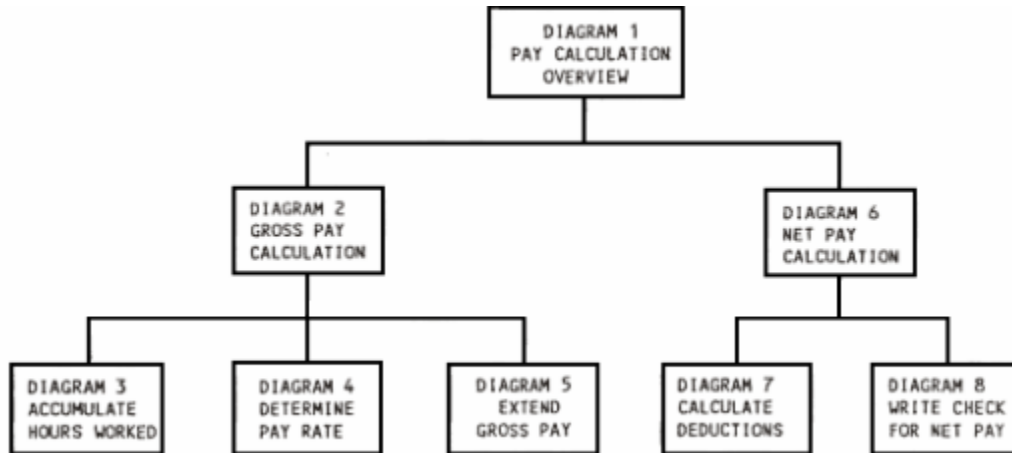
There are three types of diagrams:

- Visual table of contents
- Overview diagrams
- Detail diagrams

5 Three Types of Diagrams Associated with a HIPO

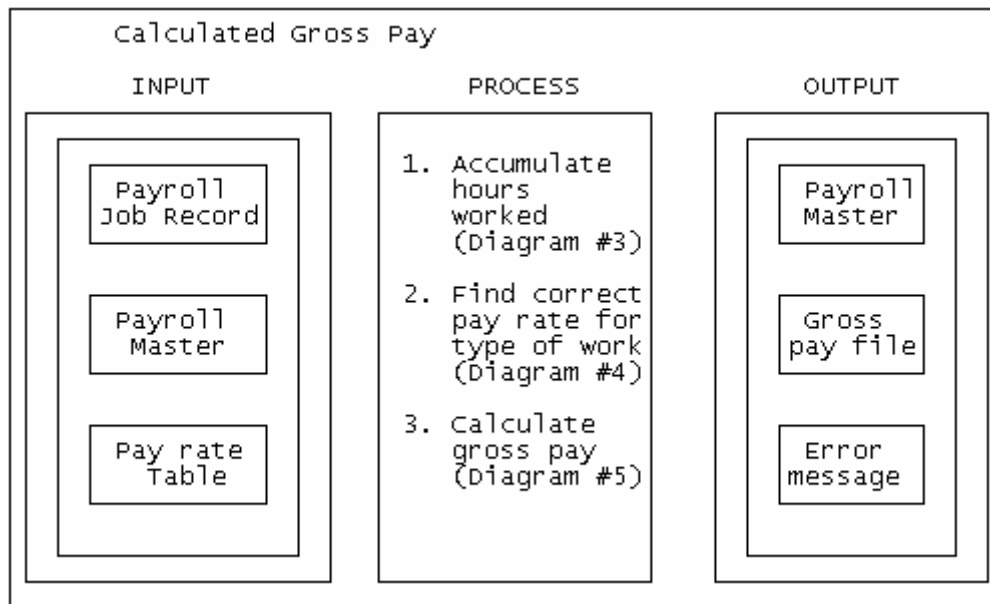


5.1 Example: Visual Table of Contents

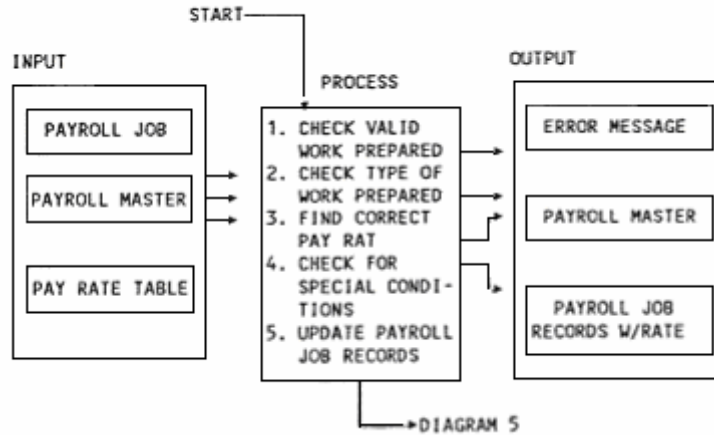


	CONTROL FLOW	1	OVERVIEW OF PAY CALCULATION FROM PAYROLL APPLICATION
	DATA TRANSFER	2	OVERVIEW FOR GROSS PAY FUNCTION
	ACCESS DATA	3 - 5	DETAIL OF GROSS PAY CALCULATION FUNCTION 3 - ACCUMULATE HOURS WORKED 4 - DETERMINE RATE 5 - EXTEND GROSS PAY
	DETAIL PICTURE	6	OVERVIEW FOR COMPLETE NET PAY CALCULATION
	FIELD OR VALUE TO BE ADDED	7 - 8	DETAIL OF NET PAY CALCULATION
			7 - PERFORM CALCULATION FOR ALL DEDUCTIONS 8 - WRITE PAYROLL CHECK AND DETAIL CALCULATION

5.2 Example: Overview Diagram



5.3 Example: Detail Diagram



Description	Flow Chart	Routine	Label
1. The program checks for a valid employee number if invalid job records are passed an error message.			
2. The program checks that the correct type of work was performed. If not job records are passed and error messages are printed.			
3. The program checks the master record and the pay rate table to determine the correct pay rate.			
4. The program checks for special conditions such as overtime shift pay or holiday pay to add to the rate.			
5. When all checks have been made, the program writes payroll job records with proper rate.			

6 Top-down Coding

The implementation of top-down coding entails that:

- Code is written in parallel with the various stages of design of a program.
- All design be completed prior to writing any code; this is the simple form.
- The design and code for each level must be completed before designing the next level; this the extreme form.

The benefits of top-down coding are:

- More precise and concise code; design information is communicated in a different format than with flowcharts and other design tools.
- Coding may reveal problems in the design of the program.
- Facilitating top-down testing.

7 Top-down Testing

Top-down testing is different than the classic bottom-up approach to testing.

First stage of testing: Unit test

- This ensures that all modules in a program function properly.

Next stage of testing: Systems test

- This tests the interfaces between the programs within a system.

Final stage of testing: Acceptance test

- This determines whether the programs in a system can be run without any help from the developers.

8 Bottom-up Implementation: Problems

The problems associated with a bottom-up implementation include:

- The complexity of integrating lower level modules.
- Testing is concentrated at the end of program development.
- The difficulty in finding errors; this is because several modules are being integrated simultaneously.
- A driver may be required for the lower level functions of a complex program. Since the driver must simulate the control logic of the program, it will be complex to code.

Guidelines

Modules should be tested from the top-down in a hierarchy chart in the order in which they are developed before coding is finished.

Lower-level modules should be initially coded as program stubs or dummy modules; this will allow for execution testing during development.

9 Top-down Testing: Benefits

The benefits of top-down testing include:

- Elimination of the need for system testing; testing is performed with the addition of each new segment of the program throughout development.
- Major interfaces of the program are tested first, revealing major bugs early in testing.
- Stepwise integration of code segments one at a time; this makes debugging easier.
- More efficient and better distribution of testing time.
- Elimination of the need for a driver; this is because control functions are coded first.
- Improved programmer morale which results from a combination of coding and testing.