# Part 2 – Counting Practices

This page intentionally left blank.

# Part 2 Counting Practices

**Introduction**

Part 1 provides the function point analysis process for sizing functionality provided by software following the IFPUG Method as well as the detailed rules for identifying and counting data functions and transactional functions.

Part 2 provides detailed counting practices and enhanced examples to assist practitioners in understanding:

- Data, including a description of the various types of data, how to distinguish business and reference data from code data, and how to count code data
- Logical files, including a description of logical files and the process to correctly group data into logical files, to correctly count record element types, and to correctly count data element types. Many examples are provided.
- Shared data, including a description of shared data and the process to identify when shared data is a data function (internal logical file, external interface file) or transactional function (external input, external output, external inquiry). Many examples are provided.
- Post development activities, including descriptions of enhancement and maintenance and those activities recognized by the function point counting process. Tables are included to assist practitioners in determining whether an activity is considered to be part of the enhancement project, as defined in Part 1, or as a maintenance activity.

**Contents**

Part 2 includes the following chapters:

This page intentionally left blank.

# Code Data

**Introduction**

This chapter uses the concepts of functional and technical requirements (described in Part 1, and in "A Framework for Functional Sizing" [IFPUG, 2003]) to identify code data and determine how it should be treated.

This chapter addresses specifically a number of examples pertaining to code data. It is recognized that there may be other examples of code data and these may be addressed in a future release of the CPM.

**Contents**

This chapter includes the following sections:

## IFPUG and ISO

**IFPUG Strategic Direction**

IFPUG's method for function point analysis is an ISO standard and must be conformant to ISO/IEC 14143-1:1998. The method can measure "functional size" only and not "technical size". This does not mean that the technical size cannot, or should not, be measured, only that it must be clearly stated as a separate measure ("A Framework for Functional Sizing" [IFPUG, 2003]).

## "A Framework for Functional Sizing"

"A Framework for Functional Sizing" provides the basis for understanding functional sizing as it relates to requirements. The paper explores different types of requirements; these concepts provide the basis for sizing software in accordance with ISO/IEC 14143-1 and the IFPUG CPM.

The basis behind "A Framework for Functional Sizing" is that there may be multiple sizing methods for different purposes. The functional size could be measured using the IFPUG functional size measurement method for function point analysis, based on the functional user requirements. Other sizing measures can be used to size, for instance, technical requirements.

Both result in different size-measures representing different dimensions of software size: IFPUG-FP for functional size and any other for technical size. While these sizes cannot be added together because they represent different dimensions (like volume and temperature of a room), they can both be used in estimating the effort towards the development of an application or a system.

"A Framework for Functional Sizing" provides guidance in distinguishing between functional size and technical size.

## ISO/IEC 14143-1 - Definition of User Requirements

In 1998, the first ISO/IEC Functional Size Measurement standard was published. This standard defines the Functional Size as “a size of software derived by quantifying the Functional User Requirements”.

ISO/IEC 14143-1 defines the fundamental concepts of Functional Size Measurement (FSM) and describes the general principles for applying an FSM Method. It does NOT provide detailed rules on how to:

- measure Functional Size of software using a particular Method
- use the results obtained from a particular Method
- select a particular Method

The definition of FSM in ISO/IEC 14143-1 is applicable when determining if a method for sizing software is a Functional Size Measurement Method. It does not prevent the development of various methods, but rather provides a basis for assessing whether a particular method conforms to FSM.

ISO/IEC 14143-1 distinguishes between three categories of user requirements:

- Functional User Requirements
- Quality Requirements
- Technical Requirements

The ISO/IEC 14143-1 definitions are as follows:

## ISO/IEC 14143-1 - Definitions

| | |
|---|---|
| **Functional User Requirements** | A subset of the User Requirements, the Functional User Requirements represent the user practices and procedures that the software must perform to fulfill the users’ needs. They exclude Quality Requirements and any Technical Requirements. |
| **Quality Requirements** | Any requirements relating to software quality as defined in ISO 9126:1991. |
| **Technical Requirements** | Requirements are related to the technology and environment, for the development, maintenance, support and execution of the software.<br>NOTE – Examples of Technical Requirements include programming language, testing tools, operating systems, database technology and user interface technologies. |

According to ISO/IEC 14143-1, only Functional User Requirements are counted in a functional sizing method like FPA.

## Examples of Quality Requirements

The Quality Requirements as defined in ISO/IEC 9126:1991 describe the degree to which the functional or technical requirements are met.

Examples of uses of the quality model as defined in ISO/IEC 9126 are to:

- validate the completeness of a requirements definition
- identify software requirements
- identify software design objectives
- identify software testing objectives
- identify quality assurance criteria
- identify acceptance criteria for a completed software product

ISO/IEC 9126:1991 defines the following types of characteristics as part of the quality model:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

For a detailed description of those characteristics, please refer to ISO/IEC 9126.

## Types of Data Entities

A review of the application data and its purpose provides an understanding of the various categories of data entities. In general, one may distinguish between three categories of data entities:

- Business Data
- Reference Data
- Code Data

The first two categories of entities are usually identified to satisfy the *Functional User Requirements*, and as such these entities will be investigated for counting as logical files (see Part 2, Chapter 2).

The third category of data, further referred to in this chapter as "Code Data", however, usually exists to satisfy technical requirements rather than functional requirements. The different categories of data are outlined below to assist in identification.

### Business Data

Business Data may also be referred to as Core User Data or Business Objects. This type of data reflects the information needed to be stored and retrieved by the functional area addressed by the application. Business Data usually represents a significant percentage of the entities identified. It has most of the following characteristics:

#### Logical

Logical characteristics include:

- mandatory for the operation of the users' functional area
- user identifiable (usually by a business user)
- user maintainable (usually by a business user)
- stores the users' Core User Data to support business transactions
- very dynamic - normal business operations cause it to be regularly referenced and routinely added to, changed or deleted
- reported on

#### Physical

Physical characteristics include:

- has key fields and usually many attributes
- may have zero to infinity records

**Examples**

Examples of Business Data include:

- Customer File, Invoice File, Employee File, Job File
- Job File, within the Job Management System, would include items such as:
    - Job Number,
    - Job Name,
    - Division Name,
    - Job Initiation Date, etc.

## Reference Data

This type of data is stored to support the business rules for the maintenance of the Business Data; e.g., in a payroll application it would be the data stored on the government tax rates for each wage scale and the date the tax rate became effective. Reference Data usually represents a small percentage of entities identified. It has most of the following characteristics:

**Logical**

Logical characteristics include:

- mandatory for the operation of the users' functional area
- user identifiable (usually by a business user)
- usually user maintainable (usually by an administrative user)
- usually established when the application is first installed and maintained intermittently
- stores the data to support core user activities
- less dynamic – occasionally changes in response to changes in functional areas' environment, external functional processes and/or business rules
- transactions processing  Business Data often need to access Reference Data

**Physical**

Physical characteristics include:

- has key fields and few attributes
- usually at least one record or a limited number of records

**Examples**

Examples of Reference Data include:

- Jobs Rates, Discount Rates, Tax Rates, Threshold Settings
- Job Rates File – stores information about the rates paid for each type of job and the skill required to do that type of job
  - Job Type
  - State, Charge Rate, Effective Date (1:n)
  - Job Skill Description (1:n)

## Code Data

The user does not always directly specify Code Data, sometimes referred to as List Data or Translation Data. In other cases it is identified by the developer in response to one or more technical requirements of the user. Code Data provides a list of valid values that a descriptive attribute may have. Typically the attributes of the Code Data are Code, Description and/or other 'standard' attributes describing the code; e.g., standard abbreviation, effective date, termination date, audit trail data, etc.

When codes are used in the Business Data, it is necessary to have a means of translating to convert the code into something more recognizable to the user. In order to satisfy technical requirements, developers often create one or more tables containing the Code Data. Logically, the code and its related description have the same meaning. Without a description the code may not always be clearly understood.

The key difference between Code Data and Reference Data is:

- With Code Data, you can substitute one for the other without changing the meaning of the Business Data; e.g., Airport-Code versus Airport-Name, Color-Id versus Color-Description.
- With Reference Data, you cannot substitute (e.g., Tax Code with the Tax-Rate).

Code Data has most of the following characteristics:

## Logical

Logical Characteristics include:

- data is mandatory to the functional area but optionally stored as a data file
- not usually identified as part of the functional requirements; it is usually identified as part of design to meet technical requirements
- sometimes user maintainable (usually by a user support person)
- stores data to standardize and facilitate business activities and business transactions
- essentially static - only changes in response to changes in the way that the business operates
- business transactions access Code Data to improve ease of data entry, improve data consistency, ensure data integrity, etc.
- if recognized by the user:
  - is sometimes considered as a group of the same type of data
  - could be maintained using the same processing logic

## Physical

Physical characteristics include:

- consists of key field and usually one or two attributes only
- typically has a stable number of records
- can represent 50% of all entities in Third Normal Form
- sometimes de-normalized and placed in one physical table with other Code Data
- may be implemented in different ways (e.g., via separate
- application, data dictionary, or hard-coded within the software)

## Examples

Examples of Code data include:

- State
  - State Code
  - State Description
- Payment Type
  - Payment Type Code
  - Payment Description

## Origin of Code Data

Historically, the reason for Code Data was to save space by storing a code rather than a lengthy textual description. For ease of maintenance, these codes and descriptions were placed in files or tables to eliminate software changes when updates were necessary.

The Code Data is a property of a descriptive attribute, so called “Meta Data”. Examples are valid values, code-descriptions or translation tables. Some Code Data is developed to meet specific user requirements and contain data that is within the user’s domain. Other Code Data may be derived from user requirements to restrict the values allowed. Code Data may also be created in an attempt to reduce requirements for disk space. Requirements may also include the ability to maintain Code Data. All of these are technical requirements.

Code Data is an implementation of technical requirements. As a consequence, Code Data may influence the *technical* size of the software product, but *not* the *functional* size of the software product [“A Framework for Functional Sizing”, IFPUG 2003].

# Methodology

## Introduction

A stepwise process for identifying what is and what is not code data:

| Step | Action |
|---|---|
| 1 | **Filter out Code Data before Assessing Logical Files**<br><br>This chapter describes the data that is included as a response to technical requirements, **and must be filtered out** *before* identifying logical files. The next section explains the different types of entities and expands on the physical data that must be excluded from identifying logical files.<br><br>This kind of data that is **excluded** from identifying logical files is not considered part of the functional size. However, data that exists as part of technical requirements could be measured using a specific measure to size the technical size; refer to “A Framework for Functional Sizing”.<br><br>The first step “Identifying Entities that should be considered for counting”, as defined in Part 2 – Chapter 2, determines which candidate entities should be taken into consideration for the logical grouping of the entities and which should be excluded. |
| 2 | **Don’t Count Code Data as an FTR**<br><br>A consequence of filtering out Code Data is that Code Data cannot be considered an FTR while assessing the complexity of a transactional function (EI, EO, EQ) because it is not a logical file. |
| 3 | **Don't Count Code Data Transactional Functions**<br><br>A consequence of Code Data being a part of another dimension (the technical dimension as opposed to the functional dimension) is that Code Data maintenance or reporting functions are not taken into account when measuring the functional size of the application in function points. Of course, like Code Data, Code Data Transactional Functions could be measured using a specific measure to size the technical dimension (refer to “Counting Code Data and Code Data Transactions”). |

## Identifying Code Data

The Code Data types summarized in "What Is Code Data" and "What Is Not Code Data" may be used as a practical help in order to determine whether or not an entity is Code Data. Some criteria partly overlap. As soon as the criteria of one of the subsections have been satisfied, the entity should be considered Code Data.

The examples provided are not an exhaustive list and may not cover all possible cases. When in doubt, evaluate entity types within the context of "Types of Data Entities".

### What Is Code Data

**Introduction**

These are the several different types of Code Data, which fall into three general areas:

- Substitution Data provides a code and an explanatory name or description for an attribute of a business object (Substitution is a sufficient condition, but not a necessary condition to be considered Code Data).
- Static or Constant Data is data that rarely changes.
- Valid Values Data provides a list of available values for an attribute of one or more business object types.

**Types of Code Data**

| Substitution | Static or Constant | Valid Values |
|---|---|---|
| Code + Description | One Occurrence | Valid Values |
| | Static Data | Range of Valid Values |
| | Default Values | |

Any of these types of Code Data may also include other attributes, such as effective date and termination date to define the time period for which the value is available. It may also include audit type attributes of creation date, created by (user-id), last updated date, last updated by (user-id). Also, a number of variations are possible (e.g., code + short/long description). The presence of these additional attributes does not affect the categorization process, but the attributes are considered to be part of the Code Data.

## Substitution

**Code + Description**

This type of Code Data contains a code and an explanatory name or description. This type of Code Data may serve as quick data entry means for experienced users, the explanatory name / description for less experienced users or for listings as in reporting. This type of Code Data may also be implemented to save storage space or be a result of normalization.

**Examples**

- States: State Code, State Name
- Colors: Color Code, Color Description

**Variations**

- Code, Language, Description (for descriptions in multiple languages)
- Code, Short Description, Long Description, Abbreviation

## Static or Constant

**One Occurrence**

This type of Code Data contains one and only one occurrence regardless of the number of attributes. The Code Data has only one row of data and the attributes are relatively constant; it may change, but very rarely.

**Examples**

- An entity with data about a particular organization; e.g., name and address.
- COTS Software with airline name, customized by user organization

**Static Data**

This type of Code Data contains data that is basically static. The number of instances of static data may change, but very rarely, and the contents of an instance rarely change.

**Examples**

- An entity chemical elements: mnemonic, atomic number, description
- The function point tables for valuing function types and complexity levels

**Default Values (template)**

This type of Code Data contains default values for (some attributes in) new instances of a business object.

## Valid Values

**Valid Values**

This type of Code Data contains one attribute; it provides a list of valid values for an attribute of one or more business object types. This type of Code Data is implemented to satisfy requirements such as reducing errors and increasing user friendliness. This type of Code Data is typically used to list available values for user selection and/or validating input provided by the user.

**Examples**

- State name: contains all valid values for the attribute state name
- State code: contains all valid values for the attribute state code
- Color: contains all valid values for the attribute color of a business object

**Range of Valid Values**

This type of Code Data contains data that is basically static; if it is not, it may be Reference Data.

**Examples**

- Allowable Telephone Number Ranges: lowest telephone number, highest telephone number.
- Heat temperature range.

## What Is Not Code Data

This section describes data that is not considered Code Data because it is either Business Data or Reference Data. CPM Part 1 and Part 2 – Chapter 2 (Logical Files) contain the rules for these types of entities. Sometimes tables are called 'code tables' but are in reality Reference Data or even Business Data.

### Examples

Examples of Business or Reference Data that should not be considered Code Data:

- Entity types with financial amounts, exchange rates, and tax rates, if they are not constants. This data does not restrict valid values; rather, it adds meaning to a value within a particular range.
- Control data: User-maintained data that contains business rules telling the application what to do or how to behave.
- Currency Exchange Rate Table: Exchange rates contain country and current exchange rate to US dollars. It is not possible to substitute the code for exchange rate of the country, the data is essentially not static, and the data supports business activities; therefore, this is an example of Reference Data.
- Tax Rate Ranges for a Progressive Tax System: A different tax rate is applicable for different ranges of income. This contains minimum and maximum values for each tax rate. However, you cannot substitute the value for the entity, and the data supports business activities. The tax rate does not restrict the income range. Therefore, this is an example of Reference Data.

## Counting Code Data and Code Data Transactions

The Code Data as identified in “What Is Code Data” represents the implementation of technical requirements rather than the implementation of a Functional User Requirement. As a consequence, as stated in Section “IFPUG and ISO”, Code Data doesn’t count towards the functional size of an application.

At this moment there are no separate methods available to count Code Data and their maintenance and reporting functions to yield a size of the technical dimension.

## Works Cited, Works Consulted

The following sources were consulted or cited in this chapter:

Definitions and Counting Guidelines for the Application of Function Point Analysis: A Practical Manual, Version 2.0. (English), NESMA, 1997.

ISBN: 90-76258-02-3.

Note: This manual is also called NESMA Counting Practices Manual. It describes the standard FPA methodology, and many aspects related to the application of FPA. It may be used together with the IFPUG manual. For more information, see the NESMA web site www.nesma.org.

IFPUG "A Framework for Functional Sizing", IFPUG, 2003.

ISO/IEC 14143-1:1998 Information technology – Software measurement Functional size measurement – Part 1: definition of concepts, ISO/IEC, 1998

ISO/IEC 9126-1:1991 Information technology – Software product evaluation – Quality characteristics and guidelines for their use, ISO/IEC, 1991

2

# Logical Files

**Introduction**

This chapter applies the definitions and rules pertaining to logical files, using a descriptive process for the identification and classification of data functions.

These guidelines illustrate the identification of data functions from normalized data models. An overview of data normalization is provided to support this approach. However, this does not preclude the use of these guidelines in environments where alternative data or object modeling techniques are employed.

**Contents**

This chapter includes the following sections:

# Outline of the Methodology

## Introduction

A stepwise process for establishing the set of logical files (Internal Logical Files and External Interface Files) is used where each step looks at the data at a finer level of detail. Details of each step are explained in the subsequent sections.

| Step | Action |
|---|---|
| 1 | **Filter Out Code Data Before Assessing Logical Files**<br><br>Data that is included as a response to quality requirements from the user, for physical implementation and/or technical reasons, **must be filtered out** *before* identifying logical files.<br>This step is explained in Part 2 - Chapter 1 “Code Data”. |
| 2 | **Identify Logical Files and Classify**<br><br>For each logical data entity, identify how related entities are to be grouped into logical files, which reflect the ‘user view’. E.g., determine whether data entities are themselves an independent logical file or whether related entities should be grouped into a single logical file. After identification each identified logical file is classified as ILF or EIF.<br>This step is explained in the Section "Identifying Logical Files and Classifying". |
| 3 | **Identify Record Element Types (RETs)**<br><br>For each logical file, identify how related data are to be grouped into **record element types**, which reflect the ‘user view’.<br>This step is explained in the Section "Identifying Record Element Types". |
| 4 | **Identify Data Elements (DETs)**<br><br>For each logical file, identify the **data elements** used by the application being counted.<br>This step is explained in the Section "Identifying Data Element Types". |

The steps with the most impact on the counting results are step 1 and 2, assessing and identifying logical files, because the identification of the right *number* of logical files is decisive in achieving inter-counter consistency. Identifying RETs (step 3) and identifying DETs (step 4) influence the outcome of a function point count to a considerable lesser degree than step 1 and 2 because they don't affect the number of logical files, only their complexity rating.

For this reason, the most important part of this chapter is step 2, identifying logical files. Step 1 has already been described in Chapter 1 – Code Data.

# Data Modeling Concepts

## Introduction

A review of the definitions used in the domain of data analysis (which includes logical and physical data modeling) can provide both a foundation of understanding as well as a clarification to the intent of the counting practices rules as they relate to the identification of Logical Files, Record Element Types and Data Element Types. A strong understanding of data concepts is implicit in Function Point Analysis when appropriately and correctly counting the Data Functions. Section “Data Modeling Terms” summarizes the data modeling terms.

## Key Concepts in Data Modeling

A review of the definitions used in the domain of data analysis (which includes data modeling and DBMS) can provide both a foundation of understanding as well as a clarification to the intent of the counting practices rules as they relate specifically to Logical Files, Record Element Types (RETs) and Data Element Types (DETs). An understanding of data concepts is implicit to apply the guidelines as described in the next chapters to appropriately and correctly count the Data Functions.

## Entity (type)

**Definitions of Entity (or Entity Type)**

- Any distinguishable person, place, thing, event or concept about which information is kept (Thomas Bruce, 1992).
- A thing that can be distinctly identified (Peter Chen, 1976).
- Any distinguishable object that is to be represented in the database (C.J. Date, 1986)
- A data entity represents some "thing" that is to be stored for later reference. The term **entity** refers to the **logical** representation of the data (Clive Finkelstein, 1989).
- The word entity means anything about which we store information (e.g. a customer, supplier, machine tool, employee, utility pole, airline seat, etc.). For each entity, certain attributes are stored (James Martin, 1989).
- An entity may also represent the relationship between two or more entities, called **associative entities** (Michael Reingruber, 1994).
- An entity may represent a subset of information relevant to an instance of an entity, called **subtype entity** (also known as secondary or category entity) (Michael Reingruber, 1994)

To summarize, an entity:

- is a principal data object about which information is collected
- is a person, place, thing or event of information
- can have an instance (an occurrence)
- is a fundamental thing of relevance to the user, about which a collection of facts is kept; an association between entities that contains attributes is itself an entity
- involves information, a representation of similar things that share characteristics or properties
- often depicted in a data model through a rectangle, with entity name written inside the rectangle

### Data Element

In the world of data modeling, the base element is called the ***data element*** or ***data item.*** It is

- the fundamental component
- the fundamental atomic particle in the information system universe (Gary Schutt)
- the smallest names unit of data that has meaning in the real/user world (Graeme Simsion).

## Data Modeling Activities

Data modeling addresses data items, logical records and files. A **File System** is composed of records and data items. **Data items** are defined as the smallest named unit of data that has meaning to the real world. A group of related items that is treated as a unit is known as a **record**. A **file** is a collection of records of a single type.

In the physical implementation of data through relational databases, the following terms are used: a data item is called an "attribute" or "column", a record is called a "row" or "tuple", and a file is called a "table". These terms do not change the base meaning of the concepts.

## Mapping Data Concepts to Function Point Terminology

We can further map these terms to function point analysis as shown in the following matrix:

| Data Modeling Concept | Data Modeling Term | Relational Database Term | FPA Term | FPA Concept |
|---|---|---|---|---|
| Smallest named unit of data that has meaning to the real world | Data Item | Attribute or Column | Data Element Type (DET) | A data element type (DET) is a unique, user recognizable non-repeated field |
| Groups of related items which are treated as a unit | Record | Row or Tuple | Record Element Type (RET) | A record element type (RET) is a user recognizable subgroup of data elements within an ILF or EIF |
| Collection of records of a single type | File | Table | Logical File (Internal Logical File - ILF or External Interface File - EIF) | File refers to a logically related group of data and not the physical implementation of those groups of data |

Once all required data has been identified, the data analyst applies various normalization rules to depict the data in various Entity Relationship Diagrams. A summary of normalization rules can be found in Section "Data Modeling Terms".

## Entity Relationship Concepts

Once all required data has been identified, the data analyst applies various normalization rules to depict the data in various Entity Relationship Diagrams. The following table can be applied to further understand the concept of Record Element Types.

| **Entity Relationship Concept** | **E-R Term** | **FPA Term** | **Part 1 Definition** |
|---|---|---|---|
| Principal data objects about which information is collected (person, place, thing or event); a fundamental item of relevance to the user about which a collection of facts is kept | Entity or Entity Type | Internal Logical File (ILF) or External Interface File (EIF) | File refers to a logically related group of data and not the physical implementation of those groups of data |
| An entity type that contains attributes which further describe relationships between other entities | Associative Entity Type | Optional or Mandatory subgroup | User recognizable subgroup of data elements within an ILF or EIF |
| An entity type that further describes one or more characteristics of another entity type | Attributive Entity Type | Optional or Mandatory subgroup | User recognizable subgroup of data elements within an ILF or EIF |
| A division of an entity type, which inherits all the attributes and relationships of its parent entity type; may have additional, unique attributes and relationships | Entity Subtype | Optional or Mandatory subgroup | User recognizable subgroup of data elements within an ILF or EIF |

## Data Modeling Terms

### Entity (or Entity Type)

- Principal data objects about which information is collected
- Person, place, thing or event of information
- Entity instance (an occurrence)
- Depicted through a rectangle, with entity name written inside the rectangle
- A fundamental thing of relevance to the user, about which a collection of facts is kept. An association between entities that contains attributes is itself an entity.

### Associative Entity Type

- An entity type that contains attributes which further describe a many-to-many relationship between two other entity types.

**Attributive Entity Type**

- An entity type that further describes one or more characteristics of another entity type.

**Entity Subtype**

- A subdivision of entity type. A subtype inherits all the attributes and relationships of its parent entity type, and may have additional, unique attributes and relationships

**Relationships**

- Represent real-world associations among one or more entities
- One-to-One
- One-to-Many
- Many-to-Many
- Represented by a line which connects the entities.
- The relationship name is written besides the line

Relationships are defined by the way the entities are connected:

- Optional, shown in text with parentheses 1:(N), (1):(N)
- Mandatory, shown in text without parentheses 1:1, 1:N

**Attributes**

- A characteristic of an entity. Attributes are generally analogous to Data Element Types (DETs).

**Normalization**

Data is normalized through 5 rules

1. Eliminate Repeating Groups (1st Normal Form)
2. Eliminate Redundant Data (2nd Normal Form)
3. Eliminate Columns not dependent on Key (3rd Normal Form)
4. Isolate Independent multiple Relationships (no table may contain two or more 1:N or N:M relationships – fourth normal form)
5. Isolate semantically related multiple relationships (practical constraints may justify separating logically related many to many relationships – 5th normal form)

Look at the logical data model, preferably 3rd normal form.
Ignore multiple entities placed for technology (often 5th normal form)

# Identifying Logical Files and Classifying

## Background

In FPA, a logical file is a logical group of data as seen by the user. A logical file is made up of one or more data entities. This chapter provides guidelines on how to group the candidate entities identified into one or many logical files.

The process consists of the following steps, all of which are explained in detail in the following paragraphs of this section:

| Step | Action |
|---|---|
| 1 | Identify the entities that should be considered for counting (page 2-10). |
| 2 | Identify the user view (= business view) of the data grouping by investigating:<br>a) How the data is accessed as a group by elementary processes within the application boundary (page 2-12)<br>b) The relationships between the entities and their interdependency based on business rules (page 2-13 through 2-22). |
| 3 | Classify each identified logical file as ILF or EIF (page 2-23). |

The most difficult step is the grouping of data (step 2). The final grouping of data into logical files is the result of the combined effect of two grouping methods:
- Method a) is process driven, based on the user transactions in the application.
- Method b) is data driven, based on the business rules.

However, because the user transactions are (or should be) based on the business rules as well, both methods support each other. This double approach may disclose eventual shortcomings in the functional specifications and makes the identification process of logical files reliable and repeatable.

## Identifying Entities that Should be Considered for Counting (Step 1)

Before deciding which entities should be grouped together into logical files, one should determine which candidate entities should be taken into consideration for the logical grouping of the entities (step 2) and which should be excluded.

The general guiding principle of Part 1 – Count Data Functions is clear: only consider entities that are significant to and required by the user.

The following steps will help to identify these entities in a repeatable way:

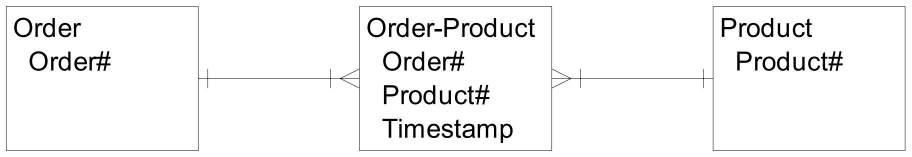
**1.1** Determine which entities don't contain user-recognizable attributes required by the user; but contain technical attributes only. Examples of these technical attributes are attributes that exist as a result of a design or implementation consideration. E.g., index files created for performance reasons such as the alternate index (see Part 1 – Count Data Functions). Exclude these entities from further consideration; they are not counted as a logical file or RET.

**1.2** Determine which entities are associative entities. An associative entity contains the foreign keys of the connected entities in addition to other attributes.
Note that two situations can arise here as a result:

a. The additional non-key attributes are a result of a design or implementation consideration or to satisfy a technical requirement (not required by the user; e.g., a date/time stamp for data recovery purposes). These technical attributes are not counted as data elements.
Treat these entities as key-to-key entities (see step 1.3).

b. The additional non-key attributes are necessary to satisfy the functional user requirements and are required by the user.
These entities are assessed in the next sections: identifying Logical Files (Step 2A/2B).

**Example:**



*Timestamp is usually a technical attribute not recognized by the user. In that case, for the Order-Product entity, guideline 2a applies. The key-to-key entity is resolved by adding the Product# as a foreign key to Order and the Order# to Product (step 1.3).*

**1.3** Determine which entities are a key-to-key (intersection) entity; i.e., they only have keys as their data elements and do not have any other non-key attributes. These entities typically represent the implementation of a many to many (**N:M**) relationship in a normalized data model. They exist for data modeling and data base design purposes only and not as the result of a user requirement. Exclude these entities from further consideration; they are not counted as a logical file or RET. According to the rules (Part 1), the referring attribute (foreign key) is counted as a data element for both entities connected by this key-to-key entity. See also the guidelines in Section "Identifying Data Element Types".

**1.4** Pay special attention when identifying logical files from a (normalized) data model:

- Do not assume that all entities are logical files; e.g., index files, entities on a physical data model.
- Logical files may exist from a user perspective, but may not be identified in the (normalized) data model in some cases; e.g., historical files containing aggregated data. Do not forget to include these logical files in the rest of the process.

**1.5** Look for entities that are not related to any identified business entity. They tend to be technical rather than functional data. If so, exclude these entities from further consideration.

**1.6** Determine which entities are not maintained by an elementary process in this or another application. Exclude these entities from further consideration because they are not counted.

**1.7** Check if all remaining entities are a result of functional user requirements. These entities, and the relationships and interdependencies between these entities, will be assessed in the next step: identifying Logical Files (Step 2A/2B).

## Identifying Logical Files Using Elementary Process Method (Step 2A)

The user's business view of the data is reflected in how the user transactions access the data.

Review how the elementary processes within the application boundary maintain the entities. If several entities are always *created* together and *deleted* together then this is a strong indication that they should be grouped into a single logical file. Also review the elementary processes used to extract the data to determine if the extraction processes access the same group of entities. Note: transactions which modify data will often just target one entity in the group, so the modify transactions are not as effective guidance for grouping data as the create and delete transactions.

**Example**

A customer purchase order is a single group of data from a user business perspective, it is made up of the Order Header (customer, address, date etc) and the details about each item ordered. From a business perspective an order cannot be created unless it has at least one item and if the order is deleted both the header and all items are deleted. However the header and items may have independent maintenance transactions. E.g., change order status, is a different function to changing Items ordered. The *create* and *delete* functions indicate from a user perspective that 'order' is a single logical file which has grouped the order header and order items.

Use Step 2B to validate the identified candidate logical data groups.

## Identifying Logical Files Using Entity (In-)Dependency Method (Step 2B)

### Introduction

The Entity (In-)Dependency Method, as defined and explained in this section, assists in correctly and in a repeatable way identifying logical files from a data model. The term "entity" in this section refers to an entity in a data model in a normalized form (usually third normal form).

Section "Types of Relationships"explains the different types of relationships, and the difference between the concepts "mandatory/optional relationship" and "entity dependence/independence".

Section "Entity (In-)Dependence Illustrated for All Types of Relationships" explains the method in more detail for each type of relationship.

Section "Summary: from Entities to Logical Files via Entity (In-)Dependence" summarizes the types of relationships and the conditions when to count a LF.

The Entity (In-)Dependency Method groups entities by assessing the relationships and interdependencies of the entities against the business rules. The guiding principles are *entity independence* and *entity dependence*.

### Entity In-dependence

*Entity independence* means that an entity is meaningful, has significance to the business in and of itself without the presence of other entities.

### Entity Dependence

*Entity dependence* means that an entity is *not* meaningful, has no significance to the business in and of itself without the presence of other entities.

### Example

Consider two entities A and B that are linked by a relationship.

(1) If B is significant to the business apart from A, then B is *entity independent* of A, and A and B should not be grouped into one logical file.

(2) If B is not significant to the business apart from A, then B is *entity dependent* on A, and A and B should be grouped into one logical file.

### Note

Don't confuse the concept of *entity independence/entity dependence* with the concept of *optional / mandatory relationship*. The examples in Section "Entity (In-)Dependence Illustrated for All Types of Relationships" clearly show that these are different concepts.

**Determine**

To determine whether entity B is dependent on or independent of A, one needs to determine:
"*Is B significant to the business apart from the occurrence of A linked to it*?"

An easy test to determine the situation (entity dependency or independence) is the following.
Even if there is no user requirement for deletion, (still) ask yourself the question:
*"Suppose we would want to delete an occurrence "a" of entity A, what would happen to an occurrence "b" of entity B linked to "a"?"*

Depending on the business rules two essentially different situations are distinguished:

**Situation 1**

If an occurrence of B according to the business rules does not have independent meaning/significance to the user and might be deleted as well, then apparently the occurrence of B does not have meaning to the user apart from the linked occurrence of A. Entity B is considered to be entity dependent on A.
The entities A and B should be grouped together into one logical file.

**Situation 2**

If the occurrence of B does have meaning to the business even apart from the linked occurrence of A, the business rules will not allow deleting the occurrence of B.
The entities A and B are considered to be separate logical files.

Assessing the data model of an information system by assessing all pairs of linked entities results in the identification of the logical files.

In Section "Entity (In-)Dependence Illustrated for All Types of Relationships" this method is explained in more detail for different types of relationships.
Section "Summary: from Entities to Logical Files via Entity (In-)Dependence" summarizes per relationship type how to count it within FPA.

## Types of Relationships

Before making the Entity (In-)Dependence principle conclusive for all types of relationships, one should clearly understand the different types/natures of relationship. This section explains the different types as well as the concepts "optional" and "mandatory".

### Example

Two entities, Job and Employee for example, can be connected to each other via a relationship; e.g., "employs".

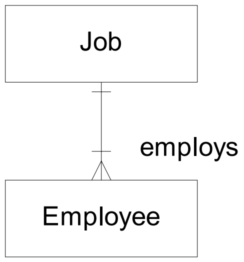
### Nature of Relationship

The nature of the relationship determines how many employees can work on a job according to the data model (0, 1, or more) and how many jobs a single employee can work on (0, 1, or more).

### 1 : N

Assume that the business rule states that several employees can be used for a job (but at least one), and that a single employee must work on one (and only one) job. In such a case we say that the relationship between Job and Employee is 1:N.

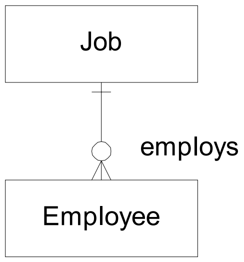
Graphically we can show this as in the figure aside.



### 1 : (N)

More likely the business rules state that a job *can* have a vacancy, i.e., no employee(s) is (are) assigned to the job. In this case the relationship between Job and Employee is optional and denoted as 1:(N).

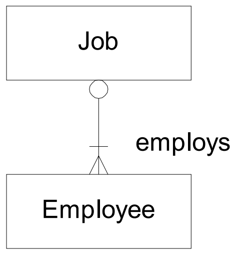
Graphically we can show this as in the figure aside.



### (1) : N

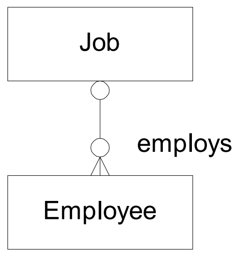
If the business rules state that an employee can exist without a job, but a job always must have an employee assigned to it, we denote the relationship between Job and Employee as (1):N.

Graphically we can show this as in the figure aside.



**(1) : (N)** In the situation where a job can have vacancies and an employee can exist without a job, both sides of the relationship are optional. The relationship between Job and Employee is denoted as (1):(N).

Graphically we can show this as in the figure aside.



**Concept "Mandatory/Optional" versus "Entity (In)Dependence".**

To clarify the difference between the concepts "mandatory/optional relationship" and "entity dependence/independence" assume, as an example, the following extension of the business rules "employee(s) is (are) not allowed without a job" (relationship types 1:N and 1:(N)).

When a job becomes obsolete, this does not mean that employees are no longer of significance to the business. An employee has significance to the business apart from the linked job. Employee is entity independent of job. Because of the mandatory relationship with job, all employee(s) must be reassigned to a new job, before job can be deleted.

So it can happen that an occurrence of entity B (e.g., Employee) may have a mandatory link to *an* occurrence of entity A (e.g., Job) in the relationship A:B between entities A and B, but that entity B does have significant meaning of its own to the business. In such a case, when one would want to delete an occurrence of entity A, one must first reassign a linked occurrence of B to another occurrence of A.

## Entity (In-)Dependence Illustrated for All Types of Relationships

**Entity (In-)Dependence in a (1):(N) Relationship**

**(1) : (N)** If a relationship between two entities A and B is bilaterally optional, the entities can exist independently and (all occurrences of) A and B are significant to the business apart from the linked occurrence(s) of the other entity.

Therefore, A and B are considered *entity independent* of each other. FPA counts the entities A and B as two separate logical files as indicated in the table in Section "Summary: from Entities to Logical Files via Entity (In-)Dependence".

**Entity (In-)Dependence in a 1:(N) Relationship**

**1 : (N)** In a 1:(N) relationship between two entities A and B (see figure 1), an occurrence of entity A may exist to which none, one or many occurrences of entity B are linked. On the other hand, each occurrence of B must be assigned to an occurrence of A.

**Example** In the 1:(N) relationship between Employee and Child (or Dependent) in an HR-application, an Employee may have 0, 1 or many Children linked to it, but a Child must be linked to one (and only one) Employee (see figure 2).

Because B must be linked to an A, this raises the question whether B is dependent on or independent of A.

To determine whether entity B is dependent on or independent of A, one needs to answer:
"*Is B significant to the business apart from the A linked to it*?"

The following is an easy test to differentiate between entity dependence and independence.
Even if there is no user requirement for deletion, ask yourself the question:
*"Suppose we would want to delete an occurrence of entity A, what would happen to the linked occurrences of entity B which has a mandatory link to an occurrence of A?"*

The business rules can result in two possibilities:

**Situation 1** If the deletion of A is allowed all Bs linked to it must be deleted as well, because the business is no longer interested in the occurrences of B. For example (see figure 2): An HR application maintains information about employees and their children. Assume that the business rule states that when an Employee (A) leaves the company, there is no significance for the business to keep the information about the Children (B).

**Situation 2** The deletion of A is not allowed as long as Bs are still linked to it, because the business is still interested in the Bs, even beyond the context of the linked A. For example (see figure 3): An organization adopts children and assigns each child to an employee. The employee is the contact person between the company and the child. In the case when an employee leaves the company, information about the associated children (of the terminated employee) is still significant to the business. So, before you are allowed to delete the Employee (A), you will first have to assign an associated Child (B) to another Employee (A) (because the nature of the relationship does not allow a Child without a linked Employee).

In situation (1) we say that B is *entity dependent* on A, and in situation (2) that B is *entity independent* of A.

FPA counts the entities A and B as one logical file in situation (1) (*dependence*), whereas in situation (2) A and B are each a separate logical file (*independence*) as indicated in the table in Section "Summary: from Entities to Logical Files via Entity (In-)Dependence".

Illustration of 1:(N) relationships:

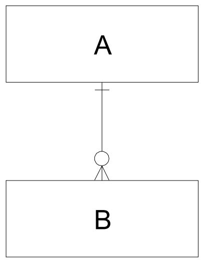


Fig. 1:
Each entity of type A may be linked to 0, 1 or many entities of type B. An entity of type B must be linked to exactly one entity of type A.

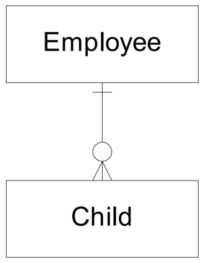


Fig. 2:
HR application maintains information about Employees and their Children.

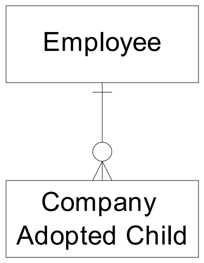


Fig. 3:
HR application maintains information about Employees and about company adopted Children who are assigned to an Employee.

Figures 2 and 3 have similar data models, but different business rules result in different logical files identified.

## Entity (In-)Dependence in a (1):N Relationship

**(1) : N**

A (1):N relationship between two entities A and B (see figure 4) can be treated in a similar way. These kinds of relationships, however, seldom appear in practice.

In a (1):N relationship between two entities A and B, each A must be assigned 1 or many Bs. On the other hand, a B may (but need not) be assigned to one occurrence of A.

**Example**

In the (1):N relationship between Committee and Organization Member, a Committee must have members (at least 1). An organizational member may (but need not) serve on a Committee (see figure 5 and 6).

Because an occurrence of A must be linked to a B, this raises the question whether A is dependent on, or independent of B.

To determine whether entity A is dependent on or independent of B, one needs to answer:
"*Is A significant to the business apart from the B linked to it*?"

The following is an easy test to differentiate between entity dependence and independence.
Even if there is no user requirement for deletion, ask yourself the question:
*"Assume we have an occurrence of entity A to which one of more occurrences of entity B are linked. Suppose we would want to delete the final linked occurrence of entity B, what would happen to that occurrence of A which has a mandatory link to at least one occurrence of B?"*

The business rules can result in two possibilities:

**Situation 1**

When the last B is deleted, the linked A is also deleted because the business is no longer interested in that A. For example (see figure 5): An organization has committees to which members are assigned.

The business rule is that a committee *must* have members, but not all members need to participate in a committee. An additional business rule is that the organization disbands a committee as soon as no members participate in it; one could say that committees are seen as ad hoc working groups.

In this case when the last member of a specific committee leaves, there is *no* significance to the business to keep information about the committee. The committee data is deleted as soon as the last member leaves the committee.

**Situation 2** The deletion of the last B is *not* allowed as long as an A is still linked to it, because the business is still interested in that specific A, even beyond the context of the linked Bs. For example (see figure 6), an organization has committees to which members are assigned.

The business rule is that a committee *must* have members but not all members need to participate in a committee. Committees are seen as part of the organizational structure. They have meaning to the business beyond the members serving on it.

Before the last member of a specific committee leaves, a new member must be assigned to that committee because the nature of the relationship does not allow a committee without members.

In situation (1), A is apparently *not* significant to the business unless it is related to one or more Bs, whereas in situation (2) it *is* significant.

In situation (1) we say that A is *entity dependent* on B and in situation (2) that A is *entity independent* of B.

FPA counts the entities A and B as one logical file in situation (1) (*dependence*), whereas in situation (2) A and B are each a separate logical file (*independence*), as indicated in the table in Section "Summary: from Entities to Logical Files via Entity (In-)Dependence".

Illustration of (1):N relationships:

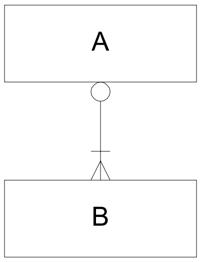


Fig. 4:
Each entity of type A must be linked to1 or many entities of type B; an entity of type B may, but need not, be linked to an entity of type A.

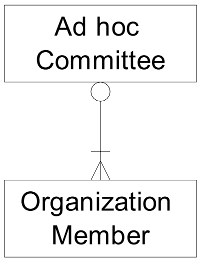


Fig. 5:
Members of an organization may (but need not) be active in a working committee. A Committee must have (one or more) members participating.

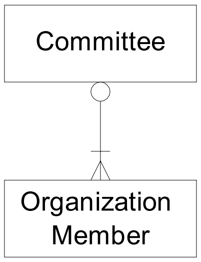
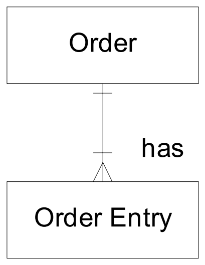


Fig. 6:
Members of an organization may (but need not) be active in a working committee. A Committee must have (one or more) members participating.

Figures 5 and 6 have similar data models, but different business rules result in different logical files identified.

## Entity (In-)Dependence in a 1:N Relationship

**1 : N** In a 1:N relationship between two entities A and B, each entity B must be assigned to one and only one A, and each A must be assigned to at least one B. The same guidelines regarding entity dependence and independence apply.



**Situation 1** If B is *not* significant to the business apart from the A linked to it, then B is considered to be entity dependent on A.

**Situation 2** If B *is* significant to the business apart from the A linked to it, then it is considered to be entity independent of A.

FPA counts the entities A and B as one logical file in situation (1) (*dependence*), whereas in situation (2) A and B are each a separate logical file (*independence*), as indicated in the table in Section "Summary: from Entities to Logical Files via Entity (In-)Dependence".

## Summary: from Entities to Logical Files via Entity (In-)Dependence

In the table below, A and B are two entities from a (normalized) data model that should be counted according to Section "Identifying Entities that Should be Considered for Counting", and that are interconnected via a relationship. The table summarizes how occurring situations are counted.

| Type of relationship between two entities, A and B | When this Condition Exists | Then Count as LF |
|---|---|---|
| (1) : (N) | (A and B are independent) | 2 LFs |
| 1 : N | If B is entity dependent on A | 1 LF |
| | If B is entity independent of A | 2 LFs |
| 1 : (N) | If B is entity dependent on A | 1 LF |
| | If B is entity independent of A | 2 LFs |
| (1) : N | If A is entity dependent on B | 1 LF |
| | If A is entity independent of B | 2 LFs |
| (1) : (1) | (A and B are independent) | 2 LFs |
| 1 : 1 | (A and B are dependent) | 1 LF |
| 1 : (1) | If B is entity dependent on A | 1 LF |
| | If B is entity independent of A | 2 LFs |
| (N) : (M) | (A and B are independent) | 2 LFs |
| N : M | If B is entity dependent on A | 1 LF |
| | If B is entity independent of A | 2 LFs |
| N : (M) | If B is entity dependent on A | 1 LF |
| | If B is entity independent of A | 2 LFs |

### Legend

| | |
|---|---|
| LF | = Logical File (ILF or EIF) |
| (..) | = Optional side of relationship |

### Notes

1: When in doubt, choose entity independent.

2: In some situations more than two entities can also form a logical file.

## Classifying Logical Files into Internal or External Logical Files (Step 3)

The identified logical files need to be checked against the ILF/EIF counting rules in Part 1.

Classify a logical file as an Internal Logical File (ILF) if elementary processes within the boundary of the application being counted, maintain (create, update or delete) data elements within the file.

Classify a logical file as an External Interface File (EIF) if elementary processes within the boundary of the application being counted only reference data elements within the file, *and* the logical file is maintained by an elementary process of another application.

If an identified logical file is *not* maintained by an elementary process (either within the application or in another), then the logical file is not counted at all.

# Identifying Record Element Types

The record element type (RET) represents the user's view of coherent *subgroups* of data within an identified logical file, which has been discussed in the previous Section “Identifying Logical Files and Classifying”.

Record element types typically correspond to the entities that were grouped into the logical files as discussed in Section “Identifying Logical Files and Classifying”. They need to be reviewed closely to ensure that the user would identify them as a logical subgroup and thus counted as a Record Element Type (RET).

## Record Element Type Terms and Definitions

This chapter looks at the logical data model in 3rd normal form and ignores multiple entities created for technical reasons; if you don't have a data model an attempt should be made to (de)normalize the data.

The working definitions are based on the data model concepts described in Section “Data Modeling Concepts”:

| | |
|---|---|
| **Associative Entity Type** | An entity type that contains attributes which further describe a many-to-many relationship between two other entity types. Also known as intersecting entities |
| **Attributive Entity Type** | An entity type that further describes one or more characteristics of another entity type |
| **Entity Subtype** | A subdivision of entity type. A subtype inherits all the attributes and relationships of its parent entity type, and may have additional, unique attributes and relationships |

## Mapping Data Modeling terms to Function Point Terminology

Data Modeling terms can be mapped to function point analysis as shown in the following matrix:

| Data Modeling Concept | Data Modeling Term | Relational Database Term | FPA Term | FPA Concept |
|---|---|---|---|---|
| Groups of related items which are treated as a unit | Record | Row or Tuple | Record Element Type (RET) | A record element type (RET) is a user recognizable subgroup of data elements within an ILF or EIF |
| Collection of records of a single type | File | Table | Logical File (Internal Logical File - ILF or External Interface File - EIF) | File refers to a logically related group of data and not the physical implementation of those groups of data |

The following table can be applied to further understand the concept of Record Element Types.

| Entity Relationship Concept | Entity Relationship Term | FPA Term | FPA Concept |
|---|---|---|---|
| Principal data objects about which information is collected (person, place, thing or event); a fundamental item of relevance to the user about which a collection of facts is kept | Entity or Entity Type | Logical File | File refers to a logically related group of data and not the physical implementation of those groups of data; if there are no other subgroups, the logical file is counted with a single Record Element Type (RET) |
| An entity type that contains attributes which further describe relationships between other entities | Associative Entity Type | Could be a logical file or a possible Record Element Type (RET); see Section "Analyzing Associative Entities to Determine RETs" for further consideration | User Recognizable subgroup of data elements within an ILF or EIF, can be optional or mandatory |
| An entity type that further describes one or more characteristics of another entity type | Attributive Entity Type | Possible Record Element Type (RET); see Section "Analyzing Attributive Entities to Determine RETs" for further consideration | User Recognizable subgroup of data elements within an ILF or EIF, can be either optional or mandatory |
| A division of entity type, which inherits all the attributes and relationships of its parent entity type, and may have additional, unique attributes and relationships | Entity Subtype | Possible Record Element Type (RET); see Section "Analyzing Subtypes to Determine RETs" for further consideration | User Recognizable subgroup of data elements within an ILF or EIF, can be either optional or mandatory |

Function Point Analysis looks at associative, attributive and subtypes as subgroups of data. These will be explored in the discussion of how to count RETs.

## Analyzing Associative Entities to Determine RETs

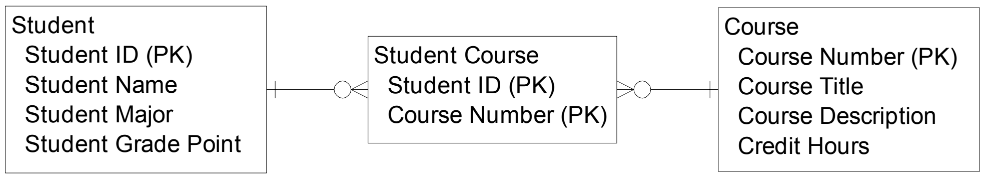
An associative entity is used to associate two or more entities as a way of defining the many-to-many relationship. This type of entity is often created by the data modeler to resolve some of the business rules required to relate two separate entities.

There are three possibilities to consider when looking at associative entities.

### Situation 1

Associative entity *is not* counted as a RET

A registrar has a requirement to manage all students registered for a course and to know the courses that the student has previously taken. Course is an entity, and Student is an entity. The data modeler creates an associative entity called Student Course as an intersection between the two, and this associative entity contains only the keys of each entity.



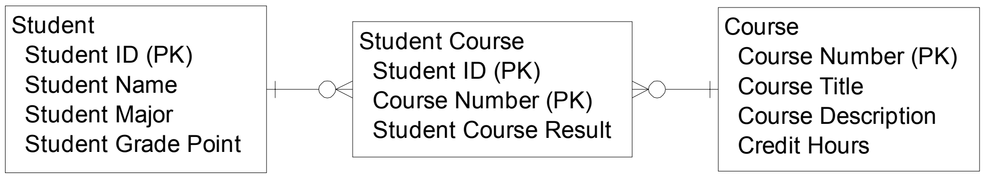
The entity Student Course *is not* considered a RET nor does it count as a separate logical file because it does not contain any additional data elements other than the two primary keys (PK) of the entities that it intersects.

Student is a logical file with 1 RET (Student) and Course is a logical file with 1 RET (Course).

**Situation 2**

Associative entity *is* counted as a RET

A registrar has a requirement to identify all students registered for a course. In addition he needs information about the results of the course for the student(s). Course is an entity, and Student is an entity. The data modeler creates an associative entity called Student Course as an intersection between the two, and this associative entity contains the keys of each entity as well as the student course result.



There are no business rules that require student course to be kept independently; therefore Student Course does not satisfy the rules to be counted as a separate logical file. In this case, the Student Course entity *is* considered a RET because it contains at least one user recognizable attribute (*) in addition to the two primary keys of the entities that it intersects.

(*) In Section “Identifying Entities that Should be Considered for Counting (Step 1)”, non-key attributes that are a result of a design or implementation consideration or satisfy a technical requirement are not considered data elements.

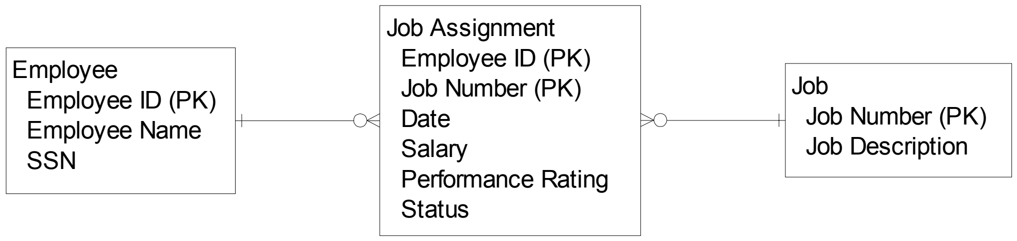
In addition to the primary keys, an additional RET in a logical file always contains one or more unique attributes to be counted as a RET. Student is a logical file with 2 RETs (Student and Student Course) and Course is a logical file with 2 RETs (Course and Student Course).

If the business requirement indicates that the associative entity relationship belongs to only one of the logical files, the RET would be counted only with that logical file.

**Situation 3**

Associative entity counted as a *logical file*, single RET

An HR department maintains information about Employee, Jobs and Job Assignments. Job Assignments is required, even if an Employee is no longer associated with the Job or if a Job is no longer a current job available for placement.



Although Job Assignment is an associative entity, it is more than a key-to-key mapping between two entities, and it is more than a RET associated with a logical file. If a business rule requires the Job Assignment information to be retained independently, Job Assignment is considered a logical file as described in Section "Identifying Logical Files Using Entity (In-)Dependency Method (Step 2B)".

If there are no business rules requiring the Job Assignment information to be retained independently, then this associative entity is counted as in Situation 2 above.

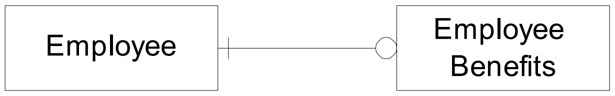
## Analyzing Attributive Entities to Determine RETs

An attributive entity is an entity type that further describes one or more characteristics of another entity type. By definition it is a logical extension of another entity; in Function Point Analysis an attributive entity represents a Record Element Type of that entity.

An attributive entity is counted as either a RET of the logical file it is further defining (Situation 1) or as an extension of the logical file (Situation 2).

### Situation 1

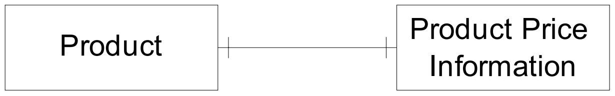
An *optional* attributive entity
An employee may enroll in employee benefits. In our data model Employee is an entity. Employee Benefits is an attributive entity of Employee and contains information about the benefits the employee has decided to carry. Employee Benefits cannot exist without Employee, and is thus logically related.



Employee Benefits is counted as a RET because it is an optional attributive entity. Employee is a logical file with 2 RETs, Employee and Employee Benefits.

### Situation 2

A *mandatory* attributive entity
A sales system must maintain information about each product and its related price information. Product is an entity. Product Price Information is an attributive entity related to Price containing: previous price, current price, projected future price and effective dates of the prices. Product Price Information does not exist without Product and is thus logically related.



Product Price Information is *not* counted as a RET. Product is a logical file with 1 RET, containing Product and Product Price Information.

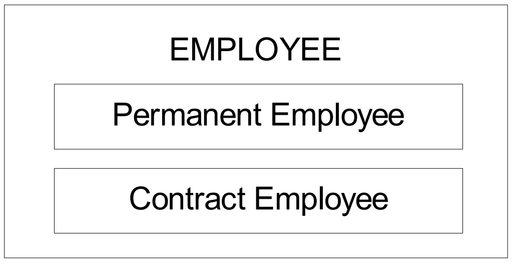
## Analyzing Subtypes to Determine RETs

An entity subtype is a subdivision of entity type. A subtype inherits all the attributes and relationships of its parent entity type, and may have additional, unique attributes and relationships. Data modeling rules state that an entity may have any number of sets of independent subtypes associated with it, which can be optional or mandatory. Each subtype can have only one generic parent. In data modeling, although the parent and subtype are represented as different entities, they are logically part of the same entity.

When analyzing subtypes in a data model, look at the required relationship to the 'parent' entity, as shown in the following situations.

### Situation 1

<u>Subtype that is a subgroup and therefore *is* counted as a RET</u>
An employee must be either a permanent employee or a contract employee, but cannot be both. The common employee data is relevant to all employees and is mandatory. In addition, this common data is inherited by the mandatory subtype entities, permanent and contract.



When reviewing this data from a function point perspective, two logical subgroups of the Employee logical file are identified:

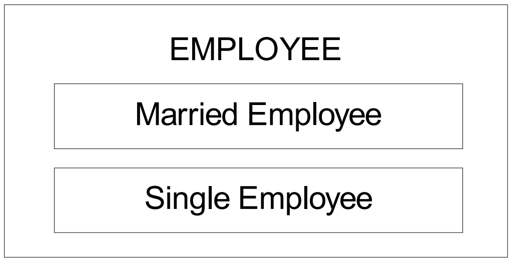
- Permanent employee data includes permanent employee information as well as common employee information.
- Contract employee data includes contract employee information as well as common employee information.

In this situation there is one logical file (Employee) with two RETs, permanent employees and contract employees.

**Situation 2**

Subtype that *is not* a subgroup and therefore is not counted as a RET
If there are unique attributes among entity subtypes, give serious thought to whether a separate subgroup really exists that would constitute a record element type (RET). A single optional, unique attribute would not result in a different RET from a user perspective, even if depicted as an entity subtype in a logical data model.

The marital status of an Employee may be married or single. If married, the spouse's name is recorded. Although it may be depicted as a subtype in a data model, the spouse's name is only an optional attribute within the logical group of data Employee.



One different attribute in this case does not make a significant difference between married and single employees from a business perspective.

**Hints**

Look at the data model carefully. When in doubt, ask the user the intent of the separate subtypes. The data analyst creates a data model depicting his view of the user's world. In practice, it depends on the user view/business rules whether these entity subtypes are relevant to the user and should be considered as RETs.

If there are separate add/update transactions with unique attributes for these entity subtypes, this would be an indication that we *may* have separate RETs for these entity subtypes.

## Miscellaneous Situations

If you do not have a data model to work with, you may encounter some of the following situations. Here are some additional hints for counting.

**Repeating Groups/Data** Repeating groups are instances of multiple occurrences of the same data, which could be repeated multiple times within one logical file.

**Situation 1** Repeating *groups* counted as RET
The Order group of data consists of Order Header and can have multiple instances of Order Line Item. Order Line Item contains more than one unique attribute. Order Header and Order Line Item represent two separate subgroups. We count two RETs for the Order logical file.

**Situation 2** Repeating *data* not counted as a RET
One repeating field (DET) would not result in a separate subgroup or RET. For example, an Employee may have several banking account numbers. That would *not* imply two RETs for Employee ("all data without banking account number" and "banking account numbers").

## Remarks

If in doubt, do *not* count a subgroup of information as a RET.

Identifying the correct number of RETs does not influence the *number* of logical files identified, it influences only the complexity of the logical file. Therefore, although this step does influence the outcome of a function point count, it does so to a lesser degree than identifying logical files in Section “Identifying Logical Files and Classifying”.

## Considering Record Element Types in Conjunction with Logical Files via Entity (In-)Dependence

Now that the Record Element Types have been discussed, the table shown in Section "Summary: from Entities to Logical Files via Entity (In-)Dependence" is expanded, applying RETs to the table.

| **Type of relationship between two entities, A and B** | **When this Condition Exists** | **Then Count as LFs with RETs as follows:** |
|---|---|---|
| (1) : (N) | (A and B are independent) | 2 LFs |
| 1 : N | If B is entity dependent on A | 1 LF, 2 RETs |
| | If B is entity independent of A | 2 LFs |
| 1 : (N) | If B is entity dependent on A | 1 LF, 2 RETs |
| | If B is entity independent of A | 2 LFs |
| (1) : N | If A is entity dependent on B | 1 LF, 2 RETs |
| | If A is entity independent of B | 2 LFs |
| (1) : (1) | (A and B are independent) | 2 LFs |
| 1 : 1 | (A and B are dependent) | 1 LF, 1 RET |
| 1 : (1) | If B is entity dependent on A | 1 LF, 1 or 2 RETs |
| | If B is entity independent of A | 2 LFs |
| (N) : (M) | (A and B are independent) | 2 LFs |
| N : M | If B is entity dependent on A | 1 LF, 2 RETs |
| | If B is entity independent of A | 2 LFs |
| N : (M) | If B is entity dependent on A | 1 LF, 2 RETs |
| | If B is entity independent of A | 2 LFs |

**Legend**

| | |
|---|---|
| LF | = Logical File (ILF or EIF) |
| (..) | = optional side of relationship |
| RET | = Record Element Type |

**Note**

In some situations more than two entities can also form a logical file; in such a case more than 2 RETs should be counted

## Identifying Data Element Types

The data element  is the smallest unit that has meaning to the user and represents a specific fact about a business, for example:

| **Name** | **Value** |
|---|---|
| **Fee** | **$900** |
| **Date of Birth** | **15 Jan 1965** |
| **Name** | **InfoMerge** |

### Data Element Terms and Definitions

Once we start looking at a logical data model, we begin to consider these data elements as attributes. An attribute represents a specific fact about an entity or a relationship. In the following table the entities are shown in UPPER CASE, the attributes in lower case:

| **Representation** | **Example** |
|---|---|
| **ENTITY_attribute** | **COURSE_fee** |
| **ENTITY.attribute** | **CLIENTCOMPANY.name** |

Data elements/attributes can be found in

- user views (reports, screens)
- data dictionaries (business models, data models)
- current files (record structures in programs, file layouts)

When reviewing the data, the data analyst follows this basic premise: every data element known to the user must be treated as an attribute, and thus must be shown in relation to a specific entity.

The attribute may have the following properties: attribute name (alias), characteristic, purpose (usage), value source, value range, value (structure), unit of expression, and dependencies. We will explore these properties before reviewing the IFPUG Function Point Analysis mapping of DETS to data elements/attributes.

**Attribute Name**

A unique name that summarizes the characteristics being represented. It contains the following components:

**home** (entity/relationship) *followed by a period*
**descriptive** (adjective of the attribute) *followed by a hyphen*
**class** (base attribute)

Many technologies do not accept spaces. So multiple names are concatenated with hyphens.

Example:
CLIENT_COMPANY.shipping-address
SEMINAR_ENROLLMENT.effectiveness-evaluation

**Characteristic**

The property of the environment being measured or represented. The name of the entity that has the characteristic is always stated; for example, Shipping-Address: the address to which materials will be shipped to the CLIENT-COMPANY

**Purpose**

Asking the question “what is the attribute used for in the business” justifies the att

Example: COURSE.Qualification-date
Purpose: Used in scheduling refresher sessions

**Dependencies**

The situations where other values of attributes in the model influence or constrain the value of this attribute. For example, COURSE-final grade cannot exist before term-end but must exist upon course completion.

**Attribute Keys**

Provide the relationship between one entity and another. There are different types of keys in a data model, e.g., primary keys, secondary keys and foreign keys.

A **Primary key (PK)** is the unique id of an entity.

**Secondary keys** (SK) are attributes designed to provide fast access to the information, such as:

TEXTBOOK.cost (SK)

TEXTBOOK.publisher-name (SK).

Secondary keys are not part of the information data model (logical data model) but are used primarily as an aid to access (physical implementation).

**Foreign keys (FK)** are attributes used to represent relationships of one entity to another.

**Attribution**

The last data modeling concept we must consider before analyzing our DETs is Attribution, which prescribes/describes where the attribute resides, either within the entity or within a relationship. There are some general attribution rules that are followed in data modeling:

1. An attribute is assigned to its single best "home", which is indicated on the characteristic property in the ATTRIBUTE DEFINITION FORM.
2. The unique id (primary key) of an entity will also be attributed to every relationship in which it participates.

There are some additional guidelines that are followed when trying to place an attribute in the appropriate entity:

1. If the attribute definition refers to one entity, assign the attribute to that entity
2. If the attribute definition refers to multiple entities, then create a relationship and assign the attribute to the relationship or the entity where it applies.

## Mapping Data Elements to FPA Data Element Types

Now that we have reviewed the basis of data elements and attributes from a data modeling perspective, we can relate these concepts to the IFPUG Function Point definitions and rules:

| Data Modeling Concept | Data Modeling Term | Relational Database Term | FPA Term | FPA Concept |
|---|---|---|---|---|
| Smallest named unit of data that has meaning to the real world | Data Item | Attribute or Column | Data Element Type (DET) | A data element type (DET) is a unique, user recognizable non-repeated field |
| Groups of related items which are treated as a unit | Record | Row or Tuple | Record Element Type (RET) | A record element type (RET) is a user recognizable subgroup of data elements within an ILF or EIF |
| Collection of records of a single type | File | Table | Logical File (Internal Logical File - ILF or External Interface File - EIF) | File refers to a logically related group of data and not the physical implementation of those groups of data |

**Data element types (DETs)** are unique user-recognizable, non-repeated fields or attributes.

The following rules apply when counting DETs in logical files:

- Count one DET for each unique, user-recognizable, non-repeated field maintained in or retrieved from a logical file, including all key attributes
- When two or more applications maintain and/or reference the same logical file, count only the DETs being used by each application
- Count a DET for each piece of data required by the user to establish a relationship with another logical file

Do not count attributes that exist purely to satisfy a technical requirement and have not been specified by the user. Examples of these technical attributes are attributes that are a result of a design or implementation consideration.

Example: The ORDER_date is counted as a DET on Order since it needs to be maintained to satisfy a user business requirement. However the time and date stamp on each order record is there to satisfy data integrity and data reliability. The technical solution to these quality requirements was to back up the database and be able to restore based on this time and date stamp information.

## Miscellaneous Situations

The following are examples of counting our basic element, data element type (DET).

### Attributes

Attributes that are composed of several related data elements are stored separately.

Should the attributes be counted as multiple data elements or a single data element? The DET Rules in Part 1 tell you to "count each user recognizable field".

How do you determine whether it is user recognizable as one thing or multiple things? Review the transactions within the application to determine whether the attribute is treated as one item or more than one.

Consider the following items in making a decision:

a) If the attribute is always used in its entirety, then it is counted as a single data element (DET). There should be no situations when an individual component of the attribute is used without the others. Based on that usage, the attribute is counted as a single data element.
b) If in some situations, only one part of the attribute (i.e., last name) is used, then more than one data element would be counted. Look at the usage of the components within the application to determine how many recognizable pieces exist. It need not be a one or all option. Based on what you're seeing, it might be appropriate to count just two DETs, even though there are actually five physical pieces.
c) Look for the existence of sort or edit requirements and selection criteria. If lists or reports are sorted or filtered by a single component of the attribute, this suggests independence of the components in the user view.

## Counting Names

Name (first name, middle initial and last name)

Many applications are required to keep track of people's names.
Should the name be counted as multiple data elements or a single data element?

Review the transactions within the application to determine whether the name is treated as one item or more than one. For example: In Case Studies 1, 2, and 3, look at how the Employee Name is used in the various transactional functions.

Does it always use the entire name, or does it sometimes just use one piece?

In the Case Studies the Employee Name is always used in its entirety. There are no screens or reports where a single piece of the name is used without the others. There are also no situations where a portion of the name is used as sort, edit or filter criteria. Its usage within the application suggests that Employee Name is a single data element (DET).

## Counting Addresses

Address (street address, city, state and zip code)

Many applications are required to keep track of addresses. Should the address be counted as multiple data elements or a single data element?

Review the transactions within the application to determine whether the address is treated as one item or more than one. For example: In Case Studies 1, 2, and 3, look at how the Location Address is used in the various transactional functions.

Do these transactions always reference the entire address, or do they sometimes use just one piece?

In the Case Studies the Address is always used in its entirety. There are no screens or reports where a single piece of the address is used without the others. There is no situation where a portion of the address is used as sort, edit or filter criteria. Its usage within the application suggests that Location Address is a single data element (DET).

If there were a location list that allowed the user to list all locations within a particular city, state or zip code, then more than one data element would be counted. Based on the information provided, it appears that city, state and zip code are recognizable by the user as separate from the address itself. Therefore four DETs would be counted (Street Address, City, State, and Zip Code).

## Counting Repeating Fields

Often applications maintain multiple occurrences of a data element. According to the DET Rules in Chapter 6 of Part 1, count a repeating field only once.

After the repeating fields are reduced to a single DET, verify that the business requirements are still being met. Consider the following examples:

### Example 1: Employee ID

In Case Studies 1, 2, and 3, look at the requirements for Employee Maintenance and the relationship between Employee and Dependent in the Entity-Relationship Diagram (ERD). According to the results of the Case Studies, Employee is a LF that includes Dependent. The logical files for Employee and Dependent would each contain the Employee ID.

Applying the “repeating” DET rule, count Employee ID as a single DET for the Employee ILF. Now determine if the business requirements are still met.

The requirement was to maintain Dependent as part of the Employee information. Yes, the business requirement is satisfied therefore one DET is counted.

### Example 2: Daily Hours Worked

A time reporting system typically maintains the number of hours a person works each day. Upon reviewing the data structures, separate Hours Worked are saved for each day of the week (Monday Hours Worked, Tuesday Hours Worked, etc.).

Applying the “repeating” DET rule, count only Hours Worked. Determine if the business requirements are still met. If the application only keeps track of the Hours Worked, does it meet the business requirement to maintain hours worked each day?

No, it does not. In order to meet that requirement, count Day of the Week. The application has the ability to separately track each day’s hours worked (Monday Hours Worked, Tuesday Hours Worked, etc). Therefore two DETs are counted.

## Counting Status Fields

Applications frequently keep track of the current status of data (i.e., Active, Inactive, Pending, Approved, etc). This status is typically updated through various transactions within the application. These status fields may or may not actually be physically visible to the user through the application’s transactions. Consider the following examples:

**Example 1: Status Inactive**

In Case Studies 1, 2, and 3, include an indicator of Status. When a job or employee is deleted, the user requirements indicate that any related job assignment should be updated to set the status to "inactive".

Upon review of the job assignment screens throughout the application, the status is never displayed or directly updated on any screen. Despite that lack of visibility, the Status is still counted as a DET for the Job Assignment. The fact that it appears in the Logical Data Model and the User Requirements suggests that it is user recognizable in the Logical File but not in any transaction. Therefore one DET is counted in the ILF.

**Example 2: Status *Not Counted***

The user requires the ability to delete Employees. The technical team does not want to actually delete the records; therefore, they implemented a status flag on Employee. When the user deletes an Employee, the Status is set to Inactive. The user is unaware of the Status field on Employee. Therefore, the Status field should not be counted as a DET.

## Counting System Dates

Applications frequently retain system dates associated with their data to reflect the currency of the data. System Dates may have many different names (last updated, last approved, etc.) and are often accompanied by a user ID (last updated by, last approved by). These system dates are typically updated through various transactions within the application. In many cases the system date is being kept for business purposes. The user needs to know when the data was changed or approved. There are also cases where the system date is being kept for technical reasons only.

Consider the following examples:

**Example 1: Effective Date**

Case Studies 1, 2, and 3 include a reference to *Effective Date*. When a job or employee is deleted, the user requirements indicate that any related job assignment should be updated to set the effective date to the current system date.

Upon review of the job assignment screens throughout the application, the effective date is never displayed or directly updated on any screen. Despite that lack of visibility, the Effective Date is still counted as a DET for the Job Assignment. The fact that it appears in the Logical Data Model and the User Requirements reference it by name suggests that it is user recognizable. Therefore one DET is counted in the Logical File.

**Example 2: Recover-to-Date**

The Backup/Recovery tool used by the application uses system dates stored on the tables to restore the data to a particular point in time. In this case, the *recover-to-date* is not user recognizable and should not be counted.

**Example 3: Audit Date**

The technical team decided they should record the system date and user ID whenever data is changed to resolve any later questions about when or by whom a change was made, they would be able to provide the answer. In this case, the *system date* is not user recognizable and should not be counted.

## Counting Foreign Keys

Applications frequently maintain relationships from one entity to another. In some cases they exist to satisfy data validation requirements, but in other cases they indicate some business interaction between the two entities. The data modeling concept of Attribution is best illustrated by the assignment of foreign keys.

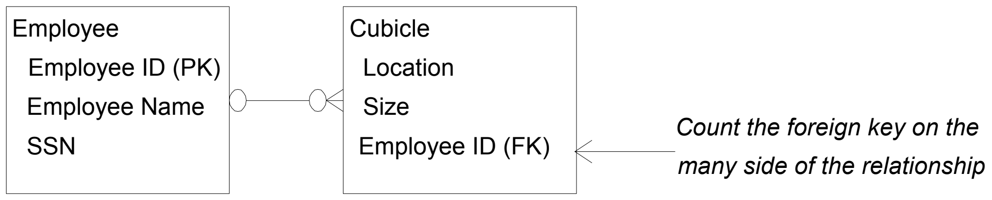
**Example 1: Location**

Case Studies 1, 2, and 3, include the user requirements for Adding and Updating Employees: "The *location* must be a valid location in the Fixed Asset System (FAS)."

The ERD also reinforces this business requirement by illustrating the relationship between the Employee entity and Location entity. According to the ERD, an Employee can have one Location, and a Location can have many Employees. In this situation, the Location Name should be included in the attributes within the logical and physical tables. Count the Location Name attribute as a data element (DET) for Employee.

**Example 2: Cubicle 1:N**

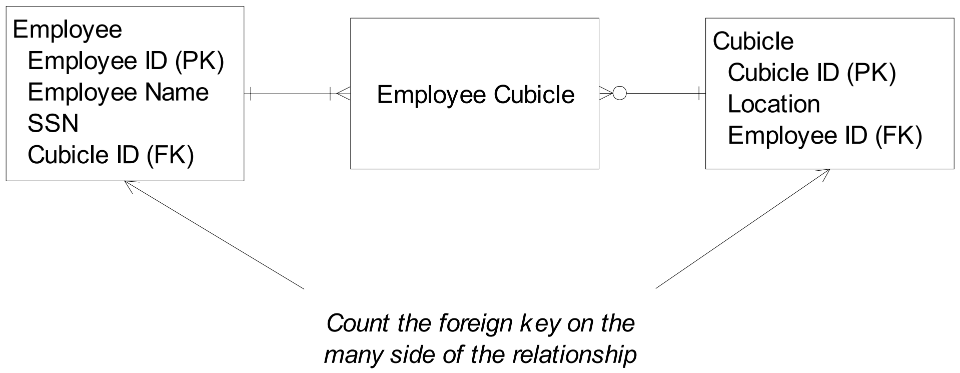
Consider a variation to the previous example. An employee may have an unlimited number of cubicles. A cubicle can be occupied by no more than one employee at a time. The cubicle ID(s) must be valid as identified in the CUBICLE table. The ERD would again reinforce the business requirement by illustrating the relationship between the CUBICLE entity and the EMPLOYEE entity. According to this ERD, an Employee can have many cubicles, but a Cubicle can be occupied by only one Employee. In this situation, the Employee ID is an attribute in CUBICLE.



The relationship would instead be reflected in the Cubicle table by identifying the Employee that occupies the cubicle. Employee ID is counted as a data element (DET) for Cubicle.

**Example 3: Cubicle N:M**

Consider a further variation to the previous example. An employee must have at least one cubicle, but may have an unlimited number of cubicles. A cubicle can be occupied by more than one employee at a time. The Cubicle ID must be valid as identified in Cubicle. The ERD would again reinforce the business requirement by illustrating the relationship between Employee and Cubicle. According to this ERD, an Employee can have many cubicles, and a Cubicle can be occupied by many Employees.



The relationship would be reflected in the Employee-Cubicle table, which would contain an occurrence for each employee to cubicle relationship. It would include the Employee ID and the Cubicle ID as the primary keys. As explained in Sections "Identifying Logical Files and Classifying" and "Identifying Record Element Types", Employee Cubicle is not counted as a Logical File nor as a RET. Cubicle ID is counted as a data element (DET) in Employee and Employee ID is counted as a data element (DET) in Cubicle.

## Remarks

Identifying the correct number of DETs does not influence the *number* of logical files, but their complexity only. Therefore, this step does influence the outcome of a function point count to a limited degree, considerably less than step 1 (Section "Identifying Logical Files and Classifying") and even less then step 2 (Section "Identifying Record Element Types").

Don't forget to reference the rules and hints concerning counting DETs in Part 1, because they are decisive.

## Considering Data Element Types in Conjunction with Logical Files via Entity (In-) Dependence and Record Element Types

Now that we have discussed Data Element Types, we will expand the count shown in Section "Summary: from Entities to Logical Files via Entity (In-)Dependence", applying DETs to the table.

| Type of relationship between two entities, A and B | When this Condition Exists | Then Count as LFs with RETs and DETs as follows: |
|---|---|---|
| (1) **:** (N) | (A and B are independent) | 2 LFs, **DETs** to each |
| 1 **:** N | If B is entity dependent on A | 1 LF, 2 RETs, sum **DETs**: count each unique, non-repeated field |
| | If B is entity independent of A | 2 LFs, **DETs**: count each unique non-repeated field for each LF |
| 1 **:** (N) | If B is entity dependent on A | 1 LF, 2 RETs, sum **DETs**: count each unique, non-repeated field |
| | If B is entity independent of A | 2 LFs, **DETs**: count each unique, non-repeated field |
| (1) **:** N | If A is entity dependent on B | 1 LF, 2 RETs, sum **DETs:** count each unique, non-repeated field |
| | If A is entity independent of B | 2 LFs, **DETs**: count each unique, non-repeated field |
| (1) **:** (1) | (A and B are independent) | 2 LFs, **DETs**: count each unique, non-repeated field |
| 1 **:** 1 | (A and B are dependent) | 1 LF, 1 RET, sum **DETs**, counting each unique, non-repeated field |
| 1 **:** (1) | If B is entity dependent on A | 1 LF, 1 or 2 RETs, sum **DETs**, counting each unique, non-repeated field |
| | If B is entity independent of A | 2 LFs, **DETs**: count each unique non-repeated fields |
| (N) **:** (M) | (A and B are independent) | 2 LFs, **DETS**: count each unique, non-repeated field |
| N **:** M | If B is entity dependent on A | 1 LF, 2 RETs, sum **DETs:** count each unique, non-repeated field |
| | If B is entity independent of A | 2 LFs, **DETs**: count each unique, non-repeated field |
| N **:** (M) | If B is entity dependent on A | 1 LF, 2 RETs, sum **DETs**, counting each unique, non-repeated field |
| | If B is entity independent of A | 2 LFs, **DETs**: count each unique, non-repeated field |

**Note** Count the foreign key on the many side of the relationships.

## Legend

| | |
|---|---|
| LF | = Logical File (ILF or EIF) |
| (..) | = optional side of relationship |
| RET | = Record Element Type |
| DET | = Data Element Type |

## Works Cited, Works Consulted

The following sources were consulted or cited in this chapter.

Booch, Grady, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language User Guide. Reading: Addison-Wesley, 1994. ISBN: 0-2015-7168-4.

Definitions and Counting Guidelines for the Application of Function Point Analysis: A Practical Manual, Version 2.0. (English), NESMA, 1997. ISBN: 90-76258-02-3.
Note: This manual is also called NESMA Counting Practices Manual. It describes the standard FPA methodology, and many aspects related to the application of FPA. It may be used together with the IFPUG manual. For more information, see the NESMA web site www.nesma.org.

Garmus, David, David Herron. Function Point Analysis: Measurement Practices for Successful Software Projects. Boston: Addison-Wesley Information Technology Series, 2001. ISBN: 0-201-69944-3.

Martin, James, Carma McClure. Diagramming Techniques for Analyst and Programmers. Englewood Cliffs: Prentice-Hall, Inc., 1985. ISBN: 0-132-087944.

*MLA Handbook for Writers of Research Papers, Fifth Edition*, Modern Language Association of America. Boston: Addison Wesley, 1999.

Reingruber, Michael C. and William W. Gregory. The Data Modeling Handbook: A Best- Practice Approach to Building Quality Data Models. Canada: John Wiley & Sons, Wiley-QED Publication, 1994. ISBN: 0-471-05290-6.

Silverman, Len, W. H. Inmon, Kent Graziano. The Data Model Resource Book: A Library of Logical Data Models and Data Warehouse Design. Boston: Addison-Wesley, Inc. Out of Print: AISN: 0-471-15364-8.

Simsion, Graeme. Data Modeling Essentials: Analysis, Design, and Innovation. Boston: International Thomson Computer Press, 1994. ISBN: 1-850-932877-3.

Schuldt, Gary. "Information Modeling for Information Systems Analysts", A workshop at AT&T Bell Laboratories. Holmdel, N.J., May, 1992.

Teorey, Toby J. Database Modeling & Design: The Fundamental Principles, Second Edition. San Francisco: Morgan Kaufmann Publishers, Inc., 1994. ISBN: 1-558-60291-1.

3

# Shared Data

**Introduction**

This chapter provides additional guidelines to aid in identifying external interface files (EIFs) and internal logical files (ILFs) as well as transaction files when two or more systems interact (i.e., guidance, clarification, and direction in counting data that is shared across systems).

**Contents**

This chapter includes the following sections:

# Counting Data Shared Between Systems

This chapter provides additional guidelines to aid in identifying external interface files (EIFs) and internal logical files (ILFs) as well as transaction files when two or more systems interact. It provides further clarification and direction in counting data that is shared across systems.

<u>The purpose of shared data:</u>
Systems that share data with other systems:

a) reference or utilize the data to complete a transaction being processed within the system receiving or accessing the data, or
b) maintain the internal logical files within the system receiving or accessing the data.

<u>Methods of sharing data:</u>
Shared data, used by elementary processes within an application to maintain data on an internal logical file or to present data to the user, may be transferred:

1. via on-line screens (e.g., screen scraping)
2. via direct access of data files from other systems
3. via transferred files
4. via direct on-line real-time information requests
5. via web-applications

In order to correctly analyze these implementations, users need to consider primary intent and have a common understanding of the terms that represent the various implementation techniques.

## Primary Intent

The concept of primary intent is helpful in the identification of internal logical files and external interface files as it relates to the use of the data within the system being analyzed. Primary intent refers to the most significant or important thing the function is intended to accomplish. The definition of primary intent is “first in importance.” Therefore, it is important for us in our discussion of each scenario **to determine the primary intent**. The physical implementation does not affect the primary intent and thus should not influence the analysis

## Part 1 Definitions (ILF/EIF)

**Internal Logical File (ILF Definition):**

An internal logical file (ILF) is a user identifiable group of logically related data or control information maintained within the boundary of the application. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted.

Note: The term maintained is the ability to modify data through an elementary process. Examples include, but are not limited to, add, change, delete, populate, revise, update, assign, and create.

**External Interface Files (EIF Definition):**

An external interface file (EIF) is a user identifiable group of logically related data or control information referenced by the application, but maintained within the boundary of another application. The primary intent of an EIF is to hold data referenced through one or more elementary processes within the boundary of the application counted. This means an EIF counted for an application must be in an ILF in another application.

## Common Terms

The following common terms are utilized in this document to describe physical implementation techniques:

| Term | Use in Document |
|---|---|
| **Copy** | IEEE[1] definition:<br>(1) To read data from a source, leaving the source data unchanged, and to write the same data elsewhere in a physical form that may differ from that of the source. For example, to copy data from a magnetic disk onto a magnetic tape.<br>(2) The result of a copy process as in above. For example, a copy of a data file. |
| **File** | IEEE definition:<br>"… set of related records treated as a unit. For example, a file could consist of a set of invoice records." |
| **Image** | An exact replication of another object, file, or table usually created through a utility. |
| **Load** | IEEE definition:<br>"… to copy computer instructions or data from external storage to internal storage …" |

[1] Institute of Electrical and Electronics Engineers (IEEE) Std 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology*

| Term | Use in Document |
|---|---|
| **Merge** | Multiple files with the same data elements consolidated into a single file. |
| **Refresh** | The process of recreating a set of data to make it current with its source. |

## Organization of Counting Scenarios

Function point analysis is often based on descriptions from developers of the physical characteristics of an installed system. This chapter will address the physical descriptions a function point analyst frequently encounters to assist in the correct interpretation of many of these system interfaces.

This chapter utilizes various scenarios as an aid in analyzing shared data situations.

### Approach

This chapter uses the following approach in our discussion of shared data:

- ***Description*** – A high level statement of the example being discussed.
- ***Scenario*** - An example is presented that often describes a physical activity or transaction regarding files being transferred between two systems; e.g., sharing data.
- ***Scenario Diagram*** - The scenario is depicted graphically, as an aid to map a similar situation or scenario.
- ***Counting Interpretation*** - A counting interpretation for the scenario is provided, which includes a discussion of the example and how the example should be counted as well as any assumptions regarding primary intent.
- ***Solution Diagram*** - The solution is depicted graphically.
- ***Counting Summary*** - The count is summarized in a table showing the data and transactional functions applicable to each system.
- ***FAQs/Variations*** – If applicable, some common variations of a scenario may be included following the discussion.

### Symbols Used in Diagram Solution

The following symbols are found in the diagram solution:

☑ above a component type depicts the component **is counted** for system;

☒ above a component type indicates the component **would not be counted** for the scenario.

## Scenario Naming Conventions

For consistency, the following naming conventions have been used throughout the scenarios

| Term | Description |
|---|---|
| **System A** | The source system for the reference/transaction data. |
| **System B** | The receiving system for the reference/transaction data. |
| **File X** | An ILF counted in System A. |
| **File X' (X Prime)** | An ILF counted in System B which is a subset of the File X data. |
| **File Y** | An ILF counted in System B. |
| **File Z** | A data transfer file. This file is generated by System A and read (processed) by System B. |
| **Application Boundaries** | System A and System B represent two separate applications, and thus represent two separate application boundaries. |

## Outline of Scenarios

The following scenarios do not represent a complete list of the various ways shared data is implemented, but they provide guidance on many of the situations encountered. Understanding these examples will facilitate the understanding of additional scenarios that may be encountered.

The scenarios focus on situations where the data required to complete elementary processes within System B is obtained from System A. System B is the application being counted. The scenarios are logically divided into two groups, each of which has various implementations:

**GROUP 1:**

The primary intent is for System B to reference data maintained by System A. There are two areas which are addressed, functional and technical.

**Functional** For Functional Reasons (business requirements), systems share data in the following scenarios

| Scenario Number | Scenario | Summary Description |
|---|---|---|
| 1 | READ | System B physically accesses System A's data. This example is currently documented in the CPM. |
| 2 | STATIC IMAGE COPY | System A generates an image copy of a data store, which reflects the current state of the data at a certain time and remains within its boundary. |

**Technical** For Technical Reasons (performance, security, etc.), System B must use System A's data and does so in the following ways

| Scenario Number | Scenario | Summary Description |
|---|---|---|
| 3 | IMAGE COPY / LOAD, No Processing Logic | System A generates an image copy with no additional processing logic and sends it to System B; System B loads the copy with no additional processing logic |
| 4 | IMAGE COPY / LOAD .A Subset of an ILF | System A generates an image copy of a subset (e.g., RET) with no additional processing logic and sends it to System B. System B loads the RET with no additional processing logic |
| 5 | COPY MERGE "Refresh" | Data stored in two systems is image copied and merged to form one file that is loaded into a third system. |
| 6 | SCREEN SCRAPING | Accesses System A's screen transactions to reference/obtain data to assist in processing a transaction within System B. |

**GROUP 2** The primary intent is for System B to maintain its own data from data maintained by System A.

| Scenario Number | Scenario | Summary Description |
|---|---|---|
| 7 | MAINTAIN COMMON DATA STORE | The same data store is maintained by two different applications. This example is currently documented in the CPM. |
| 8 | STANDARD TRANSACTION DATA | Transaction data provided by the source system. |

## Counting Scenarios

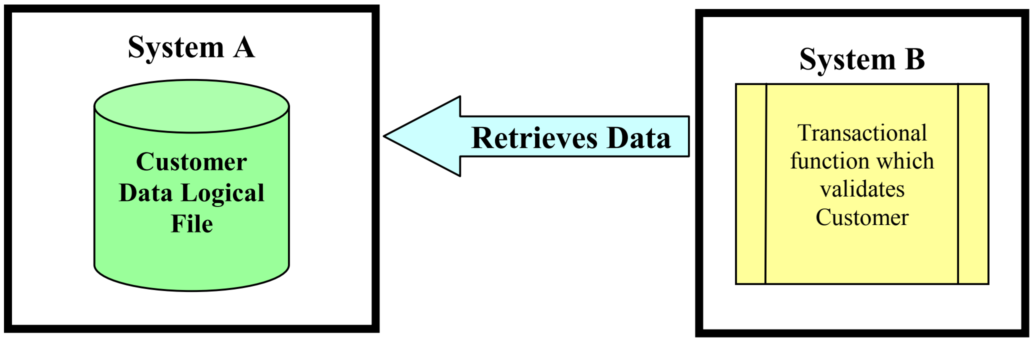
In each of these scenarios, the primary intent is for System B to reference data maintained by System A and may be implemented in the following ways:

### Scenario 1: Read

**Description** System B physically accesses System A's data to execute a query.

**Scenario** A transaction processed by System B, requires information from a data store maintained within System A. System B is responsible for accessing the data in System A, and System B maintains the software for that access.

**Diagram**

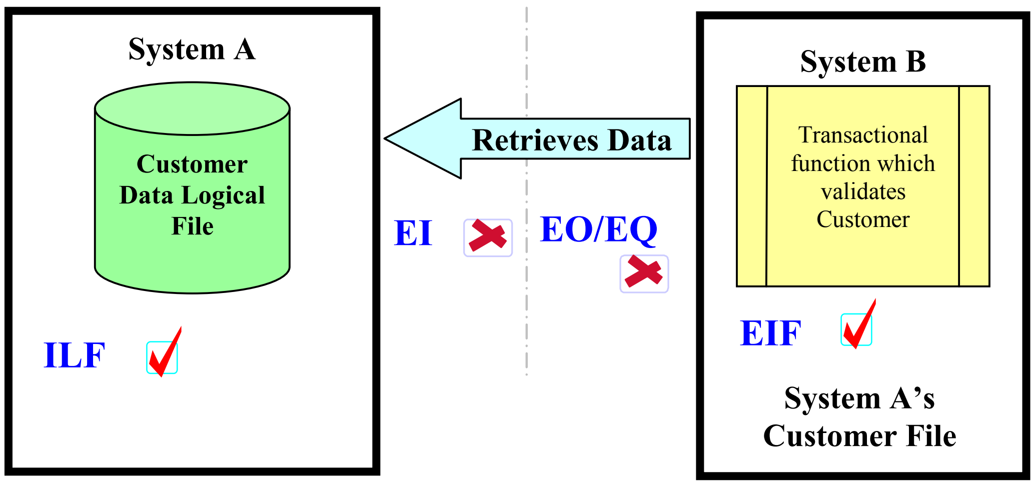

System A
Customer Data Logical File
Retrieves Data
System B
Transactional function which validates Customer

**Counting Interpretation**

***System A*****:** From System A's perspective, there is no requirement to send data. The data is available in System A. No credit is given to System A for the transaction performed by System B, although the data file is an ILF for System A.

***System B***: From System B's perspective, both logically and physically, there is only one data store involved. System B counts the data store, which resides in System A as an EIF. System B also counts that data file as an FTR in the transaction.

**Solution Diagram**


System A
Customer Data Logical File
ILF
Retrieves Data
EI
EO/EQ
System B
Transactional function which validates Customer
EIF
System A's Customer File

**Counting Summary**

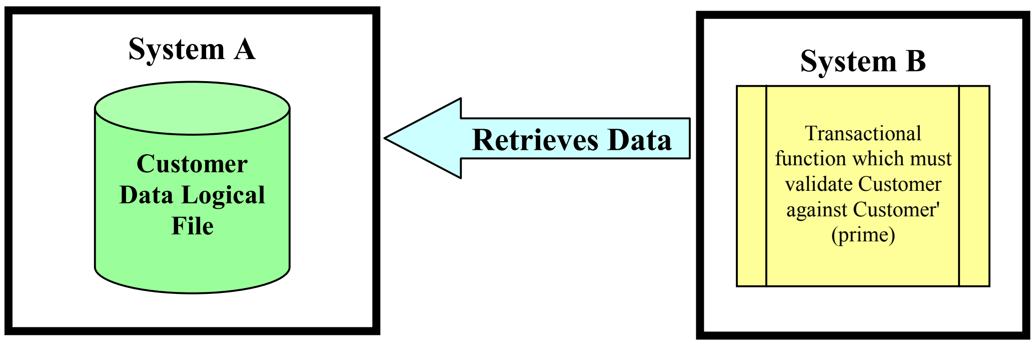
| | ILF | EIF | EI | EO/EQ | Note |
|---|---|---|---|---|---|
| System A | ☑ | | | | |
| System B | | ☑ | | | Customer is also counted as an FTR in transactional function |

## Scenario 2: Static Image Copy

**Description** System A generates an image copy of an ILF, which reflects the current state of the data at a certain time, and remains within its boundary.

**Scenario** In the banking industry, financial transactions are settled daily between all financial institutions. Subsequent customer financial transactions are validated against the customer financial balance as of that settlement. To support that business requirement, System A periodically makes an image copy of the data from the logical file Customer to Customer prime (or Customer') so that other systems can reference it. Customer' (prime) remains within the boundary of System A. Differences can occur between the current data in Customer and the data in Customer'. System B makes use of Customer'.

**Diagram**

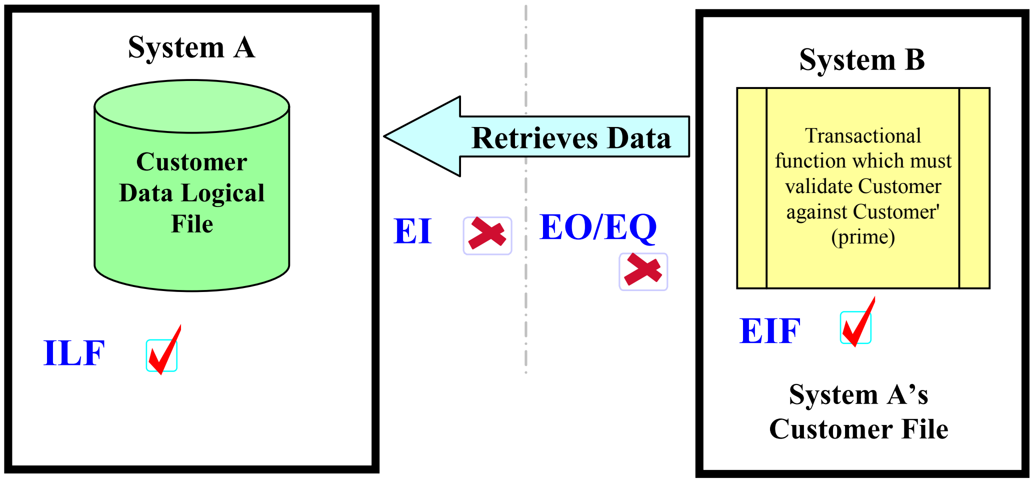

System A
Customer Data Logical File
Retrieves Data
System B
Transactional function which must validate Customer against Customer' (prime)

**Counting Interpretation**

***System A:*** From System A's perspective, Customer is an internal logical file for System A. Customer' (prime) is not counted as a separate ILF, nor is it counted as a RET of Customer. Customer' is just a snapshot of Customer at a particular time.

***System B:*** From System B's perspective, Customer is an external interface file for System B and is also counted as an FTR for the transaction in System B.

**Solution Diagram**


System A
Customer Data Logical File
ILF
Retrieves Data
EI
EO/EQ
System B
Transactional function which must validate Customer against Customer' (prime)
EIF
System A's Customer File

**Counting Summary**

| | ILF | EIF | EI | EO/EQ | Note |
|---|---|---|---|---|---|
| System A | ☑ | | | | |
| System B | | ☑ | | | Customer is also counted as an FTR in transactional function |

## Scenario 3: Image Copy/Load - No Additional Processing

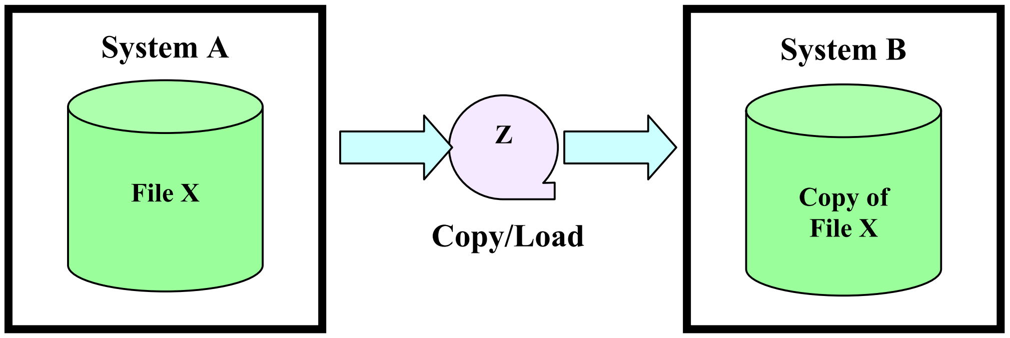
**Description**

System A generates an image copy with no additional processing logic and sends it to System B; System B loads a copy with no additional processing logic

**Scenario**

System B requires the ability to access file X in System A for validation and reference only. System B requires (e.g., performance, etc.) that System A send a complete file to System B. The existing data store in System B is "refreshed" each time with the copy

**Diagram**



**Counting Interpretation**

From System A's perspective, this data transfer is a technical solution devised to satisfy the business requirement that System B should have access, for data retrieval purposes, to the System A File X.

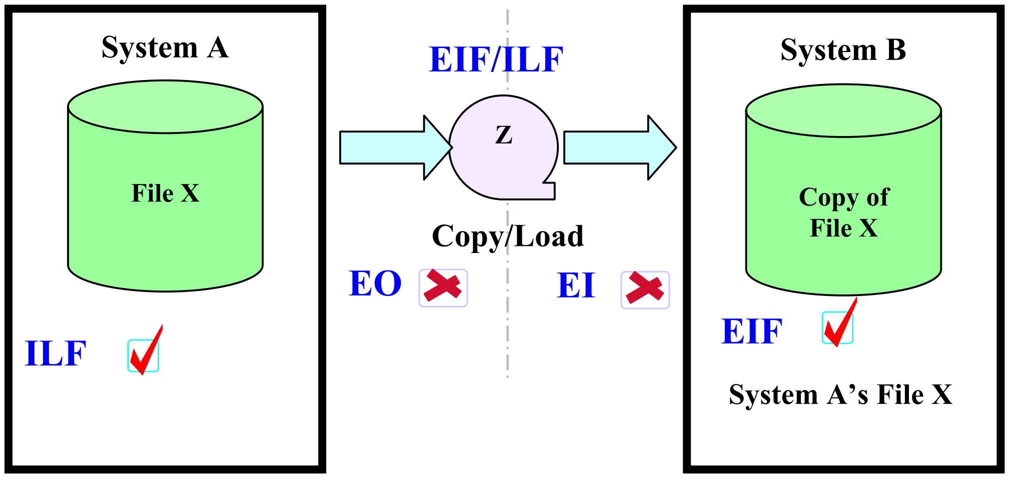
<u>Logically</u> the data store remains in System A. In this case, copying the data store from one system to another is the solution of a technical requirement (e.g., the data in System A is not available when it is required by System B).

The *primary intent* for B is to reference the data that is logically in A. One additional indication would be that the file in System B is "refreshed" each time with the copy. Also, no processing logic is performed in either System A or System B.

*Transactions* - The data transfer transactions: System A download, and System B Load, are part of the technical solution and are not counted in either system. In practice, when counting System A in isolation, it may not be apparent to the Function Point Analyst that this is a technical transaction, and it may be incorrectly counted as an EO/EQ. Neither system counts File Z as a transactional function.

*Files* - There is only one logical file involved. System A counts File X as an ILF. System B counts its copied version of File X as an EIF. Neither system counts File Z as a data function.

## Solution Diagram



## Counting Summary

| | ILF | EIF | EI | EO/EQ | Note |
|---|---|---|---|---|---|
| System A | ☑ | | | | |
| System B | | ☑ | | | |

## FAQs, Additional Variations

**Q?** What if a logical file in System A is made up of multiple physical tables and System A provides individual copies of more than one table to System B?

**A:** The EIF is identified in the same manner as in the scenario above; only the fields used are counted as DETs.

**Q?** What if you have a partitioned data store?

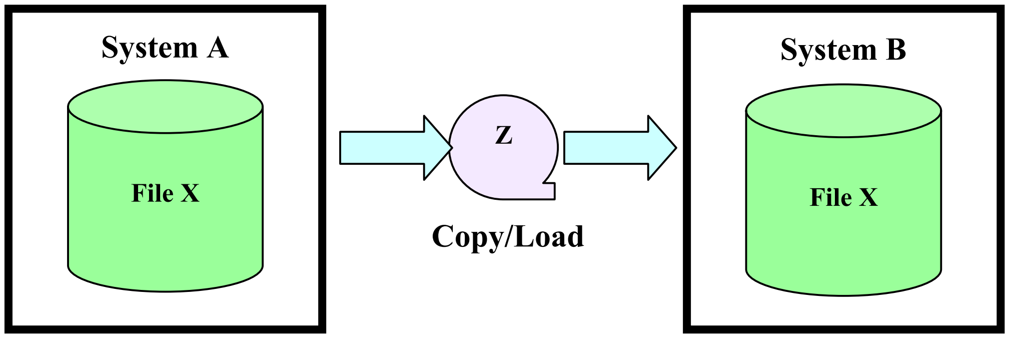
**A:** It is segmented for performance, but it is still logically one data file; physical implementation

## Scenario 4: Image Copy/Load One Physical Table - No Additional Processing

**Description** System A generates an image copy of a physical table within a logical file of System A with no additional processing logic and sends it to System B. System B loads the physical table without any additional processing logic.

**Scenario** System B requires (e.g., performance, etc.) the ability to access a portion of File X in System A for validation and reference only. System A sends a physical table within a logical file to System B. The existing view of that physical table in System B is "refreshed" each time with the copy.

**Diagram**



**Counting Interpretation** Since the data is an image copy of System A's data, the File X Table is part of the logical File X of System A. System B counts File X (with only the data elements used from the File X Table) as an EIF.

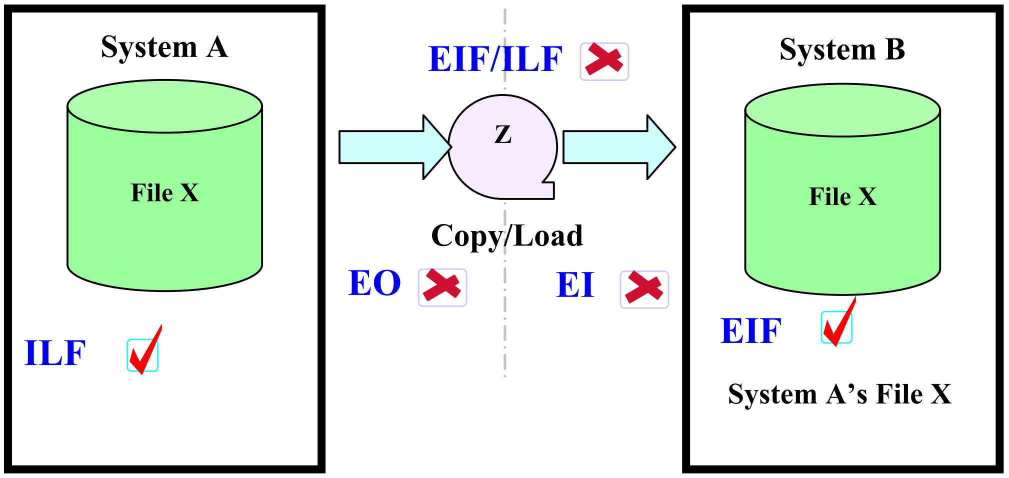
Logically the data store remains in System A. In this case copying the data store from one system to another is the solution of technical requirements; e.g., the data in System A is not available when System B requires it.

The *primary intent* is for System B to reference the data that logically exists in System A.

*Transactions* – Since there are no logical transactional functions in propagating or loading the copy, no transactions are counted for either system to support the copying and loading of the shared data. Therefore, System A does not count the copy to System B as an EO/EQ, and System B does not count an EI. One additional indication would be that the file in System B is "refreshed" each time with the copy.

*Files* - There is only one logical file involved. System A counts File X as an ILF. System B counts its copied table of File X as an EIF.

**Solution Diagram**



**Counting Summary**

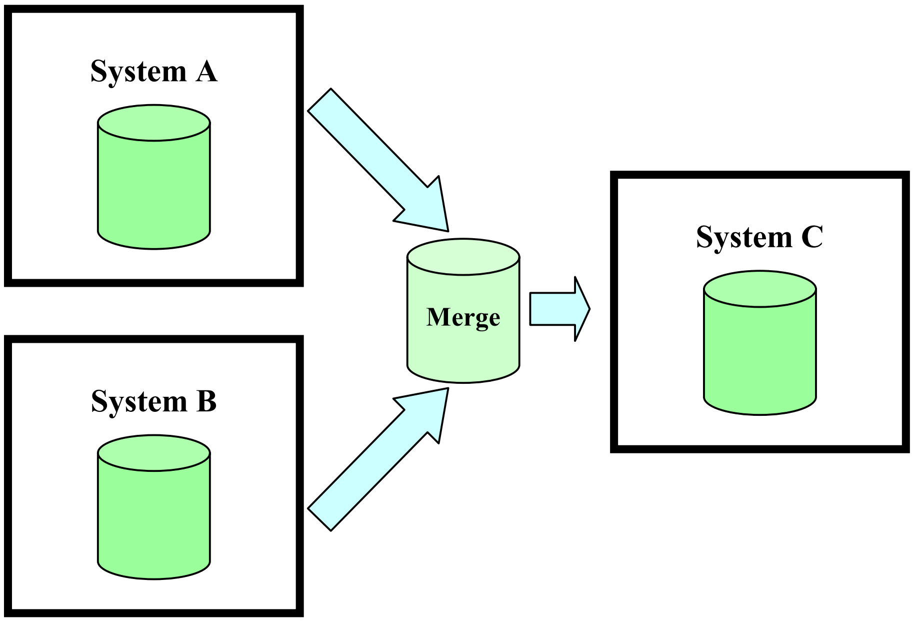
| | ILF | EIF | EI | EO/EQ | Note |
|---|---|---|---|---|---|
| System A | ☑ | | | | |
| System B | | ☑ | | | |

## Scenario 5: Copy and Merge

**Description** Data stored in two systems is image copied and merged to form one file that is loaded into a third system. Multiple files with the same data elements are being consolidated into a single file.

**Scenario** To avoid the overhead of System C having to dynamically search the data from both System A and B, the data is being copied from System A and System B and merged into a new data store in System C. The user requires that the information from System A and B be refreshed daily for validation or reference purposes only. Unload, Merge and Load utilities are used. There is no business processing logic involved. This is typically a technical solution where two applications have different instances of the same logical data required by a third application.

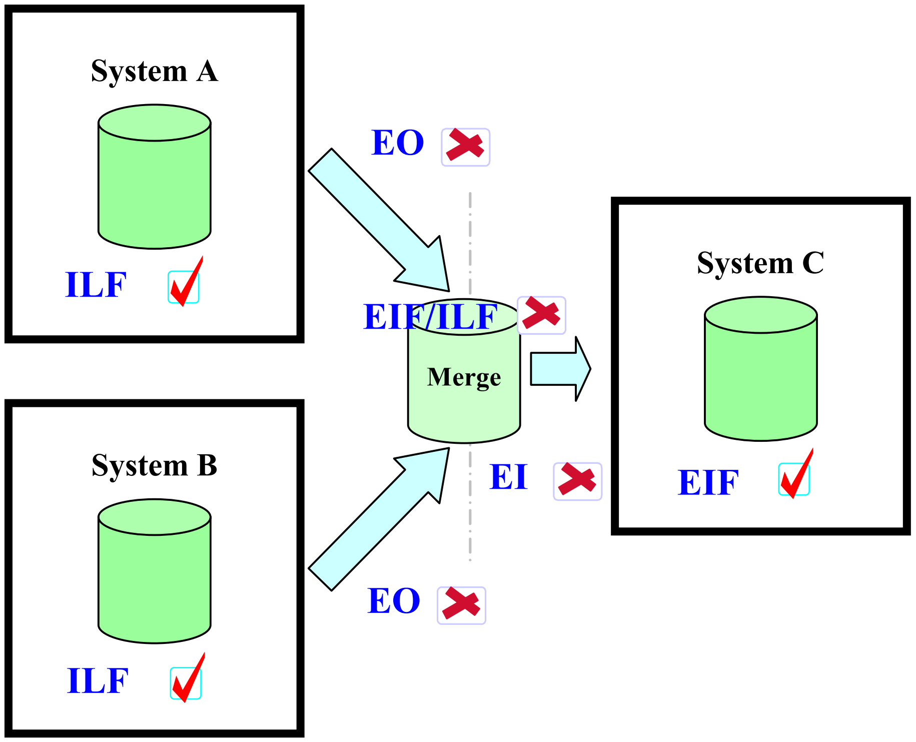
**Diagram**



**Counting Interpretation** Logically, the data stores remain in Systems A and B. Merging the data into one data store does not by itself create a new ILF for System C. Again, the primary intent for the usage of the data in System C must be evaluated. Since the data is to be used only for reference or validation and it is a complete refresh, it is counted as an EIF. Since there is no additional processing logic, no transactions are counted for any system.

The data for system C must be evaluated according to Part 1 – Data Functions and Part 2 - Logical Files. Even though the data comes from two different systems, the data elements are exactly the same (see definition for “merge”). Therefore only a single logical file is identified for System C as an EIF.

## Solution Diagram



## Counting Summary

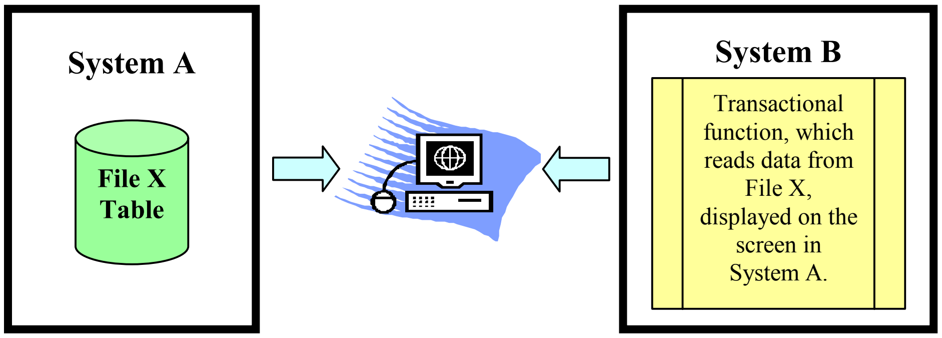
| | ILF | EIF | EI | EO/EQ | Note |
|---|---|---|---|---|---|
| System A | ☑ | | | | |
| System B | ☑ | | | | |
| System C | | ☑ | | | Also count as FTR in transactional function |

## Scenario 6: Screen Scraping

**Description** Accessing another application's screen transactions to reference/obtain data or to update that application's data.

**Scenario** System B 'reads' the content of an inquiry screen in System A and uses that data in the processing of a transactional function.

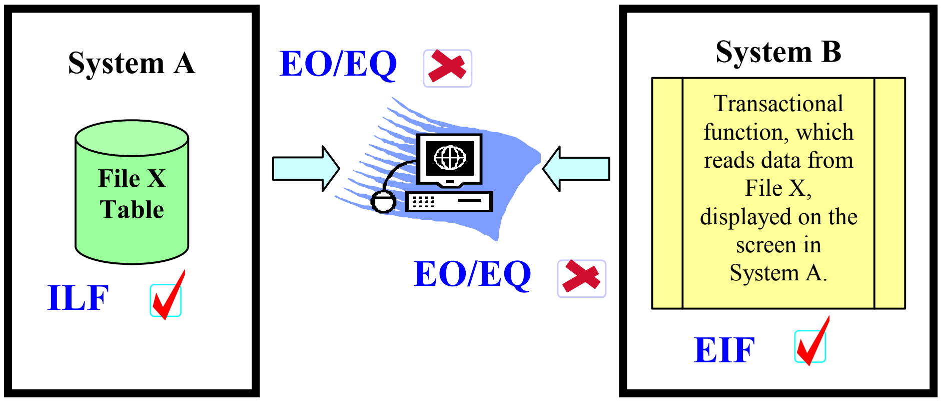
**Diagram**


System A
File X Table
System B
Transactional function, which reads data from File X, displayed on the screen in System A.

**Counting Interpretation** Logically, the data stores remain in Systems A and B. Merging the data into one data store does not by itself create a new ILF for System C. Again, the primary intent for the usage of the data in System C must be evaluated. Since the data is to be used only for reference or validation and it is a complete refresh, it is counted as an EIF. Since there is no additional processing logic, no transactions are counted for any system.

The data for system C must be evaluated according to Part 1 and Part 2, Chapter 2, Logical Files. Even though the data comes from two different systems, the data elements are exactly the same (see definition for "merge"). Therefore only a single logical file is identified for System C as an EIF.

**Solution Diagram**


System A
File X Table
ILF ✓
EO/EQ ✗
EO/EQ ✗
System B
Transactional function, which reads data from File X, displayed on the screen in System A.
EIF ✓

**Counting Summary**

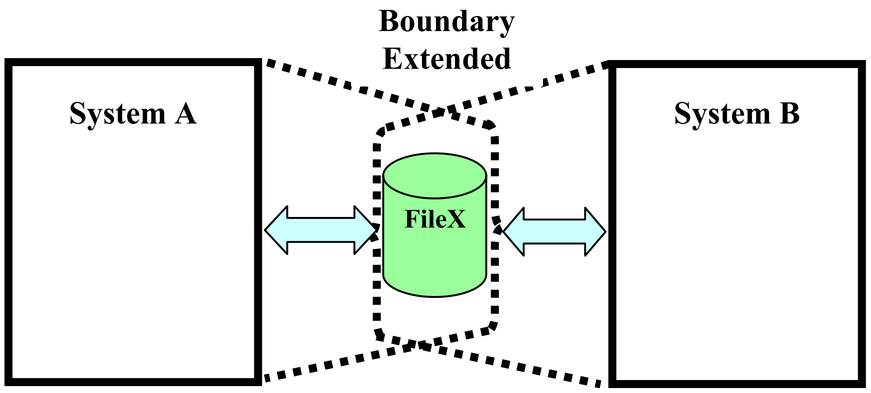
| | ILF | EIF | EI | EO/EQ | Note |
|---|---|---|---|---|---|
| System A | ☑ | | | | |
| System B | | ☑ | | | |

## Scenario 7: Updating the Same Data Store

**Description** The same data store is maintained by two different applications

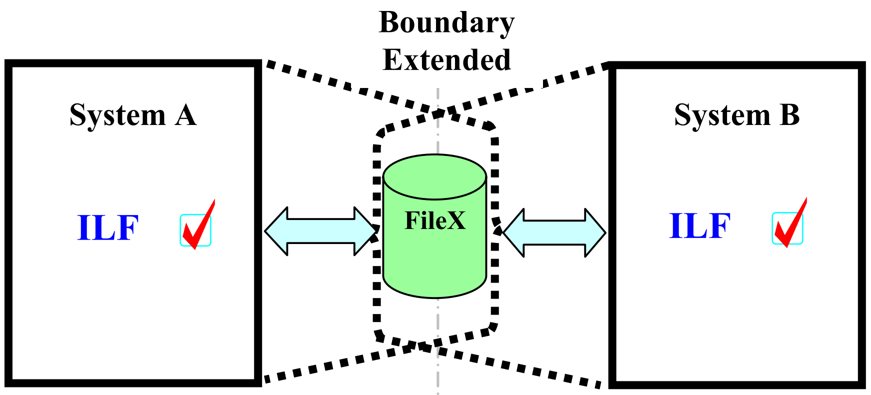
**Scenario** Both System A and System B maintain the same ILF. Each has its own unique view of the data. There are some common data elements, and some that are unique to each system.

**Diagram**



**Counting Interpretation** An ILF is counted for both systems because each has transactions to maintain it. System A and B both maintain data in the same ILF. Each system counts only the RETs and DETs maintained, used or referenced by that System.

**Solution Diagram**



**Counting Summary**

| | ILF | EIF | EI | EO/EQ | Note |
|---|---|---|---|---|---|
| System A | ☑ | | | | |
| System B | ☑ | | | | |

## Scenario 8: Standard Transaction Data

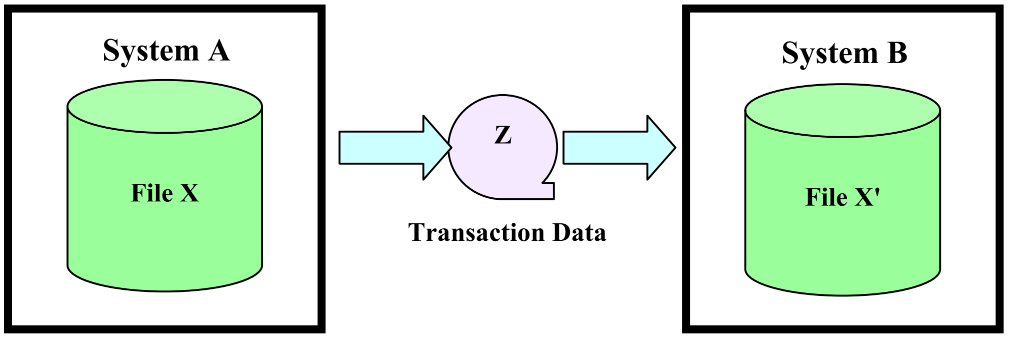
**Description** Transactional data is provided by the source system.

**Scenario** System A produces a transaction file of changes, File Z, that is loaded into System B. The records are usually of more than one type. System B processes the input transactions according to the transaction type on the File Z records, prior to updating the records on internal File X'. The DETs on System A File X, and System B File X', are different. For example File X is a Master Material Catalogue while File X' is a local Product List. Processing includes the following transaction types:

- Add
- Change
- Delete

This data transfer is a user business requirement. Both System A and B have a requirement to access a version of the File X however the DETs on the two files are different. System A sends data related to changes only. System B reads the records on File Z and based upon the transaction type initiates different logical processing.

**Diagram**

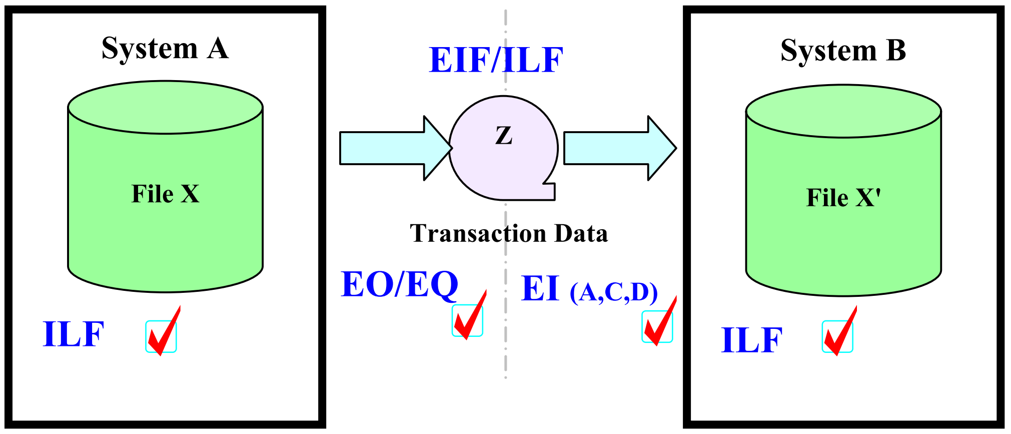


**Counting Interpretation** If every record written by System A to the File Z is processed in the same way only one EO/EQ is counted. Only when there is different logical processing involved you may have multiple transactional functions (e.g., EO/EQ) within a single file.

System B counts EIs for each unique maintenance function on File X'. The number of Transaction Types on the transaction File Z, usually determines the number of these functions, but this is not necessarily the case. Different logical processing must be demonstrated.

There are two files involved. System A counts File X as an ILF. System B counts File X' as an ILF. Neither system counts File Z as a logical file.

**Solution Diagram**



**Counting Summary**

| | ILF | EIF | EI | EO/EQ | Note |
|---|---|---|---|---|---|
| System A | ☑ | | | ☑ | |
| System B | ☑ | | ☑ | | |

## Summary

Throughout this chapter, the scenarios have focused on the use of the data within the application being counted and the application of the rules of ILF and EIF identification that reference ‘primary intent’.

This chapter does not illustrate all possible implementations of data sharing between applications. It does, however, provide enough examples that a Function Point Analyst can consistently apply the identification rules for ILFs and EIFs by focusing on the primary intent for the use of the data and the boundaries of the applications involved in the count.

This page intentionally left blank.

# Enhancement Projects and Maintenance Activity

**Introduction**

This chapter provides additional guidance for identifying and counting functional changes to installed applications. It does not address the relationship between the functional size of the enhancement and the effort required to implement that enhancement. For additional perspectives on this topic, the reader is also referred to the NESMA publication, "*Function Point Analysis for Software Enhancement*" [NESMA, 2001], at www.nesma.org.

This chapter also discusses the various maintenance and support activities that may occur during the productive life of an application and for which functional size measurement provides a useful basis for estimation and cost reconciliation.

**Contents**

This chapter includes the following sections:

## Counting Enhancement Projects

An Enhancement Project Count measures the modifications to the existing installed application that add, change or delete user functions delivered when the project is complete. The changes in functionality, i.e., changes in processing logic, might result from new or revised user requirements, statutory/regulatory changes or new users, as opposed to maintenance or other activities

### Scope and Boundary of an Enhancement Project

The enhancement function point count includes all functions being added, changed and deleted. The boundary of the application(s) impacted remains the same. The functionality of the application(s) reflects the impact of the functions being added, changed or deleted.

Note: There may be more than one application included in the counting scope. If so, multiple application boundaries would be identified, resulting in a separate enhancement count for each affected application.

If the total size of the enhancement project is required, it is calculated by summing the enhancement counts for all applications included in the counting scope.

### Counting Data Functions in Enhancement Projects

The additions of new internal logical files or external interface files by an enhancement project are generally easily identified and counted according to rules defined in Part 1. However, consideration should be given to the following:

- If the change involves only the addition of new records to a logical file or new values in an existing field within that logical file, there is no justification to count the data function as being changed.
- If a data function is changed because a field is added and that field is not used by the application being counted, then there is no change to that application.
- In order for a data function to be counted as a changed function, the general guideline is that it must be structurally altered (e.g., adding or removing a field or changing the characteristics of the field).

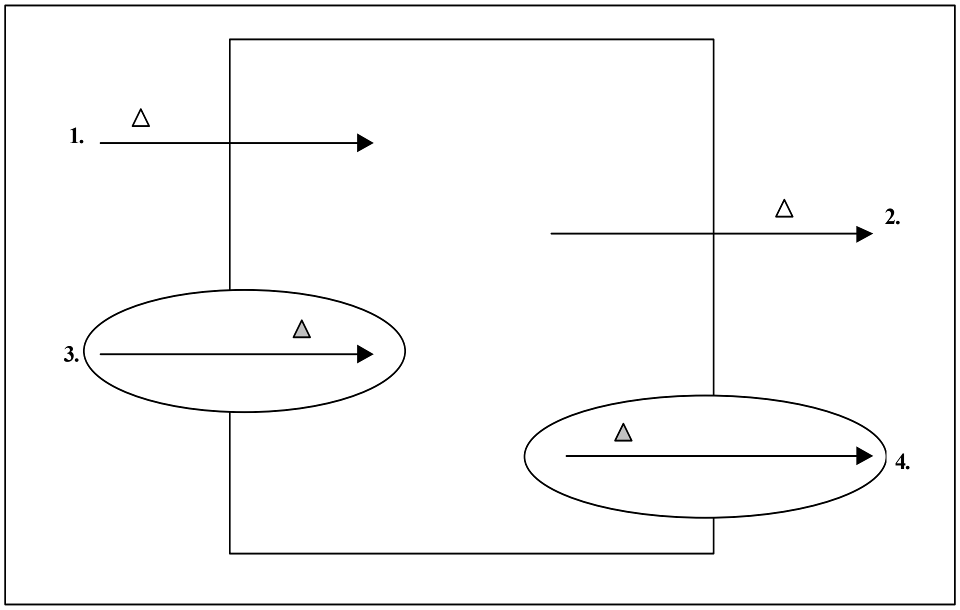
- If an application is required to use (reference or maintain) an existing field that it did not previously use, then the related data function is considered changed for that application. This can occur without any physical changes to the file.
- If new fields are added to an ILF, look for new or modified transactional functions that maintain the field in that ILF to confirm that a change has occurred.
- If a field is added to an ILF that is maintained by two applications and if one application maintains the new field but the other only references it, then both applications take credit for the changed ILF. However, the second application will not have any new or changed transactional functions that maintain the field in that ILF.
- If an application neither maintains nor references a new or changed field, then it cannot take credit for a changed data function.
- If a physical file is added by an enhancement project, it does not necessarily result in a new logical file. We first need to determine whether the new physical file is a change to an existing logical file with additional DETs and possibly an additional RET, or a new logical file. See Chapter 2 of Part 2 for additional information regarding counting logical files.

## Counting Transactional Functions in Enhancement Projects

The identification of Transactional Functions that have been changed by adding or removing Data Element Types is obvious. It is not as obvious when the user requirements are for changes in processing logic, as described in Part 1, Chapter 7. When processing logic has been altered within an application to meet business requirements, the elementary process that embodies that logic should be identified and counted as being changed.

Note that a single change in processing logic does not always effect all related transactions.

For example, when an edit or validation change is made to input processing logic and Add, Delete, Update and implied Inquiry transactions exist, then only the Add and Update transactions are counted for the enhancement. Unless there is specific deletion logic change (e.g., referential integrity edit) or query logic change (e.g., selection or retrieval), the Delete and Implied Query transactions will not change.



In the above diagram:

1. Transaction 1 is modified because an additional DET has been added and is crossing the boundary.
2. Likewise, Transaction 2 where an additional DET is sent across the boundary will be counted as changed.
3. For Transaction 3, a validation routine internal to the application has been changed. Since this is a change to the processing logic within an elementary process, the associated Transaction 3 is counted as being changed.
4. Similarly for Transaction 4 where the previous selection criteria or filter has been modified, the transaction is counted as changed functionality.

In some cases, a specific change may impact the way multiple transactional functions are processed. Regardless of whether logic changes were physically made in a common routine used by multiple transactions, changed functions should be identified based on the elementary processes that embody that logic. If multiple elementary processes were affected, then count multiple transactional functions. If only a single elementary process was affected, count one transaction as being changed. In all cases, the user requirements and the business view shall be the determining factor.

For example, the requirement is to modify the edit on new orders to include a validation for the customer's past due balance. If there are several different order transactions (Individual, Commercial, Government, etc.) but the edits for only the Commercial order transaction were modified, then only the Commercial order transaction should be counted as changed. Alternatively, if the requirement was to modify all these type of orders, then each separate transaction would be counted as being changed.

Another indication of the transactional functions to be counted would be the coverage of test cases. A single set of test cases would indicate that a single elementary process has been modified.

Often the nature of a change is related by the developer, claiming that the one module being changed is used in the production of a large number of reports or extracts, or the processing of many input transactions. All the functions may use the common routine, but only a subset of those functions use the edit that is being changed in that common routine. Only the functions that incorporate the changed processing logic would be counted as changed. The challenge is to properly assess the appropriate level of *functional change*. The emphasis must be on the business requirements, with the count reflecting the intent of the user request.

## Processing Logic

*Processing logic* is defined as requirements specifically requested by the user to complete an elementary process. Those requirements may include the following actions:

1. Validations are performed

   For example, when adding a new employee to an organization, the employee process has processing logic that validates the information being added.

   - If a requirement exists to perform a different validation or change the validation in an existing transactional function, the transaction would be counted as changed in the enhancement count.

2. Mathematical formulas and calculations are performed

   For example, when reporting on all employees within an organization the process includes calculating the total number of salaried employees, hourly employees and all employees.

   - If a business requirement exists to modify an existing calculation (e.g., before, the formula was A + B = C and now C = A * B), the transaction that includes that calculation would be counted as changed in the enhancement count.
   - Currently there is a list of employees that is counted as an EQ. The enhancement project requirements state to display summary counts of those employees. The transaction would be identified as changed and the function type would be changed from an EQ to an EO in the enhancement count.

3. Equivalent values are converted

   <u>For example</u>, an elementary process references currency conversion rates from US dollars to other currencies. The conversion is accomplished by retrieving values from tables, so calculations need not be performed.

   - If a business requirement exists to change the functionality to include the ability to convert other currencies to Euro, the transaction would be counted as changed in the enhancement count.

4. Data is filtered and selected by using specified criteria to compare multiple sets of data

   <u>For example</u>, to generate a list of employees by assignment, an elementary process compares the job number of a job assignment to select and lists the appropriate employees with that assignment.

   - If a requirement exists to modify the selection criteria or add additional selection criteria to an existing transaction (a list of employees now needs to display a list of employees by assignment and location), the transaction would be counted as changed in the enhancement count.

5. Conditions are analyzed to determine which are applicable

   <u>For example</u>, processing logic exercised by the elementary process when an employee is added and will depend on whether an employee is paid based on salary or hours worked.

   - If a requirement exists to modify the condition or add additional conditions to an existing transaction, the transaction would be counted as changed in the enhancement count.

6. One or more ILFs are updated

   <u>For example</u>, when adding an employee, the elementary process updates the employee ILF to maintain the employee data.

   - If a business requirement results in updating an additional ILF or different DETs by an existing transaction, the transaction would be counted as changed in the enhancement count.

7. One or more ILFs or EIFs are referenced

   <u>For example</u>, when adding an employee, the currency EIF is referenced to use the correct US dollar conversion rate to determine an employee's hourly rate.

   - If a business requirement results in referencing additional ILFs, EIFs or DETs in an existing transaction, the affected transaction would be counted as changed in the enhancement count.

8. Data or control information is retrieved

   For example, to view a list of possible pay grades, pay grade information is retrieved.

   - If the business requirement results in retrieving additional information in an existing transaction, the affected transaction would be counted as changed in the enhancement count.

9. Derived data is created by transforming existing data to create additional data

   For example, to determine (derive) a patient's registration number (e.g., SMIJO01), the following data is concatenated:

     - the first three letters of the patient's last name (e.g., SMI for Smith)
     - the first two letter of the patient's first name (e.g., JO for John)
     - a unique two-digit sequence number (starting with 01)

   - If the business requirement results in changing how the transaction derives the data, the affected transaction would be counted as changed in the enhancement count.

10. Behavior of the system is altered

    For example, the behavior of the elementary process of paying employees is altered when a change is made to pay them every other Friday versus on the 15th and the last day of the month.

    - If the business requirement results in altering the behavior of the system (e.g., in the example above, the transaction is changed so that the pay date parameter affects only hourly employees not all employees), the affected transaction would be counted as changed in the enhancement count.

11. Prepare and present information outside the boundary

    <u>For example</u>, a list of employees displayed for the user.

    - When the business requirement results in presenting additional data outside the boundary, the affected transaction would be counted as changed in the enhancement count.

      Changes to literals, format, color or other elements of the physical presentation are not considered changes to the processing logic, and therefore not part of an enhancement count.

12. Capability exists to accept data or control information that enters the application boundary

    <u>For example</u>, a user enters several pieces of information to add a customer order to the system.

    - When the business requirement results in different pieces of information that enter the boundary (e.g., DETs), the affected transaction would be counted as changed in the enhancement count.

13. Data is resorted or rearranged

    <u>For example</u>, a user requests the list of employees in alphabetical order.

    **Note:** Resorting or rearranging a set of data does not impact the identification of the type or uniqueness of a transactional function.

    - When the business requirement results in changing the existing sort sequence (e.g., user now requests the above referenced list of employees in location order instead of alphabetical order), the affected transaction would be counted as changed in the enhancement count.
    - The user requests an additional report of the same data (list of employees) sorted by location. A new transaction would not be counted, but a change to the existing transaction would be included in the enhancement count. Resorting or rearranging a set of data does not impact the identification of the type or uniqueness of a transactional function, but it does constitute changes in processing logic.

One elementary process may include multiple alternatives or occurrences of the above actions. <u>For example:</u> validations, filters, resorts, etc.

## Considerations and Hints

Many enhancement projects involve changes only in processing logic with no physical changes to inputs, outputs or files (e.g., added or removed fields). The following are items or questions to discuss with developers in counting enhancements to installed applications:

### Questions to Consider for an Enhancement Project

The questions listed below can be used during the interview with the developer or Subject Matter Expert (SME) during function point counting sessions. Positive responses to the questions listed below indicate possible changes to user functionality. Additional investigation is required to determine whether and how they should affect the function point count.

**What NEW functionality has been created for the users of the application?**

- Have any new permanent user files, databases, tables, entities or objects been developed/added by this project?
- Is the application now receiving and processing new input transactions or feeds/transaction files that were not previously in production?
- Have any new screens been built for the users?
- Are there any new interactions (interfaces) with other applications?
- Is the application generating new outputs, new reports or new files for another system?
- Has a new record type been added to an existing transaction file or feed?
- Have there been any new query processes established for the user?
- Have any batch processes been added?

**What user functionality has been CHANGED/MODIFIED due to user requirements?**

- Have any existing permanent files, databases or tables had fields or columns added or removed? Have the attributes of any existing fields or columns been modified? Adding new values to an existing field or new rows to an existing table does not count.
- Has any existing processing logic for input screens or inputs received via files sent by users or other applications been modified?
- Have there been any changes to the actual input screens or the feeds/transaction files from other applications (e.g., new fields, changes to

attributes of existing fields)?

- Have any existing user reports, output files or screens been modified by adding or removing fields?
- For existing outputs, have there been any changes to the processing logic to produce these files, reports or screens?
- Have any screens for query purposes been modified?
- Have there been any changes to the edit or selection criteria or filters behind any query screens?
- Have any batch processes been changed?

**What user functionality has been DELETED/REMOVED due to user requirements?**

- What user functionality has been deleted or removed from production?
- Are any existing permanent files no longer required?
- Has the application stopped accepting an input file from another system or removed a screen previously accessed by a user?
- Have any reports been removed because the user doesn't require it anymore?
- Have any output files to another system been eliminated?
- Has any user query been removed?
- Have any batch processes been removed?

**What Conversion Functionality has been provided?**

- Has there been any one-time data processing to cleanup, organize or to populate new fields as a result of structural modifications to a permanent file?
- Has there been any requirement to populate any new permanent file?
- Are there any user requested conversion reports?

**What other changes were made to the application for this project?**

- This wrap-up question may prompt the Subject Matter Expert or developer to provide additional clues to countable changes.

## General Systems Characteristics

The 14 General System Characteristics (GSCs) should be reviewed for change. Small enhancements do not normally require such a review. Examples of changes that may indicate a need to review the GSCs include:

- Addition of on-line functions to a batch application
- Increased transaction volumes and/or degraded response times now requiring performance design and testing activities
- Additional usability features requested
- Addition of a Web interface to an existing on-line application
- Addition of a new communication protocol to an existing application

## Summary of Considerations and Hints

These guidelines will assist in determining the functionality delivered by the project and its impact on the baseline of the application being counted. The combination of new functionality added, the effects of the changes made to existing functionality and user functionality deleted will be used to size the Enhancement Project as well as to calculate the updated Function Point Baseline count for the application.

## Inputs for Counting Enhancement Projects

In general, the following items are useful for any count:

- Requirements document(s)
- Entity relationship diagrams
- Object models
- Data models
- File and database layouts (with logical, user required fields identified)
- Interface Agreements with descriptions of batch feeds/transaction files and interfaces to/from other applications
- Samples of reports, on-line screens and other user interfaces
- Demonstration of application operation
- One or more application experts (for the application being counted)
- One or more customers/application users (on call during the counting session)
- User guide or training manuals
- System design documentation

In addition, for an enhancement project, the following should also be provided:

- Existing Application Function Point Count documentation
- User Requirements for any changes/modifications to the existing application
- Existing screen layouts, report layouts and/or batch job data flow diagrams for all affected functions as they were BEFORE the enhancement project
- Revised screen layouts, report layouts and/or batch job data flow diagrams for all affected functions as they will be AFTER the enhancement project
- Documentation for all new functions (new reports, outputs, data entry functions or entities) that are to be added to the application as part of the enhancement project
- Screen layouts, report layouts and/or batch job data flow diagrams for functions that are to be REMOVED from the application as a result of the enhancement project

It is recognized that, in many instances, all of the above documentation may not be available or applicable. The most important source is a knowledgeable application expert and the Requirements documentation.

If an application baseline count does not exist, care should be given in establishing the boundary and in identifying the logical data and transactional functions.

## Procedure

The following are suggested steps for performing an enhancement count.

| Step | Action |
|---|---|
| **1** | Gather and review available documentation. |
| **2** | Meet with the subject matter expert to discuss changes planned/made. |
| **3** | Identify and evaluate added functionality. |
| **4** | Identify and evaluate deleted functionality. |
| **5** | 1. Identify and evaluate changed functionality: <br>• Determine complexity of the function prior to change (from previous count documentation or count as existing prior to change). <br>• Determine complexity of the function after change. |
| **6** | Identify and evaluate any conversion or one-time functionality required to implement this enhancement. |
| **7** | Evaluate any changes to the General System Characteristics. |
| **8** | Calculate the Enhancement Function Point Count (EFP) <br>EFP = [(ADD + CHGA + CFP) * VAFA] + (DEL * VAFB) |

In order to compute the new application count sum all of the changes "before" and apply the following calculation:

Application Function Point Count (AFP)
AFP = [(UFPB + ADD + CHGA) - (CHGB + DEL)] * VAFA

## Enhancement Project Count

To illustrate some of the counting situations discussed in this chapter we would like to expand upon the example presented in Part 1, Chapter 9. The additional examples, shown in *italics,* illustrate changes in processing logic.

**Example**

This section shows an example for a sample enhancement project. The requirements for the enhancement project include the following changes:

- The user needs to receive an additional report about jobs that includes totals.
- Additional DETs are required to add jobs in batch and correct suspended transactions. A reference to security is also added for the "add job" transaction.
- *The Job Assignment function must now verify that the job type assigned to an employee matches the employee's classification, which is maintained by the Employee Relations System.*
- *The report of Employees by Assignment Duration must be changed. Instead of showing a count of employees over 12 and 24 months, the report must now show only salaried employees.*
- The user no longer needs to add a job online; therefore, that functionality is to be or was removed

**Application Functionality**

The following paragraphs explain the application functionality counted for the example enhancement project. Functionality is described as added, changed, or deleted.

**Added Functionality**

The following table shows the functional complexity for the added functionality counted when the project was completed.

**Note 1:** Providing a new report was an additional external output.

**Note 2:** The ER data referenced by the HR application is identified as an EIF.

| Data Functions | RETs | DETs | Functional Complexity |
|---|---|---|---|
| **External Interface File** | | | |
| Employee Relations Data | 1 | 2 | Low |

| **Transactional Functions** | **FTRs** | **DETs** | **Functional Complexity** |
|---|---|---|---|
| **External Output** | | | |
| Job report | 1 | 15 | Low |

## Changed Functionality

The following table shows the functional complexity for the changed functionality, as the functions will exist after the enhancement project is completed.

**Note 1:** The complexity for adding a job was increased because of the additional file type referenced. The complexity for correcting suspended transactions remained low.

***Note 2:*** *While an additional File Type Referenced is counted, there is no change in complexity for Adding a Job Assignment because it is already High.*

***Note 3:*** *Although there is a change in the selection criteria and the summation logic, there is no change in complexity of the report*

| **Transactional Functions** | **FTRs** | **DETs** | **Functional Complexity** |
|---|---|---|---|
| **External Input** | | | |
| Add job information (batch input) | 3 | 8 | High |
| Correct suspended transaction | 1 | 8 | Low |
| *Add Job Assignment* | *4* | *7* | *High* |
| ***External Output*** | | | |
| *Employees by Assignment Duration* | *3* | *7* | *High* |

## Deleted Functionality

The following table shows the functional complexity for deleted functionality identified at the end of the project.

| **Transactional Functions** | **FTRs** | **DETs** | **Functional Complexity** |
|---|---|---|---|
| **External Inputs** | | | |
| Add job information (screen input) | 1 | 7 | Low |

## Enhancement versus Maintenance Considerations

Once an application has been developed and installed, it must then be maintained (modified) in order for it to continue to meet the needs of an ever-changing business and technical environment. This maintenance includes a wide range of activities that are performed during this phase of the application life cycle, some of which involve functional changes that are applicable to FPA.

### Categories of Maintenance

The Institute of Electrical and Electronics Engineers, Inc. (IEEE) defines three categories of maintenance:

**Adaptive Maintenance:** Software maintenance performed to make a computer program usable in a changed environment.

**Corrective Maintenance:** Software maintenance performed to correct faults in hardware or software.

**Perfective Maintenance:** Software maintenance performed to improve the performance, maintainability or other attributes of a computer program.

While the body of this chapter has provided Function Point Counting hints and guidelines for enhancements to existing applications, there is no industry-wide standard for consistent classification of activities that fall within the above categories. This section provides a framework based on common industry experience from which to evaluate the applicability of FPA in the support of installed applications.

#### Adaptive Maintenance

Adaptive Maintenance includes modifications to either meet new or changing business requirements or to add functionality not accommodated in a previous release. It also may include modifications required to meet changing technical requirements. Adaptive maintenance is initiated by business requests to add, change and/or delete business functionality. It is synonymous with the concept of an “enhancement”, as defined in Part 1.

#### Corrective Maintenance

Corrective Maintenance includes modifications to repair defects. It does not involve changes to business functionality but ensures that previously delivered functionality performs as required. The effort related to these activities should be attributed to the original development or enhancement project that introduced the defect.

#### Perfective Maintenance

Perfective (or Preventive) Maintenance may include modifications to support platform or system software upgrades, performance optimization and other activities related to maintaining agreed service levels. There are no changes to business functionality associated with this work. Although Function Point

Analysis is not useful for estimating these activities, the GSCs may be affected and should be reviewed for changes.

Functional size measurement quantifies the size of business requirements. In an enhancement environment, it measures the effects of changes to those business requirements. Therefore, functional size measurement is applicable to a subset of adaptive maintenance. This includes the software functionality added, changed or deleted as well as the software functionality provided to convert data and meet other implementation requirements (e.g., conversion reports).

It is impractical to provide a complete and comprehensive list of development and support activities. Rather, the following areas are identified with suggestions regarding FPA applicability. It is strongly recommended that each organization develop its own guide with activities, definitions and terminology specific to that organization.

## Application Maintenance and Support Activities

Since maintenance and support activities are subject to inconsistent reporting, locally developed guidelines should address these areas. The following are some of the more commonly encountered activities, with suggested handling relative to FPA.

### Maintenance Requests

Regardless of duration or level of work effort required, it is the type of activity that determines how the work is classified. Function Point Analysis should not be used to size perfective or corrective maintenance work. Corrective maintenance should be charged to the development or enhancement project that introduced the defects. Perfective maintenance should not be charged to any development or enhancement projects.

There may be a tendency to track some enhancement functionality as maintenance work, but that work should be monitored and reported separately. The usual rationale for inclusion is for either immediacy or expediency. Organizations often provide a fast path for small enhancement requests, usually 40 hours or less, in order to reduce the overhead burden on the project. When business requirements are affected, Function Point Analysis should be applied at least for results measurement.

If a release contains a mix of adaptive, corrective and/or perfective maintenance requirements, care must be exercised in separation of work effort, since the latter two categories contribute zero function points to the business. While such work effort segregation may be relatively easy during the construction phase, depending on the level of granularity in effort tracking, it is generally more difficult during most final test phases. One possible approach would be an apportionment of the entire release based on proportional content.

A project involving only upgrades from one platform, language or technical environment to another, with no change in user functionality, should not be subject to an enhancement function point count.

| Activity | Within Enhancement Counting Scope |
|---|---|
| Correction of Production Errors (“break/fix”) | No |
| Perfective or Preventative Maintenance | No |
| Platform Upgrades, New System Software Releases | No |
| Project with both fixes and enhancements | Partially |

## On-Demand (Ad Hoc) Requests

Functionality that is provided to the end user in the form of one-time/on-demand reports and data extracts is certainly countable. The decision to count should be made based on whether the functions will be maintained and the business need that the function point count will meet. It must be noted that this discussion is limited to reports/extracts produced by I/S Development and does not cover user generated Ad Hoc reports or queries. It should be noted that the methodology to produce ad hoc reports is usually not as rigorous as for a full enhancement project. Therefore, care should be taken when comparing the relative costs of such work with those of general enhancement activity.

| Activity | Within Enhancement Counting Scope |
|---|---|
| One-time Reports | Local Convention |
| Table Updates | No |
| Special Job Setup | No |
| Data Correction | No |
| Mass Data Changes | Yes – as conversion if associated with a project. |

## End User Support

Any non-project work effort related to activities classified as “not countable” should be charged to a labor classification other than New Development or Enhancement. For Preliminary Estimation or Feasibility Studies, the problem is that user requirements are not yet well defined. Also, a project at either stage is usually not yet funded (and may never be funded). At best, a Rough Order of Magnitude (ROM) function point estimate can be determined, but no quantitative measurements should be applied at this point. Any resulting number is for budget and planning purposes only.

General non-project user support activities, such as answering “what-if” questions and helping users, should not be subject to Function Point Analysis.

| Activity | Within Enhancement Counting Scope |
|---|---|
| Preliminary Estimation or Feasibility Analysis | At best, ROM |
| Answering "What If" questions | No |
| General Non-Project Client Support | No |
| Help Desk Support | Partially |

## Conclusion

This chapter is intended only to provide guidance and is not to be interpreted as a set of absolute rules. Exceptions to these scenarios can always be found. Each instance must be evaluated based on the user requirements and business situation. Additionally, each IT organization has its own unique activities and development processes. These should be addressed in a local Function Point Guide, which addresses all facets of Function Point Analysis, including counting assumptions and any unique interpretations of CPM rules.

## Summary

While a major portion of IT Application Development resources are usually devoted to supporting or enhancing existing applications, most examples and illustrations within Part 3 are geared toward new development. This chapter provides guidance on what to count and insight into the identification process in an enhancement environment.

The first step is to correctly identify those Data and Transactional Functions that have been added, changed or deleted based on user requirements. For modifications, the most frequently overlooked area is change involving only internal processing logic. Part 1, Chapter 7 includes definitions of processing logic which is helpful in identifying modifications to transactional functions that should be counted. Lastly, each identified Data and Transactional Function must be counted in accordance with the rules in Part 1.

It is also important to segregate the different maintenance and support activities and their associated effort as discussed in the Enhancement Versus Maintenance Considerations Section.

# Index to Part 2