Canada

DESIGN GUIDE

INFORMATION SYSTEMS MANAGEMENT

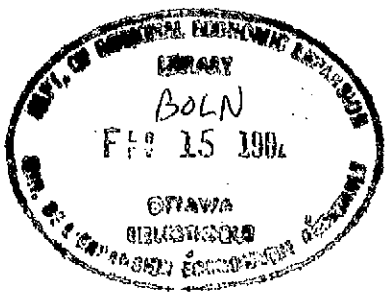
Regional Economic Expansion
Expansion Économique Régionale

Canada

DEPARTMENT OF REGIONAL AND ECONOMIC EXPANSION

SYSTEMS DEVELOPMENT LIFE CYCLE METHODOLOGY

DESIGN GUIDE



OCTOBER, 1981

I N D E X

NOTE: It is recognized that all roles referred to throughout this document will be filled by persons of either sex. However, to maintain readability, personel pronouns of the male gender are used.

He should be read as he/she.

His should be read as his/hers.

Him should be read as him/her.

SYSTEMS DEVELOPMENT LIFE CYCLE METHODOLOGY

DESIGN GUIDE

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

1.1 Purpose
1.2 Objectives
1.3 Scope
1.4 Other Aids

1. INTRODUCTION

1.1 Purpose

This manual has been prepared for the purpose of providing system designers with a guide to designing systems. Three areas of design are considered:

- computer system design, from the overall system structure or architecture to the detailed component specifications;
- test plan and test case design, to verify that what is created satisfies what was requested; and
- manual system design, done currently with computer system design, to facilitate the user's operation of the system.

During the Analysis phase, the "what" of the system was defined in terms of the application area. Now, in the Design phase, the "how" to achieve the "what" is described.

At the end of this phase, the system should be described to a level of detail to enable programming and testing to commence in the next phase. During this phase, functional specifications are packaged into system components and subsequently modules. The intermodule data flows are defined and a process narrative developed for each module. Test plans for modules and test cases are developed. The design of the user aids is also carried out.

1.2 Objectives

The structured technique is recommended for the design of a computer system. It is the optimal method of producing a GOOD design; good in terms of achieving the system objectives. A computer system has to satisfy a number of requirements:

- Performance - how fast and accurately the designed system will do the required work.
- Control - how secure is the designed system.
- Changeability - the ease with which the designed system can be modified later in its life to meet changing requirements.
- Usability - the ease of human use of the designed system.

There is a fifth requirement and this is the most important one. The designed system must satisfy the requirements of the user as defined by the requirements specified during the Analysis phase.

Structured Design gives a structure to the system that is hierarchical and functional. It isolates the components to allow:

- ease of change; individual components can be changed without affecting other components
- for the location of modules that most strongly affect the performance of the system (an industry rule-of-thumb for most systems is that 90% of the work is done by 10% of the system); and
- for the protection of critical components for control purposes.

1.3 Scope

The main inputs to this process are the Functional Specifications. These contain:

- A Logical design at a general level;
- Processing requirements at a detailed level;
- Data flow diagrams and supporting narrative; and
- Data dictionaries.

The inputs also describe the physical environment in which the system will reside including:

- the mode of the system - batch, interactive, transactional;
- the hardware;
- the configuration - distributed, local, centralized, timesharing; and
- the software - DBMS's, TP monitors, proprietary software, etc.

This manual will show the designer how to derive and document from these inputs the physical design of the system for both the manual and computer components.

Although the design of the physical data base is carried out during the Design phase, the procedure is not detailed in this guide since:

a) physical data bases address the logical data needs of the Department as a whole, rather than a single project's needs; and

b) the design of data bases tends to be dependent on the particular data base management software packages in use on the particular computers used by the Department.

Close coordination of activities between the data base administration and the project is required because of the dependency of a project on the physical design and development of the data base.

1.4 Other Aids

There are other manuals making up this methodology package that designers should be familiar with:

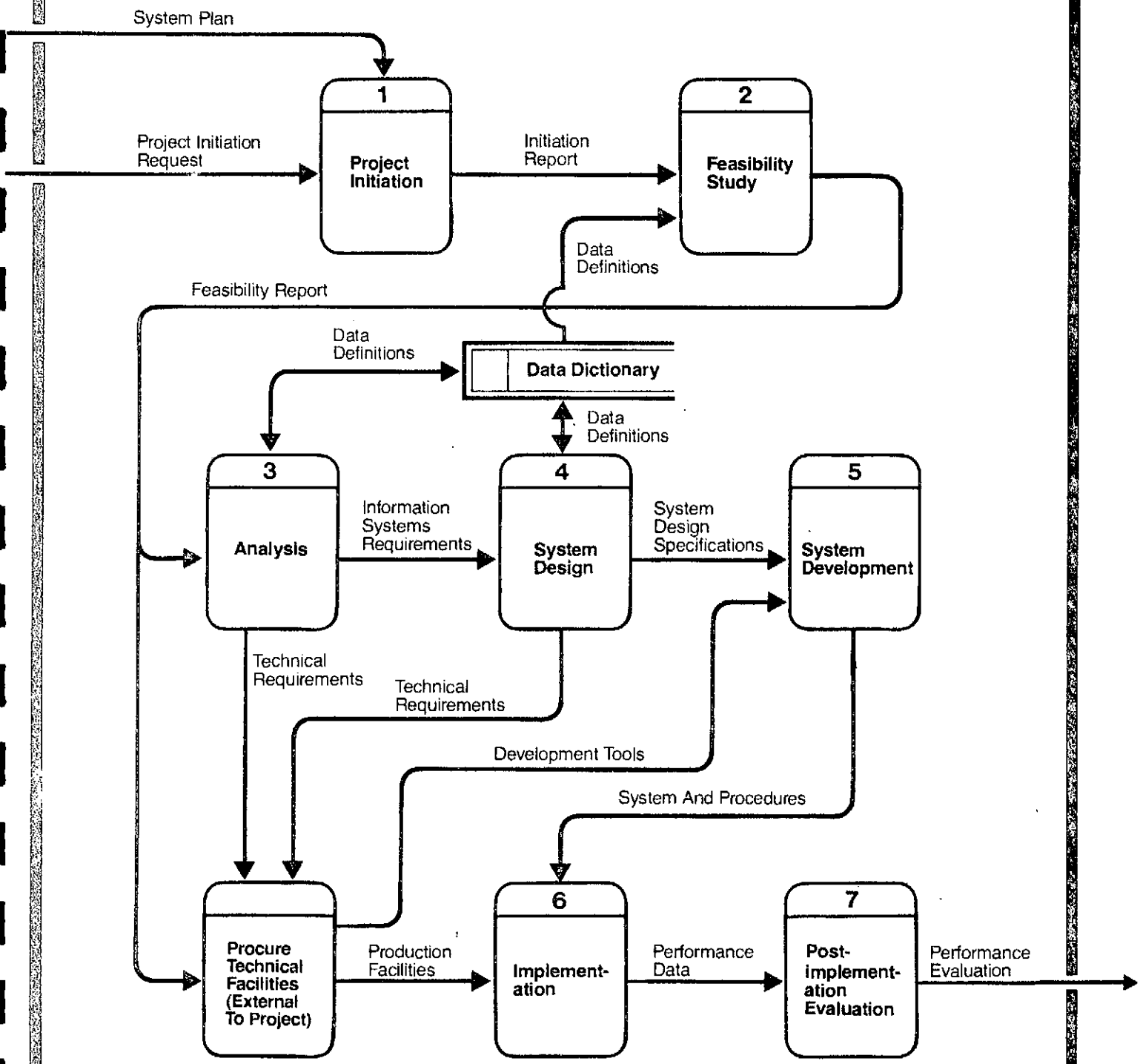
- Project Management Handbook; provides an overview of all the activities, milestones and controls throughout the system life cycle phases.
- Analysis Guide; defines the methods used to derive the Information System Requirements Specifications which form the basis for all subsequent work.
- Programming Guide; defines the programmers guidelines regarding the development of application programs.
- User's Guide; defines the user's role throughout the phases.
- Deliverables Reference Manual; defines in detail the deliverables, including the System Design Specification.

SECTION 2

SUMMARY OF SYSTEM DEVELOPMENT LIFE CYCLE

2.1 Overview Data Flow Diagram
2.2 Phase Summaries

Systems Development Life Cycle



2.2 Phase Summaries

The preceding diagram illustrates the phases comprising the System Development Life Cycle of most information systems project. It depicts how the cycle commences with the receipt of a Project Initiation Request and ends with the preparation of a Post-Implementation Evaluation Report following system implementation. The end of each phase represents a major checkpoint where management, external to the project, may review the continuing viability of the project and, as appropriate, commit only the resources needed to complete the following phase.

Summaries of each phase are as follows:

- Project Initiation

  Every project begins with the identification of an opportunity to be exploited, a problem to be solved, or a requirement to be satisfied. This phase starts when a request is received (on a Project Initiation Request form) from a user. The request is first screened to ensure that it is properly authorized, that the source of development funds is identified and that there is justification for proceeding further. Following this, details of the request are documented by a (Business) Systems Analyst. He/she prepares a brief Initiation Report which documents the issues to be addressed, objectives, scope, benefits, timeframe, policies, constraints and potential solution strategies.

  The objective of the report is to outline for management the initial perception of the issue, and to recommend an action plan to study the feasibility of various solutions.

  Normally, preparation of the report takes one half of a day or so.

- Feasibility Study

  This phase involves the Analyst working together with user management in the research and analysis of subject related data in order to identify various solutions. These solutions are both manual and automated, and are evaluated for their relevance and costs/benefits.

The overall objectives are to select a solution (or path), to develop a conceptual system design, and to secure further resources in order to perform a detailed analysis of the information systems requirements.

The Feasibility Study is carried out at a general, or conceptual, level. It provides management with an early opportunity to evaluate the project's viability before any substantial amounts of money have been expended, and to re-evaluate it in relationship to the user's priorities and strategies.

- Analysis

The Feasibility Study examined the issue being addressed by the system at a general level. The Analysis phase examines it at a very detailed level. The precise business processes and the set of information forming the business system is clearly defined.

This definition establishes the basis for outlining the system from a user perspective. For EDP systems, this means that at the end of this phase the user will know what information is included in the system and what business processes will be computer assisted.

The Analysis phase involves substantial end-user participation since it is during this phase that the business content of the system is documented in preparation for the design and development phases.

- System Design

Whereas the Analysis phase defined the "what" of the system, the System Design phase defines "how".

The specifications delivered by the Analysis phase represent the bridge between the user community, who collectively define the business requirements for the project, and the project designers who design a system to address these requirements.

User participation in this phase involves reviewing and approving more detailed aspects of the system such as report and screen layouts, office procedures, forms etc.

- System Development

  The objective of this phase is to develop the working procedures and, if automated, the computer programs according to the system design specification. Testing of the procedures and programs is also done to ensure that all components of the system work properly.

- Implementation

  In this phase the working procedures and programs developed are made operational. Users are trained in preparation for the live running of the new system, data files are converted from old media to new media, and the new system is installed. Parallel running, when applicable, takes place.

- Post-Implementation Evaluation

  This phase studies the operational performance of the system for a pre-determined period, and presents to management its conclusions and recommendations. Optionally, according to management's preferences, it may also study the effectiveness and efficiency of the development process itself.

SECTION 3

ROLES IN SYSTEM DEVELOPMENT LIFE CYCLE

3.1 Major Responsibilities by Phase (Matrix)
3.2 Summary of

3.1 MAJOR RESPONSIBILITIES BY PHASE

HASE	DELIVERABLE/TASK	APPROVAL AUTHORITY	MANAGEMENT AUDIT	USER MANAGEMENT	USER STAFF	PROJECT TEAM PROJECT MANAGER	PROJECT TEAM SYSTEMS ANALYST	PROJECT TEAM SYSTEMS DESIGNER	PROJECT TEAM PROGRAMMER
on	Initiation Report	Approve	Approve	Approve	Participate	Prepare			
ity	Feasibility Report -User Requirements -Conceptual Solution	Approve	Approve	Approve	Participate	Review	Prepare		
	Requirements		Review	Approve	Participate	Review	Prepare	Participate	
	Approval Authority Submission	Approve		Approve		Prepare			
sign	EDP Design Specification		Review		Participate	Approve	Review	Prepare	
	Design of User Aids			Approve	Participate	Review	Prepare	Participate	
	Approval Authority Submission	Approve		Approve		Prepare			
	Program Design							Participate	Prepare
nt	Program Code						Review		Prepare
	Program Test					Approve	Review		Prepare
	System Test		Participate			Approve	Prepare	Perform	Participate
	Operations Manual					Approve	Participate	Prepare	Participate
	User Manual			Approve	Participate		Prepare	Participate	
	Procedures Manual			Approve	Update		Participate		
	Training Manual			Approve	Participate	Review	Prepare		
	Approval Authority Submission	Approve				Prepare			
tion	Acceptance Test		Participate	Approve	Perform		Participate	Participate	Participate
	Conversion			Approve	Participate		Participate	Participate	Participate
	Production Operation				Perform				
	Approval Authority Submission	Approve				Prepare			
	Evaluation Report	Approve	Participate	Approve	Participate	Approve	Prepare		
tion									

3.2 Summary of Roles

Approval Authority

The Approval Authority for any information systems project may be a systems management committee, a project steering committee, the head of ISM (Information Systems Management) or a senior functional manager depending upon the nature of development project.

The Approval Authority acts on behalf of the user by approving each of the end-of-phase submissions, by allocating resources to each project phase, and by maintaining control over the project's progress. These responsibilities are exercised through periodic receipt of documents and submissions from both the Project Manager and the Systems Assurance Manager. Refer to the Departmental ISM policy manual for specific policies related to the approval process.

Business Systems Analyst

See: Systems Analyst

Data Analyst

A Data Analyst provides functional guidance and support to the project on matters related to the logical representation of data in project specifications. A Data Analyst is a specialist in data and data relationships. External to projects, he models the department in terms of its data for the purpose of developing efficient, cost effective data management facilities, e.g., data bases. In order to achieve this he must develop data models for each project application and synthesize them into the Departmental data model.

NOTE: The Data Analyst's role may not be a full-time staff position. The role may be filled by staff with other responsibilities.

Inspector

An Inspector reviews project specifications in order to assure their quality prior to release external to the project. In this regard he examines specifications for consistency in level of detail and style, and adherence to standards. He also looks for incompatabilities among related documents.

Depending upon the size of the project team and the volume of project deliverables, the Inspector may be one individual appointed for the duration of the project, or he may be any member of the

Programmer

A Programmer designs, develops and tests program modules using structured programming techniques. He may also be required to perform duties in system testing, acceptance testing, conversion and post-implementation support.

See Programming Guide for further details.

Project Manager

The Project Manager has overall responsibility for achieving the project goals through the day-to-day conduct of the project. In this respect, he develops operational plans and budgets, acquires the required resources, identifies and organizes the appropriate business and technical expertise, periodicaly submits plans, requests for approval and progress reports to the approval authority, coordinates with user management and the Systems Assurance Manager user participation in the project, conducts regular project management progress meetings and ensures effective quality control over project deliverables.

See Project Management Handbook for further details.

Steering Committee

See: Approval Authority

Systems Analyst

A Systems Analyst identifies, analyzes and specifies information systems requirements using structured analysis techniques. He may also carry out ancillary duties involving user interface such as development of user manuals, training, system conversion, and acceptance testing. Systems Analysts may be members of a user section or branch (Business Systems Analysts) or may be drawn from ISM staff.

See Analysis Guide for further details.

Systems Assurance Manager

The Systems Assurance Manager represents the departmental interest in a systems project and is responsible for ensuring that all user-related matters pertaining to quality control are addressed. Acting on behalf of the user, the Systems Assurance Manager:

- participates with the Project Manager in planning the commitment of user resources to the project;

- ensures that the appropriate level and quality of user resources are available to the project (i.e., that sufficently senior user personnel are assigned the key review and sign-off roles for all user-related deliverables produced by the project team);

- ensures that the user community's participation is comprehensive and active;

- verifies that the Project Manager has obtained user sign-off of all user-related deliverables (it is the responsibility of the Project Manager to obtain each sign-off);

- verifies that any changes to project plans which impact the user community have been agreed and approved by the user community;

- brings forward user concerns regarding the project to the Steering Committee for resolution if and when these concerns cannot be addressed through negotiations between the Project Manager and the user community;

- reports to the Steering Committee on user satisfaction with the project.

Ideally, the Project Manager and the Systems Assurance Manager should work cooperatively to support the successful execution of the project. Situations may arise, however, in which the Project Manager and the Systems Assurance Manager disagree (i.e., the Systems Assurance Manager may request, on behalf of the users, the expansion of the project scope, beyond the terms of reference understood by the Project Manager). The Project Manager and the Systems Assurance Manager are jointly responsible for making every effort to resolve any such disagreements to the mutual satisfaction of the project team and the user community. Disagreements should be brought forward to the Steering Committee only when resolution cannot be achieved through negotiation.

System Designer

A System Designer transforms information systems requirements, in the form of functional specifications, into system and sub-system design specifications using structured design techniques. Although a System Designer is normally the designer of the computer internals - system transactions, screens, files, input, output, etc. - this role may also encompass design of user aids such as training packages and user manuals.

Technical Specialist

A Technical Specialist provides functional support and guidance to the project on matters of a technical nature. These would include hardware studies, telecommunications networking, technical feasibility of design alternatives, and acquisition and use of development tools.

He is considered "external" to any project and his abilities are shared on an organization-wide basis. This is to optimize the economic efficiency of using specialized technical staff.

User

The User's role in the Systems Development Life Cycle relates to those activities which have direct impact on him and his area of responsibility. These include:

- definition of systems subject matter;
- planning and provision of subject matter expertise;
- delegation of authority to staff assigned to participate in development activities;
- quality control over subject matter documented by the project team;
- training of staff;
- preparation of administrative environment for system installation;
- approval and acceptance of project deliverables.

In some sections or branches, user staff may also be engaged in carrying out development roles, such as systems analysis. These are not considered user roles.

See the User's Guide for further details.

SECTION 4

THE METHODOLOGY

4. THE METHODOLOGY

4.1 Introduction

This section describes the inputs, working documents and deliverables that System Designers are to use for developing system specifications. In addition, it describes the method for designing a system, which is:

- Divide the System into Sub-System Components, specifying the interfaces. This gives the preliminary physical structure of the new system, and indicates how the components communicate with one another and with external entities.

- Design the Sub-System Structure. This provides the internal architecture or framework around which the remainder of the physical system is designed.

- Design Detailed Components. Each module identified in the above process is described in detail.

- Design the User System. The user system and aids are designed in an initial form.

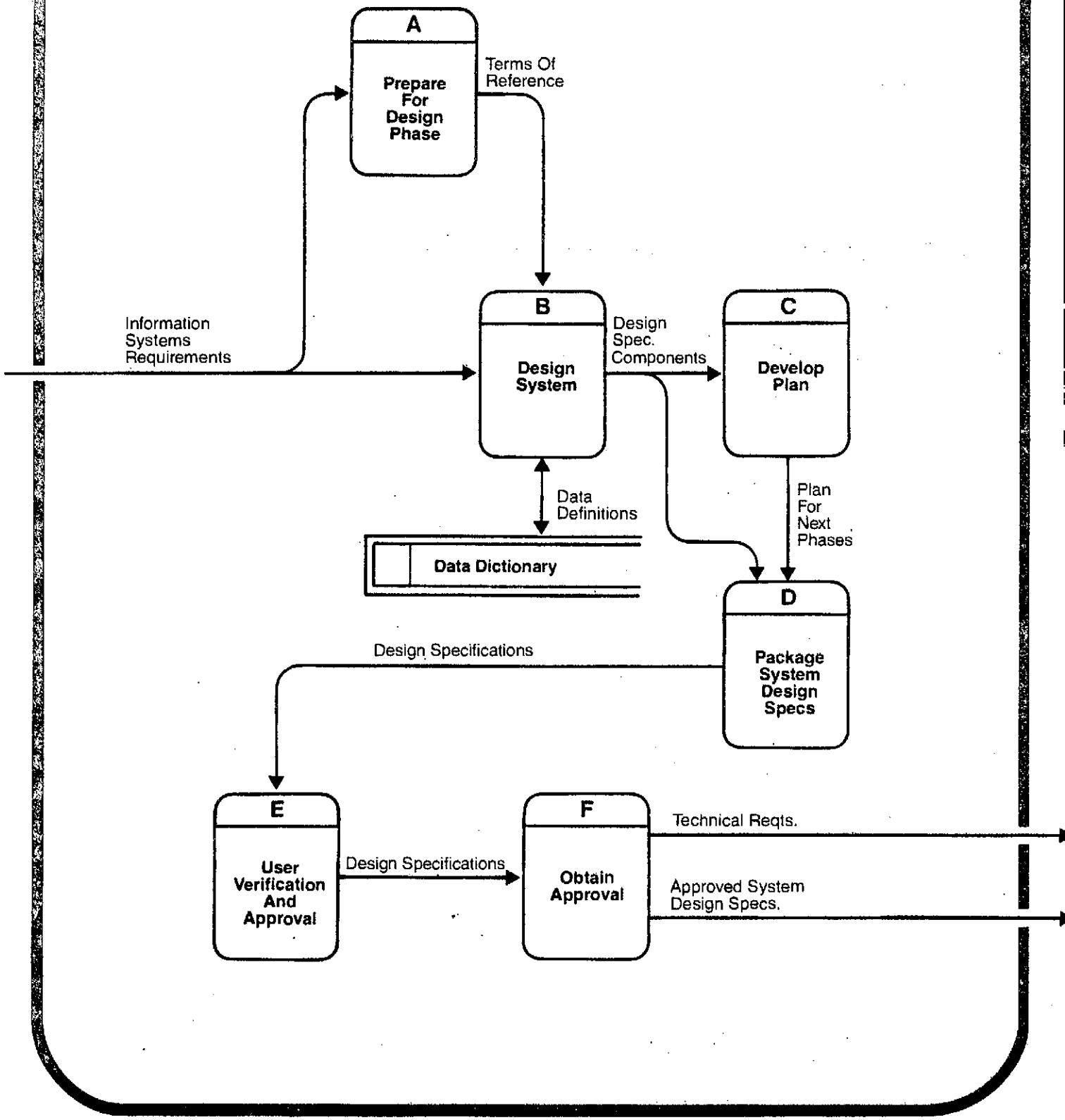
- Design the System Test. The test plan and test cases are prepared.

- Design Conversion Procedures. Modules, user aids and plans are developed for conversion from the old system to the new one.

A data flow diagram of the process showing the information connections between the steps is shown on the next page.

Management discretion should be applied when planning activities for small or modest sized projects where less rigorous formality may be appropriate. Some of the steps could be combined so that some of the work would occur simultaneously.

Phase 4 — System Design

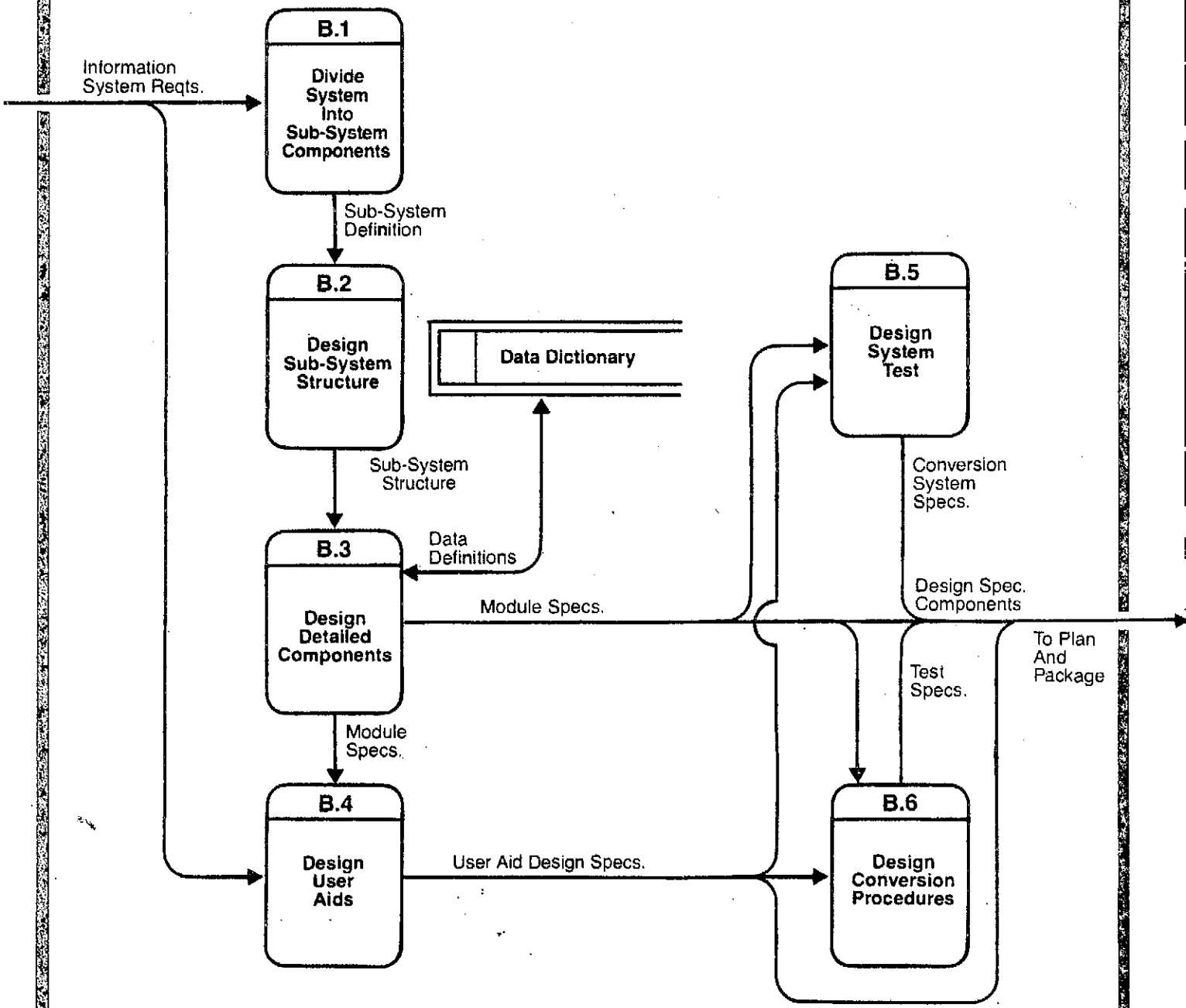


4.2 Prepare for Design Phase (4.A)

Designers must become familiar with the project, its objectives, and the output of the Analysis phase before attempting any design. Ideally, project management will include designers in the Analysis "inspection" process so that there is more than just documented bridges between the Analysis and Design phases. However, documented background material can be found in the Feasibility Report, and the following sections of the Information Systems Requirements are essential input:

- Executive Summary
- Functional Specifications
- Performance and Security Goals
- Cost/Benefits
- Technical Requirements

Activity 4.B – Design System



4.3 Divide the System into Sub-System Components (4.B.1)

Objectives

The objectives of this step are:

- to divide the overall logical system into component sub-systems;
- to further develop the physical system structure by detailing the man-machine interfaces; and
- to confirm requirements for the the technical environment.

Input

Information System Requirements

- Functional Specifications
- Performance and Security Goals
- Technical Requirements
- Cost/Benefits

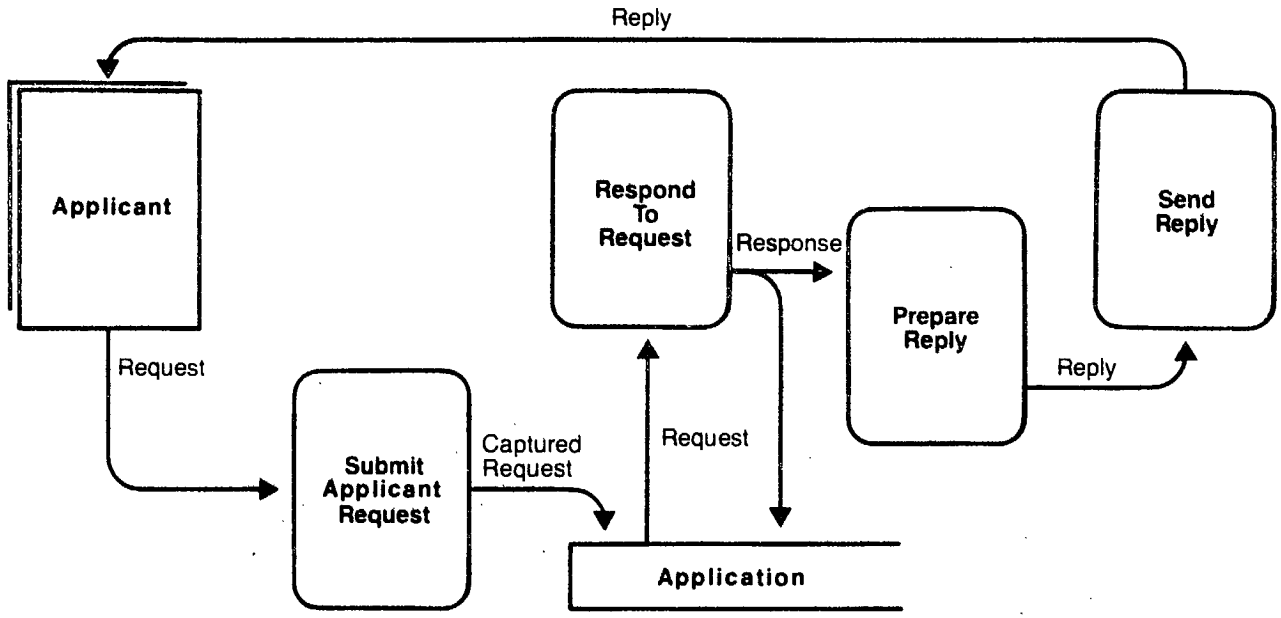
Methods

- Summary of Steps
  - divide the system into sub-systems
  - define the inputs
  - define the outputs
  - verify sub-system division
- Divide the System into its Sub-Systems

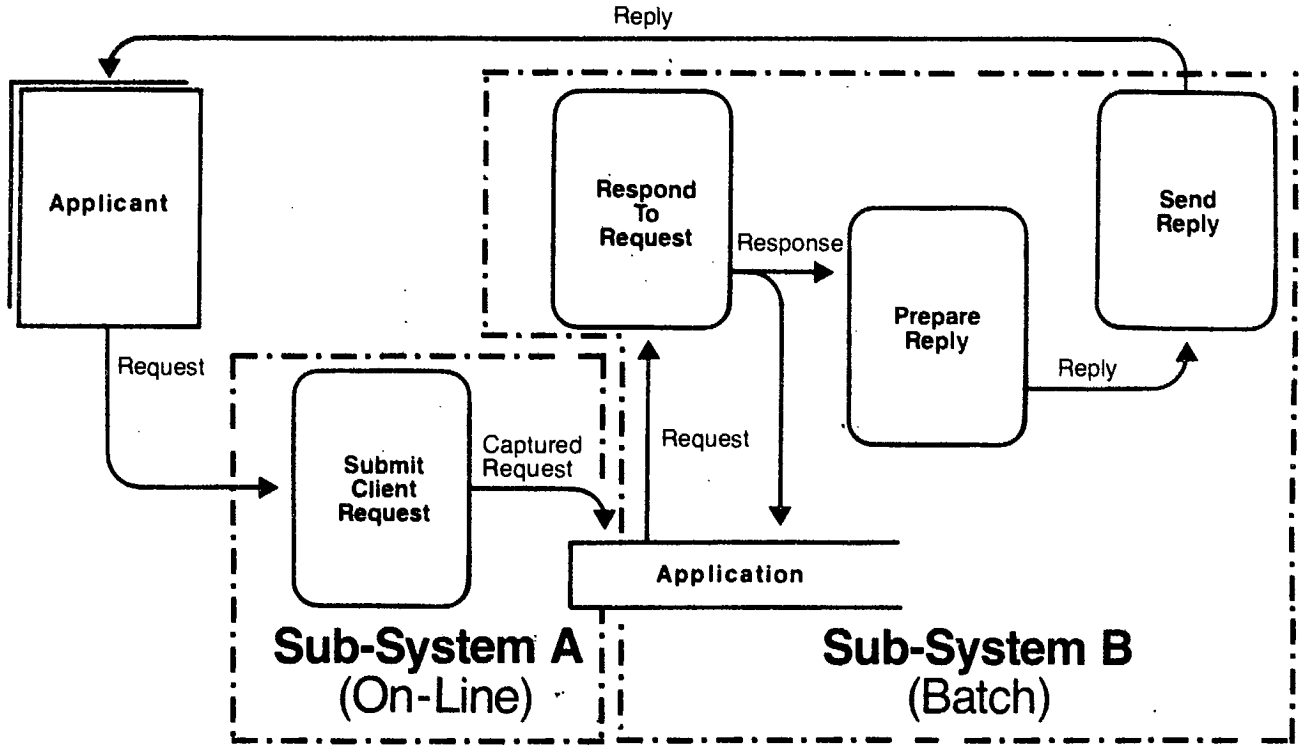
  Use the system data flow diagram to divide the system into its component sub-systems (see the figure on the next page); each being a piece of software which could run on its own or in conjunction with the others.

  The criteria to use in this process are the capabilities and limitations of the hardware, the user requirements and the cost feasibility.

Data Flow Diagrams
Example of System Division



Logical Representation



Physical Representation
Division Into Subsystems

- Define all inputs to the computer system. These inputs may consist of data entered on-line, data entered in documents, data entered as an existing automated file, or data received from another automated system.

  For each on-line input record the following information:

  - screen name
  - screen number
  - screen description
  - screen specification
  - screen format
  - program(s) involved
  - file(s) created.

  For each document input to the system record the following information:

  - document name
  - document number
  - document description
  - document sample
  - encoding instructions
  - program(s) involved
  - file(s) created.

  In most instances an automated file input represents an interface with another system. Identify the file name and input name.

- Define all system outputs. Outputs from a system may consist of on-line screen reports, printed reports or automated files. For screen reports note the following:

  - screen name
  - screen number
  - screen description
  - screen specification
  - screen format
  - program(s) involved

  For each printed output from the system define the following information:

  - report name
  - report number
  - report description
  - report layout
  - report frequency
  - report volume
  - program(s) involved

In most instances an automated file output represents an interface to another system. Specify the following:

- file name
- dispersal specification

Complete details of the specifications will be found in the System Design Specification contained in the Deliverables Reference Manual.

. Verify the sub-system division.

A walkthrough of the deliverables from this activity is conducted by the designer to verify the design and its technical feasibility. Ideally, the following should be in attendance:

. the design team leader;
. a technical specialist from ISM;
. a data analyst (representing the data base perspective); and
. an informed user representative (optional).

Following group approval (subject to modifications being agreed upon), user community representative should be asked to approve the finalized inputs/outputs and interfaces.

Working Documents

Walkthrough Records

Deliverables

. Data Flow Diagrams and complete descriptions of computer sub-systems

. Sub-system Interfaces - inputs, outputs

. System Flow charts for each sub-system

4.4 Design Sub-System Structure (4.B.2)

Objectives

- To form sub-system structures by synthesizing functional, performance, security, and physical requirements for each sub-system.
- To develop the physical data structure by transforming conceptual files and data flows into system data descriptions (files, messages, transactions)

Inputs

- Functional Specifications
- Performance and Security Goals
- Technical Requirements
- Data Flow Diagrams of Sub-Systems
- Sub-System Interfaces

Methods

- Summary of Steps
  - Decompose the Functional Computer Design to the initial system structure
  - Refine Design
  - Alter data flows and conceptual files from the Functional Specifications into system data descriptions (file design, messages)
  - Ensure Data Dictionary requirements are met
  - Verify System Design
- Decompose the Functional Computer Design to the initial system structure.

  The technique of transform/transaction analysis is to be used in deriving the initial structure of the system in the form of a structure chart.

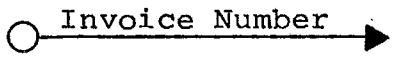
  A structure chart is a diagramatic method of showing the hierarchy of modules and the relationships among them, specifically the control connections and the data connections. A brief explanation of how structure charts are drawn, is given in the text which follows and in the example provided on the next page.

STRUCTURE CHARTS

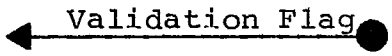
SYMBOLS AND NOTATIONS

Connector - The line connects the called transform

Parameters - System Data



- Control Data or "Flags"

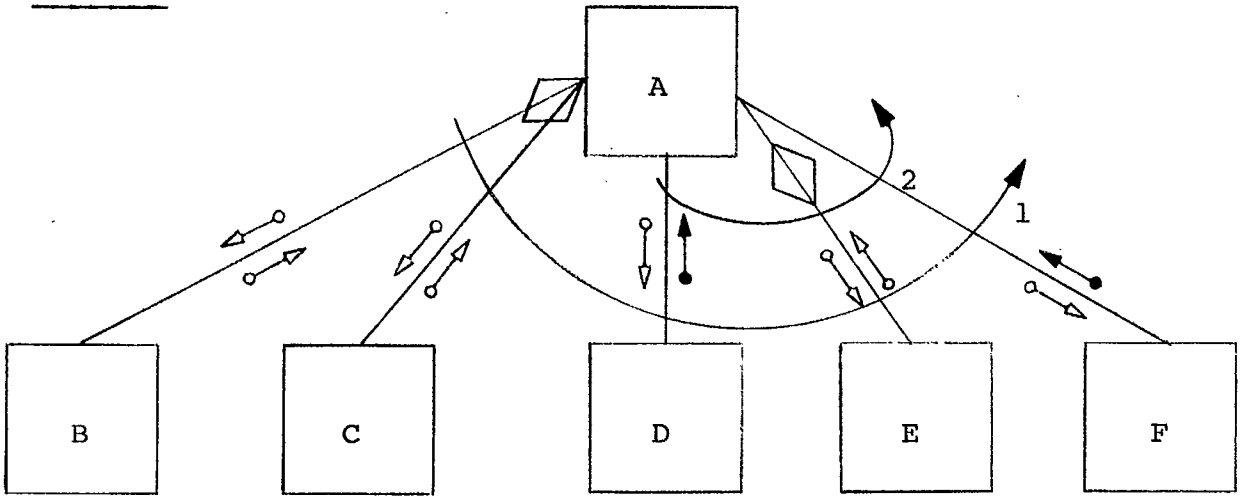


Procedural - decision or condition calls

- iterative procedure or loop

- convention, procedures performed left to right top to bottom

EXAMPLE



- "A" performs the large loop (1) of "B", "C", "D", "E", and "F"
- within loop 1, "A" decides to call either "B" or "C" and "A", then performs the loop of "D", "E", and "F", deciding within that loop, whether or not to call "E.

Synopsis of Transform Analysis

1. The top of the Structure Chart is chosen to correspond to that part of the Data Flow Diagram where inputs change to outputs.

2. Define one second level module for each major input stream, output stream and process stream.

3. Data Flows in the Structure Chart correspond to Data Flows in the Data Flow Diagram.

4. The input and output portions of the Structure Chart correspond on a one-to-one basis to the processes of the input and output legs of the Data Flow Diagram input process.

5. Data Flows in the substructures correspond to the data flows into and out of the associated processes in the DFD.

   Reference material is identified in Appendix A.

. Refine Design

Any creative process requires an iteration, a look at the end product and then a revision and refinement. This is especially needed in the complex case of information systems design. Once the structure has been initially designed, it must then go through a refinement process. This is to verify its correctness in terms of the functional specifications, to enhance its capabilities, to achieve the performance, control or changeability objectives, and to verify the module's position in the rest of the system.

The first step is to verify that it is a "good" design according to the methods of evaluation of Structured Design (see Composite Structured Design, referred to in Appendix A):

- minimize coupling - the connections between modules
- limit span of control - no more than 7 subordinates
- have the scope of effect contained within the scope of control
- limit module size.

The design must also be refined to fit into the technical and organizational environment of the system. To this end consideration should be given to the organizational impact and operational efficiency of the design.

. Alter data flows and conceptual files (data stores) from the Functional Specifications, into system data descriptions.

In a highly modularized system the greatest source of problems occurs in the inter-module and inter-sub-system data flows. A lot of care and attention must be paid to the design and specification of these parameters.

The logical contents of the files plus access would be given in the Functional Specification. Considering these specifications and the technical environment in which the system will operate, the physical files are designed in an initial form. (If a data base is required by the user, then the Department's Data Base group would design the physical file structure).

Similarly, all system messages and inter-module communications would be specified. The inputs to this specification would be the data flows (couplings) on the DFD's and the data element and data structure descriptions from the Data Dictionary.

. Data Dictionary Requirements

At all stages in the Design, the data dictionary must be kept up to date with the addition of new or altered Data Structures or Data Elements.

. Verify System Design

A careful verification of the structure, module interfaces and sub-system interfaces is critical as each module has an impact on the rest of the system. At a minimum, the designer should carefully check these elements. However, the documents should also be inspected through a formal inspection process. See Appendix B for description of a "walkthrough".

Working Documents

- Walkthrough Notes
- Initial File Design
- Initial Interface Descriptions

Deliverables

- Structure Charts
- Functional coverage matrix that ensures that all of the functions are covered
- Updated Data Dictionary

4.5 Design Detailed Components (4.B.3)

Objectives

To build a detailed specification for each component or module. It is important, when writing program specifications, to ensure that all of the information necessary for writing a program is provided.

It should also be remembered that a designer's function should be to define a problem, showing the inputs, outputs and the rules as to how the data will be manipulated. It is not his function to tell the programmer how to write and structure his program.

Inputs

- System Structure - the system structure chart
- System Data Descriptions - initial file design from the first activity plus logical file design from the Functional Specifications
- Component Data Flows - initial interface descriptions
- Data Dictionary

Methods

- Summary of Steps

  - Derive Module Structure from Structure Chart
  - Define detailed module interfaces
  - Develop module process descriptions
  - Verify the completeness and accuracy
  - Package all of the design components into the program specifications.

- Derive Module Structure From Structure Chart

  The structure chart gives all of the functional components of the system. The Designer now has to package these into program modules and uniquely identify each module. Normally, a program module is defined for each module on the structure chart. However, there may be some cases where the code of a module should be included in the code of its superordinate module. This will occur when there are too

few lines of code to justify having a separate compilable module with all of the associated overheads. This is referred to as "lexical inclusion" in Structured Design, and is shown in the structure chart with a triangle on top of the module to be included (see the next page for an example).

In the example, module 6.4.1 consists of a check-digit verification routine probably 10-20 lines of code. So it is more feasible to include the code in module 6.4.

. Define Detailed Module Interfaces

Now that the physical module structure has been specified the module interfaces have to be specified in detail.

Included are:

- names of data passed
- contents of data passed
- methods of passing the data
- physical description of data
- logical description of the passed data

The Data Dictionary must be updated to include these new physical details.

. Develop Module Process Descriptions

The intent of this section is to clearly identify the processing functions of the program.

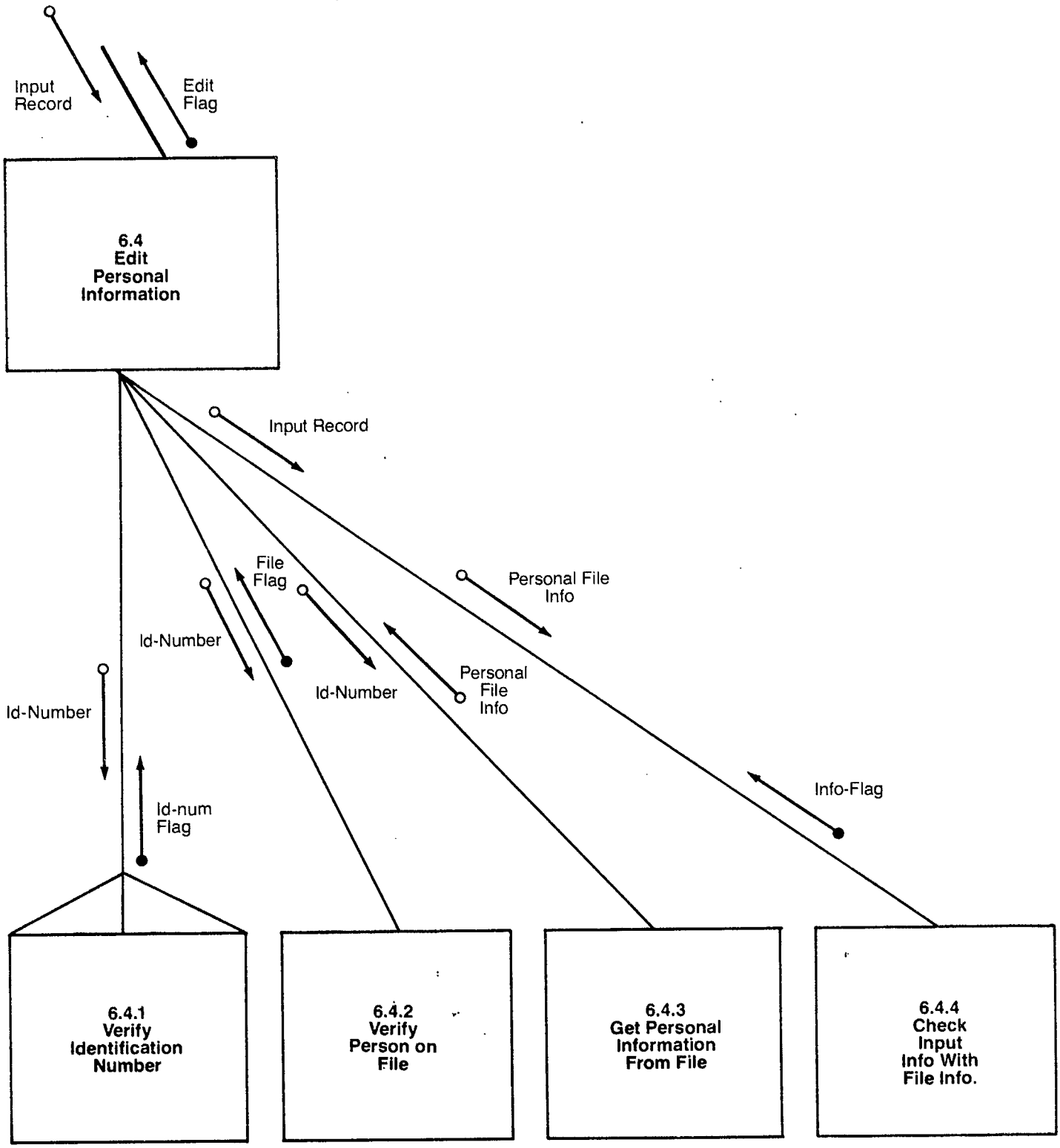
The method of documenting the process could be one or a combination of:

- narrative
- structured English or Pseudocode
- Decision Trees
- Decision Tables
- Flow Charts

Structured English or Pseudocode

Pseudocode is narrative that is used to describe a process within a system. It is more structured and formal than prose but not so formal as to be compilable.

Program Structure Chart



Pseudocode describes a module's functions in the order which they will be performed, together with the logical decisions that control the processing.

Pseudocode lends itself to "stepwise refinement". Thus, pseudocode statements are often vast generalities, such as "calculate pension entitlement", that are subsequently elaborated into hundreds of statements of code.

Note: The description which follows is meant as a guideline. It has been reviewed by practioners, and is hopefully complete. It should not be deviated from without reason. But neither should it be slavishly adhered to when you have a better idea.

There are four types of pseudocode statements:

- Unconditional imperative statements;
- Statements causing exception condition;
- Statements that express flow of control; and
- Module calling statements.

1. The unconditional imperative statement is the simplest kind. It causes no side-effects, it raises no exception conditions. e.g.:

```
MOVE transaction name to master name.
```

2. Statements causing exception conditions. Many statements can give rise to conditions that must be tested. These statements are usually only encountered in low-level, detailed pseudocode. For example, in COBOL:

```
READ input-record from trans-file
     AT END
          MOVE high-values to input-record
                or,
COMPUTE sum-squares = (data*data) + sum-squares
     ON OVERFLOW
          MOVE 'Y' to error-flag
          MOVE zero to sum-squares.
```

3. Statements that express flow of control.

   . IF, as in:

```
IF the cat is hungry THEN
   FEED the cat
ELSE
   SEND the cat outside
ENDIF.
```

   . CASE, as in:

```
CASE gender of
   male:
      PERFORM animus
   female:
      PERFORM anima
   neutral:
      PERFORM psychoanalysis
ENDCASE.
```

   . FOR, as in:

```
FOR each entry in table
   ADD entry to sum
ENDFOR.
```

   . DOWHILE, as in:

```
DOWHILE end-of-file not true
   APPLY transaction to master file
   READ next transaction
ENDDOWHILE
```

   . RETURN or STOP.

4. Module calling statements:

   . PERFORM check-trans-code using new-code

   . CALL calculate entitlement using
      new record
      new entitlement

   . INVOKE P3726 (Error-code, Personal-Identity-Table)

   At the high level of design, you may not want to explicitly define how the module is to be called (e.g. internal with PERFORM's or external with CALL's). In that case, just state the function:

   . CHECK transaction code
   . CALCULATE entitlement

Decision Tables

Decision tables are a tabular method of describing logic by showing the relationship between conditions and actions in a compact manner.

In the standardized format, the decision table is divided into four sections. The "condition statements" are listed in the upper left hand side with the subsequent "action statements" listed below. On the right-hand side are the "condition" and "action". This side is divided into a number of columns or rules. Each rule corresponds to one set of conditions being fulfilled and shows the action(s) to be performed in these circumstances.

Tables can be completed in one of two ways, "limited entry" or "extended entry". With a limited entry table, the statement of each condition or action is completely self-contained. The entry portions of this table indicate whether a particular rule satisfies the condition or requires the action stated. The entries are YES (Y), NO (N) or a "hyphen" (-) if the condition is not pertinent to the rule.

In an extended entry table, the statements of condition and action are incomplete. Both the statement and the entry sections of any particular row in the table must be considered together to decide if a condition or action is relevant to a given rule. The entries can be any statement. The advantage of the extended entry method is a saving in space, although an extended entry table can always be converted to a limited entry form. The following pages show the two methods, used to state the same problem. Further reference material is identified in reference 6 of Appendix A.

LIMITED ENTRY DECISION TABLE

STATEMENT / CONDITION	RULES 1	2	3	4	5	6	7	8
QUANTITY-ORDERED $\geq$ DISCOUNT-QUANTITY	Y	N	N	-	Y	N	-	Y
WHOLESALE	Y	Y	N	N	Y	-	N	N
QUANTITY-ORDERED $\geq$ QUANTITY-ON-HAND	Y	Y	Y	Y	N	N	N	N
ACTION								
BILL AT DISCOUNT-RATE	X	-	-	-	X	-	-	-
SHIP QUANTITY-ORDERED	X	X	X	X	-	-	-	-
BILL AT REGULAR-RATE	-	X	X	X	-	X	X	X
SHIP QUANTITY-ON-HAND	-	-	-	-	X	X	X	X
BACKORDER QUANTITY-ORDERED LESS QUANTITY-ON-HAND	-	-	-	-	X	X	X	X

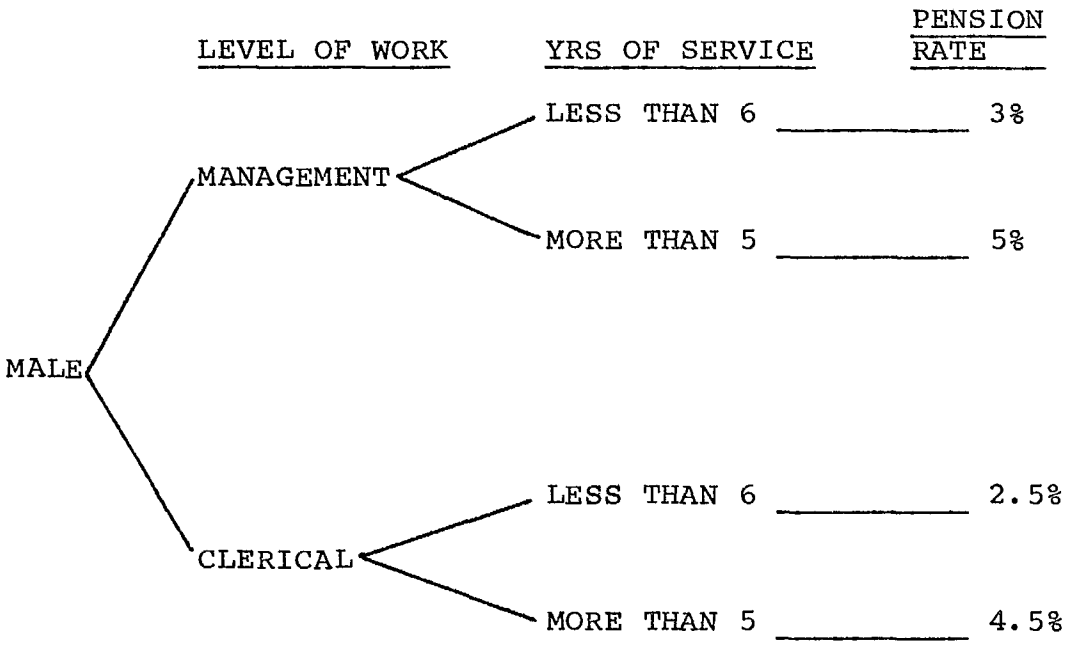
EXTENDED ENTRY DECISION TABLE

STATEMENT	RULES						
CONDITION	1	2	3	4	5	6	ELSE
ORDERED-DISCOUNT	Y	Y	Y	Y	N	N	-
ORDERED-ON-HAND	Y	Y	N	N	Y	N	-
BUYER	WHOLESALE	RETAIL	WHOLESALE	RETAIL	-	-	-
ACTION							
BILL	DISCOUNT	REGULAR	DISCOUNT	REGULAR	REGULAR	REGULAR	-
SHIP	ORDERED	ORDERED	ON-HAND	ON-HAND	ORDERED	ON-HAND	-
BACKORDER ORDERED LESS ON-HAND	-	-	X	X	-	X	-
INVESTIGATE ERROR	-	-	-	-	-	-	X

Decision Trees

Decision Trees can be used for logic verification of moderately complex decisions which result in up to 10-15 actions.

e.g. Pension Rates



Flowcharts

Flowcharts should be prepared in accordance with the Section standard practices used in the ISM section.

. Verify the Completeness and Accuracy

A careful verification that the program specifications address all required issues, greatly reduces problems in the programming phases. This verification can be performed in a number of ways, depending on project policy and circumstances.

Specifications must be checked by another member of the project team as well as by the designer. If at all possible, a formal walkthrough should be conducted. It is highly desirable to have the programmer who is assigned to do the programming participate in the "walkthrough". A description of the procedures to use for "walkthroughs" can be found in Appendix B.

A checklist of problem areas in component design should include:

- Logic
- Test and branch conditions
- Data area usage
- Return codes, messages
- Module attributes - is the module well structured?
- External linkages
- More detail/less detail needed
- Standards
- Higher level design documents
- User specification
- Maintainability
- Performance

. Package all of the Design Components into Program Module Specifications

The final task is for the Designer to gather all the components into the Program Module Specification.

In spite of program specifications being one of the most important facets of designing and implementing a computer system, they are quite often the most neglected; especially in the areas organization and completeness. The success of the system can be seriously and adversely affected by poor program specifications. Specifications should be written and packaged in a clear and organized fashion. There should be no ambiguities or misunderstandings of the requirements by the programmer. When organized in a standard manner, the Designer responsible for the system can quickly check the program specifications against the overall systems design, and also ensure that all aspects of the requirements are covered.

Complete details of the format can be found in the System Design Specifications Section of the Deliverables Reference Manual.

Working Documents

. Walkthrough Notes

. Designer's Notes
  - the Designer should keep the informal notes on file for later use by the Programmer.

Deliverable

Completed and verified Program Module Specifications
  - refer to the Deliverables Reference Manual for a detailed list of contents.

4.6 Design User Aids (4.B.4)

Objectives

At some time during the project life cycle process, there must be specific attention paid to the manual sub-system. As it is the system that most of the users come in contact with, it is just as important as the computer sub-system, (and in the eyes of the user staff, often more important). The design of the manual system is further detailed in this activity.

Inputs

- Functional Specifications - including man-machine interfaces, manual processes, inputs and outputs, and organizational requirements.

- System and Sub-system Specifications - giving detailed man-machine interfaces which could consist of input documents, input screens, input dialogues, output screens and/or dialogues, and output reports.

Methods

- Summary of Steps
  - Refine manual requirements
  - Refine manual operations
  - Design user aids
  - Test and refine the design

- Refine the Manual Requirements

  The Functional Requirements of the system as it relates to interfaces with end users, will have to be finalized.

  - Identify the end user audience:
    - who will enter or receive the data?
    - what are their present skills?
    - can their skills be changed through training?
    - are the users a homogeneous group?

  - Identify the Business Procedures - those sets of procedures that are required either before or after the actual man-machine interface.

. Refine the Manual Operations

Once the requirements have been refined and the man-machine operations are known, then the manual operations required to use the system in the business environment can be defined in detail.

. Design User Aids

Based upon the requirements and the existing skill levels, the aids required to bring existing skill up to the required levels and to operate the system can now be designed. The aids could take the form of:

- User Manuals
- Procedure Manuals
- Training programs/guides
- Computer operations Manuals

An analysis of the needs of the system will have to be performed to decide on the specific user aids required.

User Manual

During implementation of the system, and for the entire life of the system, the users' prime information document will be the User Manual. It will serve the users in three ways:

a) The User Manual will provide management with an overview of the total system, its functions, and its features.

b) The User Manual will serve as a technical reference document for line management and other system users.

c) In conjunction with the Training Manual, it will aid in training users through the use of examples and explanation of tasks.

Procedure Manuals

Procedure Manuals contain documentation of office procedures for the manual and automated systems, and administrative activities followed in an organizational unit.

This differs from the User Manual in that the User Manual is oriented towards describing a specific computer system. The procedure manuals have in contrast, an orientation toward a job function and specify when activities take place within a given organizational unit. Procedure manuals normally include documentation on: policies, regulations, authorities, responsibilities, and detailed document flow.

Training Manual

The Training Manual is intended to be a supplementary document to the User Manual. It provides additional examples of how to perform specific tasks, highlights particularly complex situations, specifies what periodic training is to take place, and basically serves as an organized means of storing training material.

It will be used during the initial training of personnel in the operation of the new system and will also serve as a permanent staff training aid.

Computer Operations Manual

The Computer Operations Manual is designed to fully describe all phases of the operation of a computer system, both by internal operations and any service bureau from which services are purchased.

- Test and Refine the Design

The design should be verified in detail, internally, by the project team, and especially by the user representative as these aids are often the main interface between the user and the system.

Methods of Verification:

- Informal checks
- Formal inspections
- Walkthroughs

Working Documents

N/A

Deliverables

User Aid Design Specifications
for each applicable user aid:
- Objectives
- Tables of Contents
- Point Form Details

4.7 Design System Test (4.B.5)

Objectives

The objective of system testing is to compare the system to its original objectives. To achieve this, the following are done:

- Requirements for testing the functions of the new system are defined.

- A plan and test cases are developed for testing of groups of components, versions of the system and sub-systems, and the entire system in as realistic an environment as is possible.

Inputs

. Functional Specifications giving the system requirements and objectives, the inputs, outputs and processes of each component, both computer and manual.

. System structure giving the sub-systems, programs and program components.

. System data flow

. User Aid Design Specifications

. Hardware environment and requirements

Methods

. Summary of Steps
  - Determine the approach
  - Draw up test-case coverage matrix
  - Define test cases
  - Review test plan and test cases
  - Document the System Test

. Determine the Approach

  System testing is done to ensure that the system meets its requirements at all stages in its evolution. Therefore the levels of the system and relevant tests must first be defined. This requires categorization of the system testing into:

  - integration levels (linking of modules)
  - sub-system levels
  - full system test

Clear objectives must be drawn up in advance of the test case preparation from the Functional Specifications, for each level of test and for each function or process being tested. The objectives could include: (see The Art of Software Testing, referred to in Appendix A):

- completeness with which process is covered; both manual as well as computer
- volume testing; subjecting the system to large volumes
- stress testing; subjecting the system to peak volumes over a short span of time
- usability testing; attempting to cover the human factor side of the system
- security testing
- performance testing
- storage testing
- configuration testing
- compatibility/conversion testing; especially if the new system replaces an older one that interfaced with other systems
- installability testing
- reliability testing
- recovery testing
- servicability testing
- documentation testing, examples in the documentation should be used as in test cases
- procedures testing, any prescribed human operations should be tested.

. Draw Up Test-Case Coverage Matrix

A table must be drawn up listing all of the components and objectives of the system testing. This will be used later to ensure that the test cases will satisfy all of the requirements.

. Define Test Cases

At the system level, test data is generated from the sub-system specification and the program specifications in the system design specifications. This is known as black box testing since it is not required that the internal working of the module be known.

Black box testing involves creating test cases based on program specifications. Black box methods include:

. equivalence partitioning;
. boundary value analysis;
. cause effect graphing; and
. error guessing.

Equivalence Partitioning

This method involves analyzing the general specification of the program module and creating a list of all decisions the logic of the program should make, dependent upon the input data. Testing all combinations of data is impossible -- therefore the technique calls for grouping the data into classes where all elements within the class test the same decision. These are equivalence partitions, where the tester assumes (reservedly) that a test of one element in the class is equivalent to a test of any other value. Consider this example:

The specification reads:

BEE codes are 2 characters alphanumeric and must exist on the HONEY table. Both characters being blank is invalid.

The following pages show the breakdown of these simple specs into equivalence classes. There must exist test cases that exercise each valid equivalence class and each invalid equivalence class.
The valid equivalence class:

"1st numeric 2nd blank"
contains these data elements:
0, 1, 2...9

It is only necessary to test one of these elements.

Similarly the invalid equivalence class
"both special character"
contains many elements some of which are
$$, !, $/, .@
Again it is only necessary to check one element.

Test Cases	Valid Equivalence Classes	Invalid Equivalence Classes
Number of characters	(one two)	three
Field Format	both alphabetic	both blank
	both numeric	both special character
	1st numeric 2nd blank	1st special 2nd numeric
	1st blank 2nd numeric	1st special 2nd alphabetic
	1st numeric 2nd alpha	1st numeric 2nd special
	1st alpha 2nd blank	1st alphabetic 2nd special
	1st blank 2nd alpha	
	1st alpha 2nd numeric	
Table Consideration		table missing
	table empty	
	entry on table	entry not on table

This list shows 15 equivalence classes, the item in ( ) will be checked within the field format classes. Any element within a class is representative of the entire class.

Boundary Value Analysis

Test cases that explore the boundary condition have a higher payoff of finding errors than cases that do not. Boundary conditions are those situations in the neighbourhood of the edge of input and output equivalence classes.

A boundary condition is a condition which causes different processing to be executed. Some examples of boundary conditions are:

- minimum and maximum ranges
- end page processing
- end of file processing
- file full/file empty
- duplicate key
- invalid key
- key missing
- sequence checking
- table empty
- table missing

In this method:

- each edge of the equivalence class is the subject of the test;

- test cases are constructed for input conditions and output conditions.

Example of Boundary Value Analysis

The day code will be 2 digits, numeric in the range of 01 to 31.

TEST CASES

VALID	INVALID
01	00
31	32
	both alphabetic
	1st alphabetic 2nd numeric
	2nd alphabetic 1st numeric

This shows 7 test cases to be input.

Cause-Effect Graphing

- divide specifications into workable pieces;

- identify causes and effects in the specification (cause is a distinct input condition, effect is an output condition);

- link the causes and effects into a Boolean graph (cause-effect graph);

- convert graph into limited entry decision table; and

- convert decision table into test cases.

This method is complex, meticulous and time consuming. The technique will not be described fully here. It is to be noted that this technique can be automated and a number of computer packages are available for performing cause-effect graphing.

More information can be found in "The Art of Software Testing" by G.J. Myers.

Error Guessing

The method consists of:

. listing possible errors or error prone situations based on specifications; and

. creating test cases to expose these errors.

The test case design methods discussed in this section must be combined to produce an overall testing strategy. The reason for combining them is that each contributes a set of useful test cases but none alone provides a thorough set of test cases. A reasonable strategy is:

- start with cause-effect graphing if the specification contains a combination of input conditions at the design specification level;

- use boundary analysis. Be sure to include both input and output boundaries. This yields a set of supplemental test conditions many or all of which can be incorporated into the cause-effect test;

- identify valid/invalid equivalence classes for input/output and if necessary supplement the test cases identified above;

- add additional test cases by using error guessing; and

- examine the program logic with regard to the test cases. By using this "white box" testing method, determine if the test cases cover all the logic conditions. If all conditions are not tested, add sufficient test cases to cause the criterion to be satisfied.

This strategy does not guarantee that all errors will be found. It does represent a considerable amount of hard work, but no one has claimed that program testing was easy.

. Review Test Plan and Test Cases

The test plan should be reviewed in walkthroughs to identify and correct errors and omissions.

The test cases should be reviewed both internally with the project team, and with the users for:

- completeness, all functions and requirements are covered; the test coverage matrix should be used to verify this; and

- validity of test cases; the user can verify that the test cases are valid in terms of the business procedures of the organization. Often, the user can provide typical processes with which the test cases can be checked.

. Document the System Test

Having considered and developed a system test, document it as a specification made up of three components:

- test plan;
- test case specifications; and
- performance evaluation criteria.

Working Documents

Walkthrough Notes on Test Cases

Deliverables

System Test Specifications
System Test Plan
Test Case Specifications
Performance Evaluation Criteria

4.8 Design Conversion Procedures (4.B.6)

Objectives

If there are existing manual or computer systems which will be replaced by the system under development, then it is necessary to design procedures for smoothly converting to the new system. A smooth conversion is important from the user's perspective and therefore care must be taken during this activity to consider when, during implementation, the conversion will take place, how the conversion will proceed and who will be responsible for the various components of the conversion.

Inputs

- Functional Specifications giving the system requirements and objectives, the inputs, outputs and processes of each component, both computer and manual.
- System Structure giving the sub-systems, programs and program components.
- Existing file descriptions from existing System Specifications.
- Component Data Flow Descriptions.
- Data Dictionary.
- Hardware Configuration Descriptions - existing and requirements.

Methods

- Summary of Steps

  - Determine conversion requirements
  - Design conversion procedures
  - Develop conversion strategy
  - Package conversion plan

- Determine Conversion Requirements

  During this step it is necessary to compare in detail the before and after image of data stores and data flows, and to define what is required to transform the existing data store or data flow into the required format.

For manual data stores and data flows it is necessary to consider such things as: storage medium (e.g., paper vs microfiche or active vs archive) form changes (e.g., form re-design and printing) and retention or transmission changes. Similarly, for computer data stores and data flows, storage medium (e.g., disk vs tape) storage format (e.g., data element type and size or record layout) and transmission changes must be considered.

It may also be necessary to consider changes to existing processes. For example, manual processes may be modified, or replaced with new manual processes or computer processes. The required changes must be identified so that conversion procedures can be designed.

. Design Conversion Procedures

Once the required conversions have been identified it is necessary to determine how the changes are to be effected and by whom. It may be necessary to design computer modules to transform a computer file from one format or medium to another. These modules should be designed using the structured approach given in the preceding sections of this Guide. In other cases, it may be necessary to arrange for data to be transformed into machine-readable form, through hiring temporary data entry staff or contracting with a service bureau.

If procedures will be changed to effect conversion, a temporary training manual should be prepared. This will be used to inform the user group of the objectives and scope of the conversion, and to assist the users during the actual conversion.

. Develop Conversion Strategy

Once the various conversion procedures have been designed, it is necessary to consider when the conversion procedures will be implemented. This consideration of the time dimension in relation to the user's requirements should lead to the development of a conversion strategy. Four possible strategies to be considered, for converting from one system to another are:

- Parallel conversion - both systems run in parallel, with the outputs being compared item by item until all discrepancies are resolved.

- Pilot conversion - a stand-alone subset of the final system or if the same system is to be installed in a number of locations, one location, is installed. The resources of the project team can then be concentrated on that pilot system until the system proves to be satisfactory.

- Phased conversion - only certain sub-systems of the new system are installed. The basis of choice of the sub-systems could be on time cycles (end-of-month processing) or on functions (data capture) or on the department's organizational lines.

- Immediate conversion - if none of the other methods is suitable, the only alternative is to end the old system one day and begin the new one the next.

Once having established a strategy acceptable to user management, full system specifications covering the conversion will be required.

Comparison of System Conversion Methods

method	relative costs	user effort required	team effort required	impact of damage if failure occurs
PARALLEL	High	High	Low	Low
PILOT	Medium	Low	Medium	Medium
PHASED	Medium	Medium	Medium	Medium
IMMEDIATE	Low	Medium	Low*	High

* if successful, otherwise very high.

. Package Conversion Plan

The proposed conversion procedures and strategy should be documented and added to the EDP System Design Specifications (section 16). The conversion plan should provide a list of activities, detailing how conversion is to be effected, when it is to proceed and who is responsible for each of the activities.

Working Documents

. Conversion procedure descriptions
. Conversion strategy evaluation

Deliverables

Conversion Plan

4.9 Complete Design Phase (4.C - 4.F)

To prepare a plan for the final phases of the project and to package the design document for approval.

Inputs

- User Aid Design Specifications
- Program Module Specifications
- System Test Specifications - with associated test plan, test cases and performance evaluation criteria.

Methods

- Summary of steps
  - Develop plan for following phases;
  - Package System Design Specifications;
  - User verification;
  - Departmental approval.

- Develop Plan for Following Phases

  The detailed plan as prepared for the Development and Implementation phases has taken into account the completed systems design and testing plans. Plans for the tools, methodologies, standards, etc., to be used in the development phase must be prepared. These plans will shorten the learning curve of the development team and minimize the time required for initial start-up.

- Package System Design Specifications

  At this stage, the system design and the completed specifications for the technical environment, plus the development phase plan and schedule should be taken into account to update the Cost/Benefits of the new system. Any major changes are to be identified. Then, all the components of the System Specification are to be packaged in preparation for transition to the Development phase.

- User Verification

  As the Design phase progresses each of the final inputs, user procedures, turnaround times, and outputs are verified and approved by appointed user representatives. Project Management must ensure evidence of user approval is retained and ultimately submitted to the approval authority when obtaining end of phase sign-off. It is not necessary to submit specifications for the internal system architecture, to users.

. Departmental Approval

A submission to the approval authority is required to secure approval to proceed to the System Development phase. It is not necessary to submit the total set of deliverables for approval as they are at an inappropriate level of technical detail for the senior staff with the approval authority.

The strategy for obtaining approval for the Design deliverables is to submit the planning aspects and evidence that the affected parties have reviewed and approved the Design specifications. This submission then would contain the following components:

- Executive Summary
- Revised Cost/Benefit

Deliverables

System Design Specification

APPENDIX A

References

REFERENCES

DeMarco, Tom. Concise Notes on Software Engineering. New York; Yourdon, c1979.

DeMarco, Tom. Structured Analysis and System Specification. Chapter 25.

Gane, C., Sarson, T. Structured System Analysis: Tools and Techniques. Chapter 9.

Gildersleeve, Thomas R. Decision Tables and Their Practical Application in Data Processing. Englewood Cliffs, N.J., Prentice-Hall, c1970.

Myers, Glenford J. The Art of Software Testing. New York, Toronto, J. Wiley & Sons, c1979.

Myers, Glenford J. Composite/Structured Design. New York, Toronto, Van Nostrand Reinhold, c1978.

APPENDIX B

Walkthroughs

WALKTHROUGH

The walkthrough/inspection concept came from IBM's programming teams. It uses the theory that the Programmer is a part of the complete team and that the team (not just the Programmer) is responsible for each program. This is commonly known as egoless programming.

The walkthrough concept is basically an extension of the desk check process. During desk checking, the programmer examines his code to discover errors. During a walkthrough members of the team inspect the code in a systematic manner to find any errors.

Although the walkthrough concept was developed for inspecting programming output, and this description is in that context, it is equally applicable to the products of analysis, design and testing.

The objective of this process is to find errors in logic, in specifications, etc. An inspection also looks for errors in style such as readibility, efficiency, unreasonable specifications, etc. the purpose of the inspection is not to find fault with the originator of the product being inspected but to improve upon that product.

Also Refer to Datamation Oct. 1977.

Inspecting Software Design and Code
By M.E. Fagan.

Below is an outline of an inspection technique used on one project.

Inspection team consists of:

- Chairman, who coodinates and schedules the meetings, chairs the inspection, notes all errors, circulates the inspection report and follows up on the rework.

- Document creator, the person who has created the document, whether it be the program specification, design or code. It is his responsibility to have all documents circulated to the other members at least 24 hours before the inspection.

- Implementors, those who will be taking over responsibility for the document (e.g., designers who will receive the specifications, programmers who will receive the program design, etc.). There will normally be one or two people in this category.

Inspection process consists of:

- Preparation and distribution of the document to all members of the inspection team prior to the meeting by the document creator.

- Preparation by all of the inspection team members which involves going over the document in some depth before the meeting.

- Inspection of the document by the whole team in the meeting. As the objective is to find errors, discussion continues only until the point where an error is recognized. The aim of the inspection is only to find errors, so often the chairman must be firm in limiting discussion. The error is then noted by the chairman. At the end, the team decides if the document passes the inspection and if not, a date is set for further inspection.

- Circulation of the inspection report by the chairman within 24 hours of the conclusion of the inspection meeting.

- Rework by the document creator to correct the errors.

- Follow-up. If the number of errors is small, than the Chairman is responsible for verifying that all errors are redressed. If there are a large number of errors, the inspection cycle is repeated.

During a walkthrough of actual code, there are major areas where problems occur. These are:

DATA REFERENCE
DATA DECLARATION
COMPUTATION
COMPARISON
CONTROL FLOW
INTERFACES
INPUT/OUT