INTERNATIONAL STANDARD

ISO/IEC 24570

Second edition
2018-02

Software engineering — NESMA functional size measurement method — Definitions and counting guidelines for the application of function point analysis

Ingénierie du logiciel — Méthode de mesure de la taille fonctionnelle NESMA — Définitions et manuel des pratiques de comptage pour l'application de l'analyse des points fonctionnels

Reference number
ISO/IEC 24570:2018(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by NESMA and was adopted, under the PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, in parallel with its approval by national bodies of ISO and IEC.

This International Standard is the latest release in the continually improving Nesma method. This method is a consistent interpretation of functional size measurement in conformance with ISO/IEC 14143-1:2007. The Nesma functional size measurement method is known as Function Point Analysis (FPA)[1)] and the unit of functional size is called Function Point.

This second edition cancels and replaces the first edition (ISO/IEC 24570:2005), which is now obsolete. Functional size measurements as determined based on this new edition of the standard are identical to those based on the previous edition of the standard. Results obtained in the past do not need to be updated.

1) In this document the abbreviation FPA is used for the term Function Point Analysis.

Introduction to this Standard

Reason for this International Standard

Over the years a number of "dialects" have arisen for function point analysis. These dialects complicate the goal of determining the number of function points and make it almost impossible for organizations to compare results. One insufficiently acknowledged reason for this is that different interpretations of the "Albrecht" method have arisen.

This International Standard provides clarity by formulating standards for the definitions and counting guidelines that pertain to FPA.

Intended audience

This International Standard is meant for everyone who performs function point analyses. It is assumed that the reader has some knowledge of function point analysis. Nevertheless, we have attempted to produce an International Standard that is both complete and includes sufficient introductory material and explanation for the new user.

Application of this standard in practice

This International Standard is one component in the Nesma publications. It is recommended that it be read in conjunction with the other Nesma publications. These provide guidance to application of the rules specified within this International Standard and background information to aid in understanding the use and applicability of the resulting functional size. Supporting Nesma publications include the following:

— Examples to illustrate the use of the Nesma method in specific situations and a fully documented Hotel case.

— Nesma website at nesma.org which contains a number of documents that can be used in a specific context, for example guidelines how FPA can be used in a Data Warehouse environment, with UML documentation, or different aspects in contracts.

Organization of this International Standard

Clause 1 describes the scope of this International Standard.

Clause 2 provides an introduction to FPA and in which the functional aspect of FPA is emphasized. It will also spell out briefly what FPA is and explains the terms that form the basis for the concept of FPA. Matters such as distinguishing between an application function point analysis and a project function point analysis are examined, just as are other various types of function point analyses, the role of FPA during a project, users, and function point analysis.

Clause 3 provides an overview of the position of FPA in a project and explains the types of function point analyses that can be carried out during the life cycle of an application. In other words, the clause explains when FPA can be applied and what information is needed minimally in order to count. The clause will also give a step-by-step plan for performing a function point analysis and indicates how projects, applications, and packaged software should be counted. Each of these requires their approach.

Clause 4 states general counting guidelines for a function point analysis.

Clauses 5, 6, 7, 8 and 9 successively give the definitions and guidelines used to identify function types and to determine the complexity of function types for internal logical files, external logical files, external inputs, external outputs, and external inquiries. The guidelines are broken down per function type for identifying the function type concerned, for determining the number of data element types, and for determining the number of record types or referenced logical files.

Annex A is meant to be a short summary of the guidelines and contains the most important features of each function type, as well as the tables for valuing the function types.

Annex B contains the definitions of the terms in this International Standard.

Annex C describes the mechanisms behind the increase in functional size.

This International Standard has been set up in such a way that the reader does not necessarily have to start at Clause 1 before continuing on to Clause 2, then 3 and 4 etc. Instead, the reader can look up what is important to him. For one reader, specific counting guidelines for a particular function type may be important, while someone else may want a more general frame of reference for an initial introduction to FPA.

Software engineering — NESMA functional size measurement method — Definitions and counting guidelines for the application of function point analysis

1 Scope

1.1 Purpose

This International Standard specifies the set of definitions, rules and guidelines for applying the Nesma Function Point Analysis (FPA) method.

1.2 Conformity

This International Standard is conformant with all mandatory provisions of ISO/IEC 14143-1:2007.

1.3 Applicability

This International Standard can be applied to all functional domains.

1.4 Focus

The International Standard focuses on how the functional size of an application is determined. The International Standard does not go into any of the aspects that play a role when project budgets are established on the basis of this functional size (e.g. productivity standards and productivity attributes).

The figure below indicates what this International Standard *will* and *will not* cover.

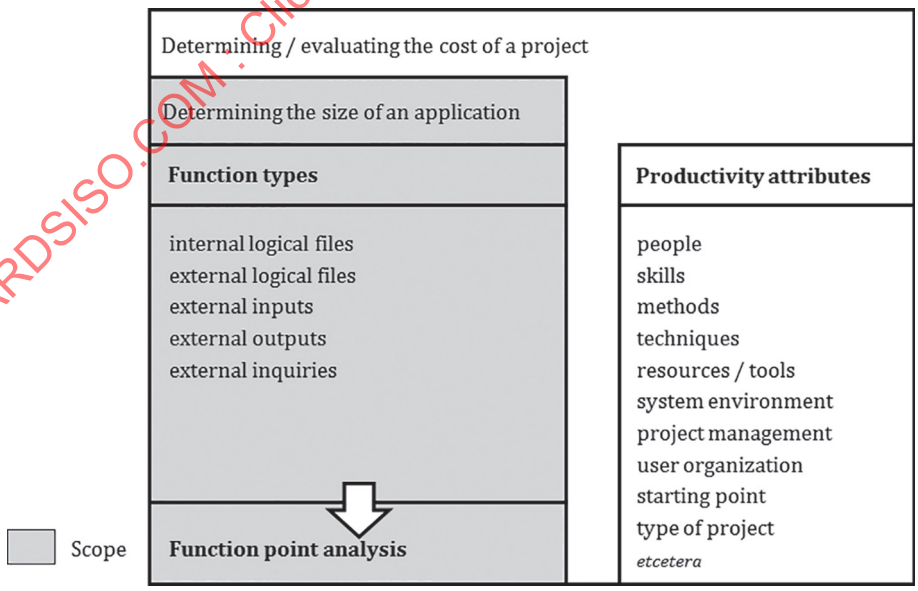


Figure 1 — Scope of the International Standard

2 Introduction to FPA

This clause gives a short description of FPA and explains a number of important concepts related to it. More specifically, subclause 2.1 provides a brief synopsis of FPA. Subclauses 2.2 through 2.4 distinguish between the different types of function point analyses. Subclauses 2.5 through 2.9 discuss each of the following successively within the context of FPA:

— The boundaries for an analysis

— Users

— Function types

— The complexity of a function type

— The valuing of function types

Subclause 2.10 defines the term *functional size* and describes how it is determined.

2.1 Brief description of FPA

2.1.1 Background, purpose and application of FPA

FPA was developed by A.J. Albrecht at IBM between 1974 and 1979 as a result of productivity research into a large number of projects. The first release of FPA was introduced in 1979, followed by adaptations based on practical experiences in 1983 and 1984.

FPA introduces a unit, the function point, to help measure the size of an application that is to be developed or maintained. The word "application" within the framework of FPA means "an automated information system". The function point expresses the quantity of information processing that an application provides to a *user*. This unit of measurement is separate from the way in which the information processing is realized in a technological sense. A function point is an abstract term and can be compared somewhat to so-called "rental points". Rental points are based on the number of rooms in a house, the surface area of these rooms, the number of facilities the house has, and the location of the house. This then serves as a measurement for a residence offered to a potential tenant.

FPA was first used to *measure the productivity* of system development and system maintenance after an application was built. It soon became clear that the technique could also be used to support *project budgeting* because the data needed for an FPA can be made available early on in a project.

2.1.2 Rationale behind FPA

The three separate words that make up the term "Function Point Analysis" can be used to explain the way of thinking behind FPA.

Function

As mentioned earlier, FPA bases itself on the functionality that an application provides to a *user* (see subclause 2.6). Because users see only the "outside" or the *boundary* (see subclause 2.5) of an application, FPA examines the specifications that describe the application's exchange of information with its environment. Functionality is derived from incoming and outgoing information flows (these can be both data or control information), as well as from the logical files that an application contains or uses. The functionality of an application is measured by identifying data functions and transactional functions (see subclause 2.7).

Point

The *complexity* of a function type is determined according to certain standard guidelines (see subclause 2.8). Each function is worth a number of points, depending on its complexity (subclause 2.9). The sum of these points yields the functional size (see subclause 2.10).

Analysis

FPA is the analysis of an application or the analysis of the description/specification of an application in order to establish its functional size. The act of establishing the functional size of an application or project is therefore called *function point analysis*.

In order to be able to perform a function point analysis the following must first be determined:

— purpose of the function point analysis (subclause 2.2);

— scope of the analysis and boundaries of the application or project to be analyzed (subclause 2.5).

This concludes a summary of the methodology and a brief description of FPA. The subclauses that follow explain the various terms used in FPA.

2.2 Use of FPA: *application* versus *project* functional size

Functional size can be linked to applications or to projects. This means that a distinction is made between the following two objectives:

— Determining the functional size of an application.

— Determining the functional size of a project.

Application functional size

is the number of function points that is a measure for the amount of functionality that an application is to supply or has already supplied to a user. It also is a measure for the functional size of an application that must be maintained.

Project functional size

is the number of function points that is a measure for the amount of functionality of a new application or of changes to an existing application. Changes to an existing application pertain to adding, changing, and deleting functions. The project functional size is an essential parameter when determining the effort and schedule required for a project.

Determining the application functional size is elaborated on in subclause 3.5. Subclause 3.6 discusses the project functional size further.

2.3 Types of function point analyses

One of three types of function point analyses can be chosen, depending on the degree of detail of the specifications available. The following represent the different types of function point analyses. Notice that they are listed by degree of detail, number one having the least detail and number three the most:

1 Indicative function point analysis

2 High level function point analysis (previously known as *Estimated*)

3 Detailed function point analysis

These function point analyses are explained further in subclause 3.2.

2.4 Function point analyses during a project

Function point analyses can be carried out at different times during a project. They can therefore be related to the phases of a project (e.g. the planning phase, the execution phase, and the evaluation phase). As a result, the following breakdown of function point analyses arises: the *initial function point analysis*, the *interim function point analysis*, and the *final function point analysis*. These analyses are discussed further in subclause 3.4.

2.5 Scope of the analysis and boundary of the application to be analyzed

The scope of the analysis is the set of functional requirements/specifications to be included in the function point analysis. When the scope has been determined, the boundary can be defined, the conceptual interface between the application and its users and/or other applications.

As indicated earlier in subclause 2.1, the scope of the analysis and the boundary of an application to be counted plays an important role in FPA. Consequently, the boundaries of the application to be counted must first be determined in order to be able to perform a function point analysis.

The boundary is necessary in order to be able to determine:

— the application that certain data belongs to;

— which data crosses the boundary.

As mentioned in subclause 2.2, a distinction is made between a function point analysis for an application and a function point analysis for a project. Subclause 3.5.1 provides guidelines for determining the application functional size and subclause 3.6.1 gives guidelines for determining the project functional size.

2.6 Users

FPA acknowledges three types of users:

— The people and/or organizations that use or are going to use the application to be measured. This category includes, amongst others, the following: end-users, functional managers, and operators.

— The owner and/or employee(s) who determine(s) the requirements and wishes included in the specifications. These requirements and wishes are recorded on the basis of the demands of the end-user(s) for example, but also on the basis of requirements that a government or its legislation can impose on the application.

— Other applications that use the data or the functions of the application to be analyzed.

Because the function point analysis takes place from the perspective of the user(s), it is always necessary to have it done in cooperation with the user or, at the very least, to have the result of the analysis verified by the user. The user, after all, is the only one who can determine whether a certain function is being requested.

2.7 Functions and function types

The function point analysis measures the size of the functions of (a part of) an application. The analysis revolves around the *what* and not the *how* of the application to be analyzed. Only those components that the user requests, can recognize and considers significant are assessed. These components are called functions or base functional components. A function belongs to a function type.

FPA defines *function types* as follows:

The five types of components of which an application exists, as seen from the perspective of FPA. These components determine the amount of functionality an application provides to a user.

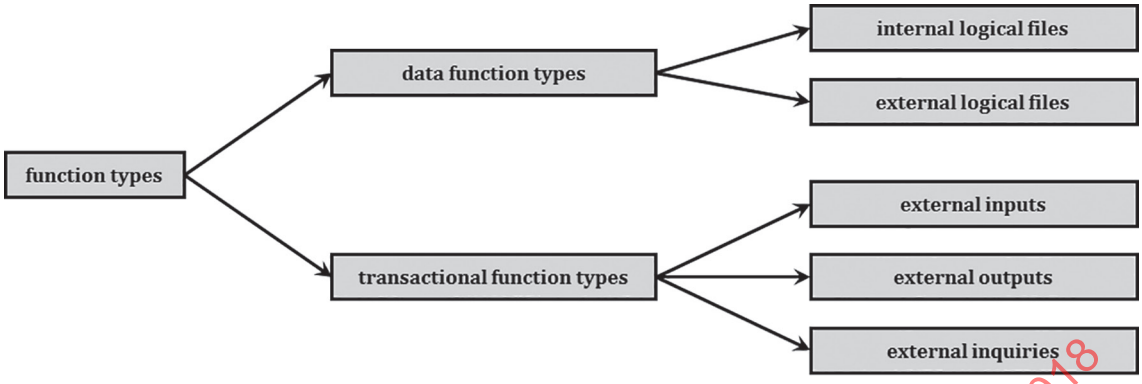


Figure 2 — Functions and function types

Function types are categorized into two main groups:

— Data function types

— Transactional function types

A *data function* is:

a logical group of data seen from the perspective of the user. FPA distinguishes between the following data function types:

— Internal logical files

— External logical files

A *transactional function* is:

an elementary process that meets the following criteria:

— the function has an autonomous meaning to the user and fully executes one complete processing of information, and

— after the function has been executed, the application is in a consistent state.

FPA distinguishes between the following transactional function types:

— External inputs

— External outputs

— External inquiries

Each function type is discussed in detail in clauses 5 through 9.

2.8 The complexity of a function

The *complexity of a function* is defined as follows:

The weight of a function on the basis of which a number of function points are allocated to the function.

The complexity of a function is determined by using the appropriate complexity matrix. A separate table has been defined for each function type. Complexity depends on the number of data element types and the number of referenced logical files connected to a given function. Three levels of complexity are distinguished:

Low: Few data element types and/or referenced logical files are involved with the function.

Average: The function is neither low nor high with regards to complexity.

High: Many data element types and/or referenced logical files are involved with the function.

The complexity tables that determine the levels of complexity are included in Annex A.

2.9 The valuing of functions

After the complexity of a function has been determined as described in clauses 5 through 9, the number of function points can be allocated to the function. This shall be done according to the rating in Table 1.

Table 1 — Function point table

Complexity	Function type				
	ILF	**ELF**	**EI**	**EO**	**EQ**
Low	**7**	**5**	3	4	3
Average	10	7	**4**	**5**	**4**
High	15	10	6	7	6

ILF = Internal logical file
ELF = External logical file
bold = value for high level FPA
EI = External input
EO = External output
EQ = External inquiry

High level specifications are enough to identify functions and their type when performing a *high level* function point analysis (see subclause 3.2.2), but it will be difficult to determine the complexity of these functions. In such a case, a data function is rated as *low*, while the rating *average* is used for a transactional function.

2.10 The functional size

The Number of function points: see Functional size functional size is the sum of the number of function points assigned to each of the functions (in the way described above) that lie within the scope of the object to be analyzed, that is the *application* or the *project*.

The functional size can also serve as a basis for preparing a project budget, by multiplying the number of function points with a productivity rate based on historical data (such as hours per function point). The preparation of a project budget is beyond the focus of this standard. More information on this subject can be found on the Nesma website nesma.org.

A Nesma FSM measurement result on the functional user requirements or specifications for a piece of software shall be labeled according to the following convention:

F(unction) P(oint) (ISO/IEC 24570:2018)

3 Guidelines to perform an FPA

This clause indicates how function point analysis shall be carried out. To this end, subclause 3.1 first presents a generally applicable step-by-step plan. Subclause 3.2 then indicates how to act when dealing with an indicative, high level, and detailed function point analysis. Subclause 3.3 goes into the role of the quality of specifications, while subclause 3.4 explains the use of FPA during a project. Subclauses 3.5 and 3.6 show how an application and a project functional size are determined in the event of development and in the event of enhancement, respectively. Subclause 3.7 introduces the definition of a functional change. Subclause 3.8 states what must be taken into consideration when dealing with the different ways of recording specifications. Subclause 3.9 concludes the clause with an illustration of how the different types of function point analyses can be applied during the life cycle of an application. For illustration purposes, this subclause will assume a generic application life cycle as phasing method.

3.1 Step-by-step plan to perform an FPA

Below follows a step-by-step plan to perform a function point analysis

Step 1: Collect the available documentation. The documentation that should be present for an indicative function point analysis, a high level function point analysis, and a detailed function point analysis is described in subclauses 3.2.1, 3.2.2 and 3.2.3 respectively.

Step 2: Determine the users of the application (see subclause 2.6).

Step 3: Establish whether an application function point analysis or a project function point analysis must be carried out. If an application function point analysis must be performed, follow the instructions stated in subclause 3.5. If a project function point analysis must be performed, follow the instructions in subclause 3.6.

Step 4: Determine from which other application(s) the application to be analyzed receives and/or uses data.

Step 5: Identify the functions and determine their type and complexity according to the guidelines described in clauses 5 through 9. When doing so, adhere to the sequence in which the clauses appear. Assign the number of function points using the function point table illustrated in subclause 2.9. This will result in the functional size.

Register the structure of the analysis and the number of function points. Particularly record any preconditions that have been used and assumptions that have been made.

Step 6: Together with the user(s), verify the result in relation to those aspects where specific interpretation of the available *specifications* was needed. If necessary, make any corrections as a result of that verification.

Step 7: Verify the result with an FPA expert in relation to those aspects where specific interpretation of the *counting guidelines* was needed. This may or may not be necessary. Make any corrections that are required as a result of that verification.

3.2 Types of function point analyses and their accuracy

Depending on the degree of detail of the specifications available, one of three types of function point analyses can be chosen: *an indicative, a high level, or a detailed function point analysis*. Each type of function point analysis mentioned in this subclause can be used both for the determination of the project size as for the determination of the application size. The minimum specifications required are different for each of the three types of function point analysis. In the subclauses below, the specifications required to perform each of the three types of analyses are stated. Each subclause, finally, will indicate when a particular type of analysis can be executed in the life cycle of an application.

3.2.1 Indicative function point analysis

Definition

An indicative function point analysis indicates the size of an application or a project based on either a conceptual data model or a normalized data model. Caution is advised when using this indicative functional size, as deviations of up to 50% are possible.

When a conceptual model, or a model with a comparable level of detail, is available, the following formula indicates the functional size:

> The number of internal logical files in the conceptual data model * 35
>
> +
>
> the number of external logical files in the conceptual data model * 15

Notice that the entities that are an FPA table (see subclause 4.20) *and* that are maintained by the application to be analyzed are counted together as one internal logical file. The same applies to the entities that are an FPA table type but which are maintained by another application, they are counted together as one external logical file. So, for all FPA tables together, at most two logical files are counted.

The factor 35 assumes that for each internal logical file three external inputs (add, change, delete), two external outputs, one external inquiry and some generic functionality will be present. Herewith a low complexity of the internal logical file is supposed (7 fp), and an average complexity of the transactions (3x4 + 2x5 + 4 = 26 fp) and 2 fp for the generic functionality.

The factor 15 is based on a low complexity of the external logical file (5 fp), an external output (5 fp) and an external inquiry (4 fp) and some generic functionality (1 fp) for each external logical file.

When a data model in third normal form, or a model with a comparable level of detail, is available, the following formula indicates the functional size:

> The number of entity types maintained in the normalized data model * 25
>
> +
>
> the number of entity types referenced in the normalized data model * 10

Here, too, the entity types that are from the FPA table type (see subclause 4.20) *and* that are maintained by the application to be counted, are counted together as one entity type. The same applies to the entity types that are from the FPA table type but which are maintained by another application. For FPA tables, therefore, at most two entity types are counted.

Applicability

An indicative function point analysis can be carried out when a data model is available, from which data functions can be derived (see subclause 3.7). A data model can be presented in many ways, including Bachmann diagram, description in textual form, Entity Relationship Diagram (ERD) or UML class model.

This data model can be at a high level (e.g. at the end of the requirements phase in a waterfall approach or completion of the product backlog in an agile approach), or at a detailed level, later on in the application life cycle. High level here means that not all details are known, so the logical files are known, with the applications that maintain them, but without complete lists of attributes and without full details on their relations and validations.

Specifications required

The above shows that the following should be at your disposal for an indicative function point analysis:

— a conceptual or normalized data model of the application to be counted;

— an indication as to how the logical files distinguished are maintained, either by the application to be counted or by another application.

3.2.2 High level function point analysis

In previous releases of this guideline, this type of analysis was named as *estimated.* Since this term could be associated with less accuracy, this term has been changed to high level, indicating that it can be used when not all details of the specification are known.

Definition

A high level function point analysis determines the number of functions for each function type (transactional function types and data function types) and uses a standard value for complexity: *Average* for transactional function types and *Low* for data function types.

These complexity values overrule all other complexity value rules for detailed function point analyses.

Applicability

A high level function point analysis can be carried out when a data model (see subclause 3.2.1) is available and an insight in the transactional functions (see subclause 2.7) that use it.

In general, in a waterfall approach this information becomes available at the end of the analysis phase and in an agile approach it is available when user stories can be related to transactional functions and data functions can be identified. This is when a sprint backlog (or sometimes, the product backlog) is ready for use.

Specifications required

The following specifications must be available for a high level analysis:

— a model that shows the logical files involved and their relations;

— an indication as to how the logical files distinguished are maintained: by the application to be counted or by another application;

— a model that shows the application functions with their incoming and outgoing information flows;

— the information flows between the functions of the application to be counted and its environment.

3.2.3 Detailed function point analysis

Definition

The detailed function point analysis is the most accurate analysis in which all the specifications needed for FPA are known in detail. This means that transactional functions have been specified up to the level of referenced logical files and data element types, and that logical files have been specified up to the level of record types and data element types. As a result, the complexity for each function identified can be established.

Applicability

A detailed function point analysis can be carried out when a detailed data model (see subclause 4.2.1) is available together with a detailed insight in the transactional functions that use it. All details on how the logical files within the data model are used by the transactional functions and all further functional details about them need to be known.

In general, in a waterfall approach this information becomes available during or at the end of functional design. In an agile approach this would be within a sprint, when corresponding user stories reach the definition of done.

Specifications required

The following specifications must be available for a detailed analysis:

— a model with all logical files and their relations (e.g. a Bachman diagram or an Entity Relationship Diagram);

— the record types and the data element types of the logical files;

— an indication as to how the logical files distinguished are maintained: by the application to be counted or by another application;

— a model that shows the application functions, their incoming and outgoing information flows, the logical files involved with the functions, and the supporting functions (help functions and so on);

— a detailed specification of the incoming and outgoing information flows of the application up to the level of data element types.

3.3 Role of the quality of the specifications

Regardless of the type of function point analysis (as described in subclause 3.2), sound functional specifications of the application are needed in order to perform a function point analysis. Depending on the method used, the form of the specifications produced can differ[2]. The function point analysis, however, should continuously render the same result for the same application.

The reliability of a function point analysis depends directly on the quality of the specifications provided. Good-quality specifications will ensure that little effort is needed to translate specifications into function points and will result in a reliable count. Flawed or incomplete specifications can make it impossible to perform an FPA, or the result of the analysis may differ considerably from person to person as a result of various interpretations of the specifications.

3.4 FPA during a project

FPA plays a role during the entire course of a system development project. For instance, during the planning phase of a project, an *initial function point analysis* is carried out. As soon as change requests pertaining to specifications already recorded have been submitted during the project, *interim function point analyses* are carried out. When a project has been fully completed, the *final function point analysis* determines how big the supplied product is and what the functional size of the project has been. It does not matter which phases of an application's life cycle are being carried out in the project. Given a system life cycle as a phasing method, for example, the project can pertain to analysis, functional design, and/or construction. Regardless of the project's content, there will always be an initial and a final function point analysis, and sometimes one or more interim function point analyses.

An *initial function point analysis* is:

an analysis at the beginning of a project for developing an application or for enhancing an application (adding, changing, and deleting functionality) in which the functional size of an application (see subclause 3.5) and a project (see subclause 3.6) is recorded.

An initial function point analysis should be performed at the beginning of a project. Depending on the specifications available, this analysis is either an indicative, high level, or detailed function point analysis. The objective of the initial function point analysis is to draw up a budget for the project in terms of effort and schedule.

An *interim function point analysis* is:

an analysis during a new development project or an enhancement project in which the size of an interim enhancement (adding, changing, or deleting functionality) is determined. This means that the

2) Nesma has published a guide "FPA applied to UML / Use Case documentation" that gives guidance how the rules in this guideline can be applied from this type of documentation.

effect that a change will have on both an application functional size (see subclause 3.5) and a project functional size (see subclause 3.6) is recorded.

An interim function point analysis should be done when functional specifications are changed. Depending on the specifications available, this analysis is either an indicative, high level, or detailed function point analysis. The objective of the interim function point analysis is to determine the effect of change requests on the price and delivery date agreed upon with the customer.

A *final function point analysis* is:

an analysis at the end of a new development project or an enhancement project (adding, changing, or deleting functionality) in which the final functional size of an application (see subclause 3.5) or the project (see subclause 3.6) are recorded.

A final function point analysis takes place at the end of a project. The project can pertain to the development of an application or to the realization of enhancements during an application's operation and maintenance phase. One objective of the final function point analysis is to determine the functional size of the *application* in function points. Another objective is to determine the size of the *project* so that productivity can be determined and, for example, a final monetary settlement for the project can take place, when a fixed price per function point has been agreed upon.

3.5 Determining the functional size of an application

As stated in subclause 2.2, FPA can be used to determine the application functional size or the project functional size. This subclause explains the use of FPA when determining the functional size of an *application*.

The purpose of performing analyses on applications is to determine the amount of functionality that is to be furnished to a user or that has already been furnished to him. This means that the functional specifications of an application must serve as the starting point and *not* the physical components such as programs or physical files.

Determining the functional size of an application can be carried out differently when developing an application than when enhancing an application. Still, in both cases, the application boundary must first be determined. This and the analyzing method employed for both development and enhancement are covered in greater depth in the subclauses below.

3.5.1 Determining the application boundary

An *application boundary* is:

the boundary between the (developing or developed) application and its environment (users and/or other applications).

An *application* in this International Standard refers to:

an automated information system. This is an application that collects, saves, processes, and presents data by means of a computer.

The following guidelines can help in determining the application boundary:

— The application demarcated by the application boundary should make up an independent whole that can function separately from other applications to a large degree.

— Establish whoever the owner or main user is. If there are several owners or key users, it often means that there are several applications.

— Look at the application through the eyes of the user, so only use the part of the application the user can actually observe. Use the specifications that describe or define the outside of the application, seen from the user perspective. This is called the application context, and it can be represented in a context diagram, among other ways. Determine what is located inside and outside the application.

Only those things that the user requests and that are relevant to him are significant to the function point analysis.

— Think of an application as a group of programs maintained as a whole. The application boundary encloses this group of programs. All functions within this boundary are identified.

3.5.2 Functional size of new applications

This pertains to the functional size of applications in the process of being built or that have already been built at the request of a user or user organization, and that provide a solution to the needs or wishes of the user or user organization.

Determining the functional size during the development of an application occurs as indicated in subclause 3.1. If the application in a development project is realized in a single project, determining the functional size of an *application* does not occur differently than when determining the functional size of a *project* (see subclause 3.6.2). Notice, however, that the size of any conversion software shall not be counted when the functional size of an application is determined.

If the application is being realized in the form of a number of sub-projects carried out in parallel, then the total functionality furnished by all the sub-projects will have to be examined in order to determine the functional size of an application. When examining these sub-projects, make sure that functionality appearing in more than one sub-project (such as a logical file) is not counted twice.

In the event of enhancement to an existing application (the adding, changing, or deleting of functionality), the functional size of the (changed) application should be determined as indicated in subclause 3.5.3.

3.5.3 Functional size of enhanced applications

This pertains to the functional size of an application after enhancement. In principle, enhancement can take place during each phase of an application's life cycle, but usually occurs during construction or during operation and maintenance. In the event of major enhancements, a separate project can be defined in which the functional size of the project is determined, in addition to the functional size of the application. In all cases, the functional size of the application is determined in the same way after the enhancement.

Below are the steps to be taken in order to determine the application functional size after enhancement:

Step 1: Determine the number of function points of the application before the change (*AFPB*).

Step 2: Identify which transactions and/or logical files are added to the application and establish how many function points they represent (*ADD*).

Step 3: Determine which transactions and/or logical files are deleted from the existing application and count how many function points they represent (*DEL*).

Step 4: Establish which transactions and/or logical files change. Then determine the number of function points that they represent before the change (*CHGB*) and after the change (*CHGA*).

Step 5: Determine the functional size of the application after the enhancement (*AFPA*) as follows:

AFPA = [AFPB + ADD DEL + (CHGA CHGB)]

3.5.4 Functional size of re-built applications

If one application is replaced by another with the same functionality, then the functional size of the new application is equal to the old application it is replacing.

If enhancements are the result of such a replacement, the functional size of the application can be determined in two ways:

— The replacement can be considered a new application. If this option is chosen, counting is done as described in subclause 3.5.2.

— The replacement can be considered an enhancement to the application being replaced. In this case, counting is carried out as indicated in subclause 3.5.3.

The result (the application functional size) of both methods is the same.

3.6 Determining the functional size of a project

As indicated in subclause 2.2, FPA can be used to determine an application functional size or a project functional size. This subclause explains the use of FPA to determine the functional size of a *project*.

Determining the functional size of a project (i.e. counting the number of function points of a project) differs in a number of ways from purely and simply calculating the functional size of an application, that is determining the amount of functionality provided or to be provided. This is because the effort required does not always result in an increase of functionality. Consider, for example, an effort to remove functionality, or to create a one-time functionality in order to convert data.

Using a number of project situations, the subclauses below spell out how the functional size of a project can be determined. Table 2 illustrates an example of how the functional size of both a project and an application are determined, as well as the differences between the two.

Table 2 — Functional size of a project versus an application

Size type	Lifecycle stage					
	Initial release	**Release 1**		**Release 2**		
ADD	1000	200		500		
DELETE		40		100		
CHANGE Before		80		220		
CHANGE After		100		200		
Project size	1000	ADD CHANGE After DELETE **TOTAL**	200 100 40 **340**	ADD CHANGE After DELETE **TOTAL**	500 200 100 **800**
Application size	1000	Initial release ADD CHANGE After CHANGE Before DELETE **TOTAL**	1000 +200 +100 -80 -40 **1180**	Release 1 ADD CHANGE After CHANGE Before DELETE **TOTAL**	1180 +500 +200 -220 -100 **1560**

Determining the functional size of a project when an application is developed can be carried out differently than when an application is enhanced. Still, in both cases, the application boundary must first be fixed. This and the counting method employed for both development and enhancement are covered in greater depth in the subclauses below.

3.6.1 Determining the scope of a project function point analysis

The *scope* of a project function point analysis is:

the set of functional requirements/specifications of a development project or an enhancement project to be included in a function point analysis. This may include one or more applications and therefore more than one application boundary may have to be determined as described in subclause 3.5.1.

This definition shows that two types of projects are distinguished:

— Development projects

— Enhancement projects

A *development project* is:

a project in which a completely new application is realized. In its execution, a development project can be split up into a number of sub-projects, each of which is responsible for a certain sub-system of the entire application. Each sub-project should then be considered an individual development project, if the sub-system itself is an application.

Re-building an existing application (re-engineering) is considered as development (see subclause 3.5.4).

FPA defines an *enhancement project* as:

a project in which enhancements are carried out on an existing application. This means that functionality can be added to, changed in, or deleted from an existing application (see subclause 3.7).

Guidelines to determine the scope of the project

The following guidelines can help in determining the scope of the project:

— Look at the application to be realized through the eyes of the user. The logical structures should be taken into consideration, not the physical structures.

— An application can be developed as a number of sub-projects executed more or less in parallel, each of which realizes a sub-system. The scope of such a sub-project therefore includes a sub-system. If the sub-systems must be able to exist independently (e.g. because of a phased implementation of the application or for functional reasons), then the exchange of data between the sub-systems is included in the functional size of each sub-project. The scope of the application contains all the sub-projects. This means that the interfaces between the sub-systems lie within the entire application boundary. The project functional size is the sum of the number of function points of the sub-projects and can be higher than the number of function points of the entire application (application functional size), because in this situation, an internal logical file of a particular sub-project (for example) is also counted as an internal logical file or as an external logical file for another sub-project that makes use of the same internal logical file.

— A practical way of figuring out whether a certain function lies within a boundary of an application requested by a user is to ask whether the user really wants to pay for this function.

— If in doubt, consult with the user if possible.

3.6.2 Functional size of development projects

A development project realizes an entirely new application. When a development project is split up into a number of sub-projects, then each sub-project must be treated as an independent development project when determining the functional size of a *project*.

The steps to be taken here are as follows:

Step 1: Determine the number of function points of each (sub)system to be realized.

Step 2: Count the number of function points of the conversion.

These function points contribute to the functional size of the project only. The conversion software required does not yield any additional functionality, but is only a one-time tool and therefore not a part of the application to be implemented.

Step 3: Determine the number of function points of the changes in other applications that are being realized in the project (see steps 1 through 5 in subclause 3.1).

These function points contribute to the functional size of the *project*. (The new functional size of the *applications* affected by the project must also be determined per application.)

Step 4: The functional size of the project is the sum of the number of function points recorded as a result of steps 1, 2, and 3.

Note: *A project functional size is used often to budget. When different applications are being modified/enhanced, make certain that budgeting is done per application because, for example, there may be different development environments and, therefore, different productivity rates.*

3.6.3 Functional size of enhancement projects

An enhancement project considers enhancements to one or more existing applications. This means that functionality can be added to, changed in, and deleted from these applications.

The steps required to determine the functional size of the project in function points are as follows[3]:

Step 1: Identify which transactions and/or logical files are going to be added to the application(s) and establish how many function points they represent (*ADD*).

Step 2: Determine which transactions and/or logical files are going to be deleted from the existing application(s) and determine how many function points they represent (*DEL*).

Step 3: Establish which transactions and/or logical files change. Then determine the number of function points they represent after the change (*CHGA*).

Step 4: Calculate the functional size for the enhancement project as follows (*EFP*):

$$EFP = ADD + DEL + CHGA$$

Step 5: If conversion software has to be made as a result of changes, determine the number of function points it entails and add it to the functional size of the project determined in step 4.

Note: *Existing internal logical files and external logical files that are used by the functions to be added, changed, or deleted, but are themselves not changed during an enhancement project, are not counted as internal logical files or as external logical files when the functional size of the project is being determined.*

3) Nesma has published a guide "FPA for software enhancement" that deals with the use of functional sizing during enhancement. The method for sizing enhancement projects, described in that guide, is not a part of this International Standard and it results in a different unit of measurement.

3.6.4 The project function point analysis during the replacement of an application

It is often necessary to replace applications that have been operational for a longer period of time with applications that are more efficient and that meet the requirements of current-day information technology. This is done with re-engineering projects.

Because an entire application has to be built in such a case, determining the functional size of the project occurs in the same fashion as when dealing with development projects (see subclause 3.6.2).

3.7 Definition of functional change

3.7.1 General

Functional specifications of one or more functions may be adjusted on the basis of a change request. If these adjustments lead to activities (adjustment of the design, software code, testing, etc.) intended to bring the application in accordance with the changed specifications, then these functions should be considered as functions within the scope of the enhancement release.

To determine the size of a function after the change, the same counting guidelines are used as for development.

All functions mentioned in the change request, and all functions that should be changed as a result of the proposed changes, have to be analyzed in the function point analysis of the enhancement release.

3.7.2 Modification of a transactional function

After the enhancement, the transactional function should be functionally changed from the user's point of view. The user's primary intent of this transactional function has not changed.

A transactional function referred to in the change request, is to be considered as functionally changed if it meets at least one of the following criteria:

— one or more DETs are added to the transactional function, and/or one or more DETs of the transactional function are changed (see subclause 4.7.4), and/or one or more DETs are removed from the transactional function;

— one or more logical files are added to the transactional function and/or one or more logical files are removed from the transactional function;

— in the enhancement release a data function is changed and at least one of the modified DETs of this data function is part of the transactional function;

— the logical way of processing of the transactional function is changed in the enhancement release (for example as a result of added, modified and/or deleted validations or calculations).

3.7.3 Modification of a data function

A data function referred to in the change request, is to be considered as functionally modified if it meets at least one of the following criteria:

— the structure of the data function has changed because in the enhancement project one or more DETs are added to the data function, and/or one or more DETs of the data function are changed (see subclause 3.7.4) and/or one or more DETs are removed from the data function;

— the nature of the data function has changed in the enhancement project. This is the case where as a result of functional changes to the transactional function (see subclause 3.7.2) the data function changes from ELF to ILF or vice versa.

3.7.4 Modification of a DET

A DET changes if it meets at least one of the following criteria:

— the length (number of positions) changes;

— the data type changes (for example from alphanumeric to numeric);

— the number of decimal places changes.

3.8 FPA in specific situations

FPA can be applied in different situations, but must be dealt with in a specific way for each particular situation. This subclause explains the use of FPA in the following situations:

— Analyzing on the basis of traditional design

— Analyzing packaged software

— Analyzing screens or windows

— Analyzing during prototyping

3.8.1 Analyzing on the basis of traditional design

Via models, a traditional design describes the functionality a user desires. Generally speaking, a data model and a process model are encountered here. A data model can take the shape of an Entity Relationship Diagram to which a description of entity types, attribute types, and relationships has been linked. A process model on the other hand can take the shape of a Data Flow Diagram to which a description of the functions and the data flows is linked. When an analysis is done from a traditional design, the models cited serve as the basis for the FPA, as well as the window and screen layouts and list layouts often encountered. The screen, window and list layouts are not necessary, but are very useful. Data functions are derived from the data model, and the transactional functions that must be identified are determined from the process model.

The major advantages of performing a function point analysis from a traditional design are that:

— analysis is done from the perspective of the functionality requested and defined, in which technological considerations hardly play a role;

— the design is a complete illustration of the functionality requested.

3.8.2 Analyzing packaged software

Packaged software implemented in an organization represents a certain quantity of functionality.

The function point analysis carried out during the phase in which the specifications are recorded is a reflection of the amount of functionality a user desires. This analysis is independent of the solution to be chosen and is separate from the implementation or non-implementation of packaged software.

At the close of the specification phase, a decision is made whether to build an application or to buy packaged software that satisfies the functional requirements.

The way in which the analysis should be carried out depends a lot on the available documentation of the packaged software. Other than that, the approach here is no different than the one given in subclause 3.1. Next we will discuss the process in which the data functions (logical files) and transactional functions (transactions) of packaged software can be identified.

General

When packaged software is considered a possible solution, the first step to be taken is to determine whether suitable packaged software is available that can run on the technical infrastructure desired. If so, the following about the chosen packaged software must be determined:

1. what functionality desired by the user can the packaged software provide, and what is the number of the function points of this functionality;
2. what functionality desired by the user can the packaged software *not* provide, and what is the number of function points of this functionality;
3. what functionality not desired by the user does the packaged software provide, and what is the number of function points of this functionality.

Group 1 represents the function points counted during the specification phase that packaged software can supply.

Group 2 makes up the number of desired function points either not provided or that must be modified. These function points should not be included in the useful application function point analysis of the packaged software, because the application function point analysis should reflect the *desired* functionality that is provided. If a decision is subsequently made to upgrade the packaged software so that it complies with all of the user's original requirements, the upgrade should be treated as an enhancement (see subclauses 3.5.3 and 3.6.3).

Group 3, the surplus functionality that packaged software provides, can of course be a reason to choose certain packaged software. The additional functionality, however, falls outside the scope of the original project while it must still be paid for nonetheless. It is a part of the total functional size of the packaged software, but not a part of the functional size of the application that is effective and useful to the user. Depending on the purpose of the function point analysis ("measure the functional size of the packaged software" versus "measure the *effective and useful* functional size of the software") one shall or shall not take this surplus functionality into consideration.

The big advantage of using FPA when acquiring packaged software is that attention is focused on the relationship between the functionality provided and the cost factors that play a role in the decision either to produce a bespoke application or to buy packaged software. With the help of FPA, a decision between "make or buy" is made possible on the basis of functionality, costs, and the time frame within which functionality becomes available.

Analyzing the size of packaged software therefore has three possible objectives:

— To establish the price-performance ratio of packaged software. In other words, what will the packaged software cost per function point? When this is assessed, only the function points of the functionality that the user or customer requires and that the packaged software is going to provide are relevant.

— To establish the ratio between the functionality provided by packaged software that a user or customer requires and the total functionality the user or customer requires. This would include both additions and changes.

— To establish the ratio between the functionality provided by packaged software that a user or customer requires and the total functionality the packaged software provides.

Determine the data functions (logical files) of packaged software

When packaged software is acquired, a conceptual data model usually is not a part of the documentation supplied. In some cases, a data model of the packaged software will be available. In that case, the data functions can be derived from this data model. In such a case, use the guidelines found in subclauses 4.20 and 4.21 and in clauses 5 and 6.

If a data model is not present, establish which logical files can probably be identified, using the functionality provided. This can also be derived from the physical database structure, when available.

Determine the transactional functions (transactions) of packaged software

The transactional functions provided must be determined as well as possible from the functional specifications or from a user's manual if they are available.

If the documentation issued is inadequate and a test or demonstration implementation of the packaged software is available, the transactional functions can be determined from the screens or windows (see subclause 3.8.3 for more about this).

If only the menu structure is available, try to derive the functions from it. Choose three external inputs (add, change, and delete) when confronted with a menu option such as *maintain*. For a menu option such as *display*, choose one external inquiry and one external output.

Determine the functionality required

When determining the functionality required, assume the functional requirements the user or customer has imposed on packaged software. In other words, only those parts of the packaged software are valued that the user has specified beforehand.

In short, use the functional requirements of the user or the customer in order to analyze.

If the functional requirements have not been defined, they must still be established in cooperation with the user by figuring out which logical files and which functions of the packaged software are relevant. (Consult the above in this subclause for how data functions and transactional functions can be determined.)

Functionality provided by the packaged software but not specifically requested by the user *can* be included when determining the total application functional size of the packaged software; however, it must *not* be included when determining the effective application functional size (the application functional size useful to the user).

3.8.3 Analyzing screens or windows

Packaged software and existing applications often lack suitable documentation needed to perform a function point analysis. If functional specifications are missing or are insufficient, it may be necessary to derive (a part of) the functions from the physical components of the application. One option is to start up the application concerned and analyze from the screens or windows. This means that functions have to be tracked down by going through each branch of the menu tree up to and including the input and output screens or windows. These input and output screens or windows reflect the functionality that an application provides. A user's manual can also be a useful means to establish the functionality supplied.

A Graphical User Interface (GUI) can present the application to a user in a variety of ways. Generally speaking, look beyond the packaging and deduce the actual functionality provided by the application in terms of internal logical files, external logical files, external inputs, external outputs, and external inquiries. The sequence and shape in which GUI elements are used in a transaction do not play a role, but must be considered as a whole. The individual components or formats of GUI elements do not lead to additional FPA functions, no matter how user friendly they may appear in design. Many of these components correspond to a data element type and are counted accordingly.

Functionality that the GUI environment provides, or functionality expected as a standard in the GUI environment and obtained (almost) automatically by tools in the GUI environment, like responsiveness of the GUI to different devices, shall be treated similarly as an extension of the operating system and do not lead to counting of additional functions or data element types.

Pay attention to the following when functions are established in this fashion:

— Prevent double counting: The same transactional function can appear at several places in a menu structure.

— Pay attention to continuation screens. Screens or windows inextricably bound together usually define one transactional function only.

— If a screen or window has continuation screens or windows that are not bound to it inextricably, the continuation screens usually result in new transactional functions, unless the new transactional functions found in this fashion have been counted somewhere else already.

— Derive from the screens or windows and the menu structure which reports are created by the application. Count each individual report as one external output, but be aware of double counting (see subclause 4.3).

— Sometimes a report with the same layout can be shown via different media (e.g. display screen, pop-up window or printer). When the logical processing is the same, only one external output should be counted (see subclause 8.1).

— An external inquiry shall be counted only if it is cited explicitly as such in the menu structure. Displaying data occurs rather often in practice as part of an external input, namely when data is changed and/or deleted. Given such a case, the displaying of data is not counted as an external inquiry.

— Count list functions as indicated in subclause 4.16.

— Using the documentation or menu options, determine which transaction files exist. Count these files accordingly as either external inputs or as external outputs. Also determine how many external inputs and external outputs should be identified per transaction file.

— Try to derive from the maintenance functions which logical files the user can maintain; in other words, which internal logical files are present.

— Often, it will not be possible to substantiate the complexity of transactional functions and data functions when analyzing from screens or windows. In that case, value each transactional function as average and each data function as low.

— Count the menu structures as indicated in subclause 4.15.

— Try to gain insight into (batch) functions executed periodically. Sometimes it is possible to see from remarks on screens or windows, in documentation, or in menus whether a transaction run will operate or should have operated (at night, for example).

3.8.4 Analyzing when prototyping

Prototyping is used:

— as a strategy in order to determine functional specifications;

— as an aid to develop the organization of screens, windows and dialogs for known functional specifications.

Prototyping to determine specifications

When prototyping is used as a design strategy to determine the functional specifications of an application, FPA should be applied carefully. Prototyping is usually applied when there is uncertainty about the information problem and about the requirements that users expect of a solution for that problem. The causes of this uncertainty are due to a communication gap between the developer and user, and to a shift or change in the information need when the user has gained experience with the application. As long as uncertainty exists about the final functionality of an application to be developed, a function point analysis will not provide reliable insight into the ultimate size of this application. If necessary, an indicative function point analysis can be carried out if a (rudimentary) data model is available. It is important, however, to document specifications when prototyping, too, so a function point analysis can be done at the end of the prototyping cycle.

Prototyping to design the user interface

If prototyping is used to design the user interface and the functional specifications are already recorded from the start, then normal FPA guidelines apply and can be used without risk.

3.9 Illustration: FPA and the application life cycle

A function point analysis can be carried out only when a certain minimum of specifications is present. Which specifications are available and the way in which they are supplied depend on the phase within the application life cycle, on the nature of the project, and on the methods that are used.

Subclauses 3.9.1 through 3.9.6 will explain this further and, in doing so, will illustrate the phasing according to a general system life cycle. Comparable remarks apply to every other arbitrary phasing method. We leave it to the reader to compare this general application life cycle with the methods his organization uses and to translate them.

3.9.1 FPA during the requirements phase

This phase assesses whether the development of a new or improved application is technologically, economically, socially, and organizationally feasible and worthwhile. It is the first step in the development of a particular application[4)].

Early in the application lifecycle the problem area is analyzed, and all application requirements are specified:

— what does the existing application look like;

— what are the boundaries of the application;

— what functionality should the application provide;

— how can users work with the application;

— what requirements have been imposed on the quality of the application;

— what parts of existing, planned, or standard hardware and software can be used;

— how should the transfer from the existing situation to the desired situation take place.

This will result in:

— a model of the business activities;

— the basic requirements for the application to be realized;

— a specification of the environment requirements of the application to be implemented;

— requirements pertaining to the technological characteristics;

— a high level data model.

Moment and type of analysis

The specifications that must be furnished at the end of the requirements phase in the system life cycle are not always adequate for performing a high level function point analysis (see subclause 3.2.2), but usually are sufficient for an *indicative function point analysis* (see subclause 3.2.1).

Be aware that the size of an application estimated in the requirements phase is usually too low. This is a consequence of the high abstraction level of these specifications that can keep relevant details hidden.

4) Nesma has published a guide "The application of Function Point Analysis in the early phases of the application life cycle" that gives guidelines for functional sizing when only limited information is available and details to perform a high level analysis are missing.

Experiences of FPA users have shown that the degree to which a functional size is underestimated in this phase is constant in an organization. Each organization should decide on a standard for itself in order to compensate for this. This compensation can be referred to as autonomous growth (see subclause 3.9.2 and Annex C).

Objective of the analysis

The objective of this analysis is to obtain an initial indication of the size of the application to be developed. This indication can be used to:

— Determine the resources necessary for the application to be developed

— Budget the project for the design of the application

— Fix a budget for the development department

— Evaluate quotations when subcontracting later phases of system development

Required documentation

In all cases, the documentation must contain the specifications as denoted in subclause 3.2.1.

3.9.2 FPA during the analysis phase

During the analysis phase, attention shifts from analyzing business activities to specifying the application that is to support these activities. Specifying the application requires an understanding of all aspects of the information that must be stored in the logical file(s), and of the way in which the user is going to communicate with the application. In fact, a number of models may be created that can be seen as a blueprint of the application to be developed.

This will result in:

— a model that indicates when and why information is required for which management and for which control decisions;

— a model that defines which data is required for which output and what the significance is of that data;

— a model in which the structure, form, and cohesion of the data of the application to be developed are established;

— a model that records the technical requirements for the application to be realized.

The models used here and the documentation produced as a result will vary for each system development method.

Moment and type of analysis

Sufficient specifications emerge during the analysis phase that will enable a *high level function point analysis*.

Analysis can be done as soon as the specifications required for a high level analysis are available. Analysis can occur at different moments during analysis, depending on the methods used. However, it usually takes place at the end of the analysis phase.

Be aware that the functional size of an application in the analysis phase is also usually estimated too low. This is a consequence of the still relatively high abstraction level of these specifications that can keep relevant details hidden. Experiences of FPA users have shown that the degree to which a count is underestimated is constant in an organization. Each organization should decide on a standard for itself in order to compensate for this. This compensation can be referred to as autonomous growth. The autonomous growth in the analysis phase is lower than the autonomous growth in the requirements phase.

Be aware that autonomous growth is different from scope creep. Autonomous growth occurs by revealing functionality while detailing and having a closer look at the functional user requirements; it covers functionality that was already implied by the requirements, but was not originally recognized.

Scope creep occurs by the addition of new functionality by the user. It generates functional size that would not have been found even after detailing the requirements.

For a detailed discussion on the increase of functional size, see Annex C.

Objective of the analysis

The objective of the function point analysis at the end of the analysis phase is to obtain a better indication of the size of the application to be developed. This estimate can be used to:

— Determine the resources necessary for the application to be developed

— Budget the project for the application to be developed

— Fix a budget for the development department

— Evaluate quotations when subcontracting later phases of application development

— Settle financial matters pertaining to the analysis phase via costing when the phase has been subcontracted by means of a contract. (A fixed price per specified function point is used in such a case.)

— Record that more or less work has to be done than an earlier function point analysis indicated

Required documentation

In all cases, the documentation must contain the specifications denoted in subclause 3.2.2.

3.9.3 FPA during the functional design phase

During this phase, design specifications are documented to such a degree that they can be used as the basis to realize manual procedures or computer programs. Additionally, the logical data structure is determined so that it can be used as the basis to establish a technical data structure or database design.

Moment and type of analysis

During the functional design phase, all the specifications required for performing a *detailed function point analysis* become available. This functional size will be equal to the final functional size if no interim changes to the functional specifications appear during the next phases.

Just as in the analysis phase, counting function points can be done during the phase or at the end of the phase.

Objective of the analysis

The following can be accomplished on the basis of the detailed function point analysis at the end of the functional design phase:

— Budgeting the continuation of the project for realizing the application

— Estimating the effort and financial means required

— Evaluating a quotation for subcontracting the construction phase

— Settling financial matters pertaining to the functional design phase via costing when the phase has been subcontracted by means of a contract. (A fixed price per specified function point is used in such a case.)

— Recording that more or less work has to be done than an earlier analysis indicated

Required documentation

In all cases, the documentation must contain the specifications noted in subclause 3.2.3.

3.9.4 FPA during the construction phase

The construction phase encompasses the actual building and testing of the application. When the application is not actually built, but acquired as standard packaged software instead, this phase consists of evaluating and choosing one of the various standard packaged software options.

Moment and type of analysis

Analysis takes place during construction when changes are made to functional specifications. In essence, then, a step backwards is taken to the analysis phase or to the functional design phase where such specifications were drawn up. In the event of a change to the functional specifications at this stage, a so-called *interim function point analysis* is performed. An interim function point analysis reflects the effect that a change has on the project functional size and on the application functional size.

Changes here involve small enhancements. If major changes must be implemented, a new project is usually defined.

The application ultimately developed must be counted once again at the end of the construction phase in order to determine the final amount of functionality that has been realized (the size of the application). This is a *final function point analysis*.

The function point analysis at the end of the construction phase can also be derived from the function point analysis after the functional design and all the interim function point analyses during the application's construction phase.

Objective of the analysis

The first objective of an interim function point analysis is to record the costs that must be added to or deducted from the price agreed upon earlier, in the event of more or less work. The second objective of an interim function point analysis is to determine the new size of the application so that an indication can be obtained of the effect of the changes on the operation and maintenance phase of the application.

The objective of the detailed function point analysis at the end of the construction phase is to record the number of function points that must be maintained. It also allows the measurement of the stability of the design on the basis of any increase in function points. Using the project documentation, the final function point analysis, and the number of hours spent on the project, productivity can be determined.

Required documentation

The document to be provided at the end of this phase is not of importance to FPA. However, the documents from the analysis and functional design phases, in which functional changes have been made, *are* important. Everything stated in subclauses 3.2.2 and 3.2.3 for the high level and detailed function point analyses should still be specified as it pertains to such changes.

3.9.5 FPA during the implementation phase

During the implementation phase any organizational changes that need to be implemented take place. Furthermore, operational data is converted and software is installed. Training and writing the user manual belong to an application's implementation phase.

Moment and type of analysis

No function point analysis is executed during the implementation phase.

3.9.6 FPA during the operation and maintenance phase

The application is now operating and should be managed and maintained in an adequate fashion.

Moment and type of analysis

Small functional changes usually belong to the operation and maintenance phase. After a change has been implemented, a function point analysis must be done in order to re-establish the amount of functionality that must be managed and maintained.

In the event of major enhancements, a new project is usually started and function point analysis takes place as indicated above.

Objective of the analysis

During this phase, use can be made of the functional size of the operational application in order to make a link between the amount of functionality to be maintained and the maintenance effort required.

Another option worth investigating is the link between the quantity of function points of the applications installed and the costs of having these applications run on the available hardware.

Required documentation

All of the documentation produced up to now should be present. The functional size of all the applications to be installed and maintained must be included in this documentation.

4 General FPA guidelines

This clause provides general guidelines that apply when a function point analysis is performed. Each subclause gives a general FPA guideline from a certain perspective. The titles of the subclause headings indicate the perspectives.

4.1 Analyzing from a logical perspective

Performing a function point analysis is based on an organization's business activities. A business activity can be supported by one or more functions. Conversely, one function can also support several business activities. How a particular application is or has been realized in a technological sense shall not be taken into consideration. The "logical perspective", after all, is what is important. The technology used shall not influence the functional size of an application.

4.2 Applying the rules

Common sense can be necessary when applying the guidelines. If any deviation from the rules and their intentions are experienced to be necessary, always record and clarify them.

Do not be subjective when establishing the complexity of a function.

4.3 No double counting

Functionality shall be counted only once in an application, regardless of whether one or more users make use of it, regardless whether the functionality shows up in one or more places of the application. This ensures that a particular logical file is counted only once in an application: either as an internal logical file or as an external logical file, but not as both. A function such as "Change customer" appearing several times in an application is counted once, provided that the functions are identical in all cases.

4.4 Built functionality, non-requested functionality

During the realization of an application, pieces of code can be copied from an existing application. As a result, more functionality can sometimes be built into an application than just the desired functionality. Additionally, developers may incorporate refinements into an application that were not requested because they think they are nicer.

It shall be determined whether the supplied functionality corresponds to the requested functionality. Non-requested functionality *does* count as part of the size of the application *supplied*, but *not* as part of the size of the application *requested*.

Requested functionality is the functionality initially specified and the functionality that results from later change requests. In diagram form:

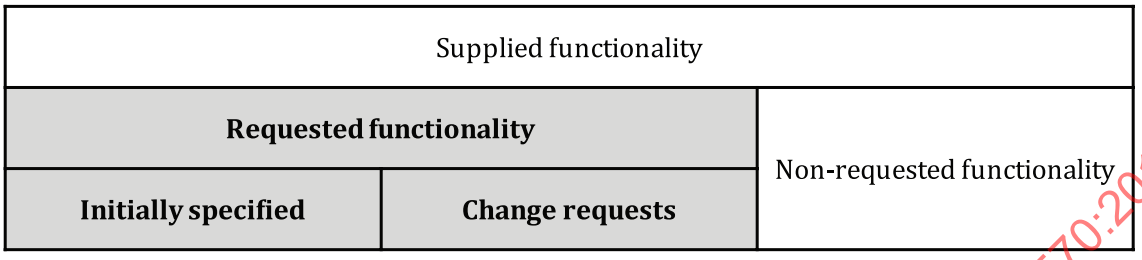


Figure 3 — Requested functionality

4.5 Production of re-usable code

Sometimes software is set up in such a general way that it is re-usable in other applications. This kind of software can be used as a functional unit outside of the application as well. The production of re-usable code does not influence the number of function points.

4.6 Re-use of existing code

If existing code is re-used when constructing an application, existing software is used to realize a specified functionality. Re-use makes it easier to produce a certain functionality. This functionality is included in the functional size in the usual way. In such a situation, it would be wise to work with an adapted productivity standard (fewer hours per function point) for those parts of an application in which full or partial re-usability is possible.

4.7 Screens, windows and reports

Screens, windows and reports are representations of the message traffic between an application and its user(s). As such, they form physical structures of messages exchanged between the application and its environment. If a description of the logical structure is lacking (i.e. if a description of the data element types which belong to the different screens and reports is not present), it will be necessary to deduce this description from the (physical) screens, windows and reports.

Exercise caution here. A physical structure can consist of several logical structures, and a logical structure can consist of several physical structures.

4.8 Input and output records

Input and output records are representations of the communication between an application and other applications. As such, they form physical structures of messages exchanged between the application and its environment. If the logical structure is missing, it will be necessary to deduce it from the (physical) record layout.

Caution should be exercised here. This kind of physical record can consist of several logical structures and a logical structure can consist of several physical records.

4.9 Security and authorization

Security functions, authorization functions, and log-on functions are often standard and, in principle, are available to all applications; therefore, they are not counted. However, if they must be built for the application to be counted, then they shall be included in the functional size as well.

4.10 Operating systems and utilities

Operating systems and utilities are standard in almost every machine and, in principle, are available to all applications; therefore, they are not counted.

Modifying and tailoring an operating system to a specific application has repercussions on productivity. It does not add any functionality and shall not influence the functional size as a result.

4.11 Report generators and query facilities

Three types of report generators and query facilities can be identified:

— Standard facilities provided by the development environment with which the user defines the selections and output products

— Specially made facilities included in an application with which the user defines selections and output products

— Regular external outputs and external inquiries in which selections and output products are fixed

Report generators and query facilities provided as part of the development environment are not counted when the size of a project or an application is being determined.

Report generators and query facilities built at the request of the user are counted as if they are a part of the application. Count the functions (internal logical files, external logical files, external inputs, external outputs, and external inquiries) on the basis of the message traffic with the user that is needed in order to compose output products and to save the queries defined.

Regular external outputs and external inquiries should be counted as indicated in Clauses 8 and 9 even when they have been created with the help of a standard report generator or query facilityExternal inquiry: query facility.

4.12 Graphs

Just as reports, graphs can be considered output. The function point analysis here does not revolve around the technique required to make a certain graph and to show it to the user, but rather around the information in the graph that is used or shown. To determine an output's complexity, therefore, FPA must use the data element types of a graph that crosses the application boundary.

4.13 Help facilities

If an application has help facilities, then one external inquiry must be counted for the entire application for each type of help facility found.

Examples of help facilities include, but are not limited to, the following:

— Help information for the entire application

— Help information for screens or windows

— Help information for fields (including list functions with fixed values) (see subclause 4.16)

— Interactive help wizard

— Context-sensitive help index

Consider the following: If help information can be called up at every screen or window, count only one external inquiry for the application as a whole. In total, there can be a maximum of as many external inquiries as there are types of help facilities in the application to be counted. The file in which help texts are stored is considered to be an FPA table (see subclause 4.20) if the help texts can be maintained.

The complexity of the help facilities' identified external inquiries must be valued as *Low* in a detailed function point analysis and must be valued as *Average* in a high level function point analysis (see subclause 3.2.2).

4.14 Messages

Messages are broken down into two types:

— computer system messages and

— function messages.

Computer system messages are generated by the operating system or other system software. These messages are not counted.

Function messages are generated by a transaction of an application and say something about the use of that transaction.

If function messages are linked to an external input, external output, or external inquiry, then an additional data element type is counted for all the messages together involved in the given function.

Function messages bearing on the use of several transactional functions or on the repeated use of the same transactional function (e.g. a log report or an error report) are counted as output functions according to the guidelines stated in Clause 8.

Note *When there is a separate entity for the messages and it can be maintained by the user, consider this entity as an FPA table (see subclause 4.20).*

4.15 Menu structures

Menu structures should not be counted as a function. One data element type *is* counted, however, for each identified transactional function for the initiation of the transaction, regardless of the actual number of steps required in the menu structure. If the user himself can adapt the menu structure and the menu texts, then these functions and their corresponding logical files are counted according to the guidelines set out for counting logical files and external inputs.

4.16 List functions

Showing a list from which a user can make a selection (e.g. a selection screen, window, pick function, pick list, list box, or pop-up function) is counted as an external output in accordance with the guidelines found in subclause 4.4. Showing a list is not considered an external inquiry because the size of such a list is not known beforehand. Any selection option available does not count as a separate function.

Bear in mind that when a list displays data stored in an entity of the FPA table type (see subclause 4.20), no separate function is counted because the functions for FPA tables ILF and the FPA tables ELF are determined for the group as a whole in a standard fashion. See subclause 4.20.

If the list shows data that is not in a logical file (internal logical file or external logical file) and not in an FPA table, then the function should be considered as a help function at field level (see subclause 4.13).

4.17 Browse and scroll functions

If an application produces output on the basis of a non-unique criterion or on the basis of "from", then it is counted as an external output. This is done when the selection is presented on an overview screen (one *line* for each item that satisfies the criterion), as well as when the user can browse through the detailed screens involved (one *screen or window* per item). No additional functions or data element types are counted for being able to browse or scroll through the output.

4.18 Cleanup functions

In an application, functions can appear that are started on-line or automatically in a periodic fashion and that delete or archive old data. If these functions have been made to satisfy requirements that the user has imposed, count one external input for each cleanup function, taking the general guidelines into account for identifying and valuing external inputs (see Clause 8). Count at the level of functions and do *not* count one external input per internal logical file.

4.19 Completeness check on the function point analysis

Expect at least one external input, one external output, and possibly one external inquiry for each internal logical file. For every external logical file, expect at least one external output or one external inquiry. Also realize that an external logical file may also be read for validation or edit purposes only, so that no external output or external inquiry is needed. If these functions are missing, ask the user whether something has been forgotten and whether the file is really relevant to the application involved.

4.20 FPA tables

Entity types in an application with constants, text, decoding, and so on are referred to as FPA tables. FPA tables that can be maintained by the user with the help of the application to be counted are counted together as one internal logical file: the FPA tables ILF. FPA tables maintained by another application together form one external logical file: the FPA tables ELF. If an entity type cannot be maintained, it may be a system table. FPA does not include this in its counting.

The following criteria must be used in order to determine whether an entity type seen from the FPA should be counted as an FPA table. As soon as one of the criterion has been satisfied, the entity type is an FPA table.

An entity type is an FPA table in the following cases:

1. The entity type can and must contain one and only one item of data (no more and no less), regardless of the number of data elements.

   Example An entity type with data about a particular organization (e.g. name and address).

2. The entity type contains only data that is constant (in principle).

   Example An entity type "chemical elements": mnemonic, atomic number, description (all data element types are constants).

   Example The function point table for valuing function types as illustrated in subclause 2.9, where all data elements are constants too.

3. The entity type consists of a (possibly compound) key + one or more explanatory descriptions, provided that the explanations are similar.

   Example Country: Country-code, Country-name-English, Country-name-French (e.g. NL, the Netherlands, les Pays-Bas)"

4. The entity type contains boundary values, algorithms, and minimum or maximum values, provided that the key is single.

   Example Telephone number range: range number, lowest telephone number, highest telephone number.

The following entity types are not an FPA table:

1. Entity types with amounts, rates, and (VAT) percentages, if they are not constants.

2. Entity types with several different kinds of data (except for those listed above).

   Example Buyer data: buyer nr, buyer name, district-name (district name is a different data element type).

Notice that you must always determine whether an entity type makes up a logical file by itself or together with other entity types (see subclause 4.21).

Note *The above summary of entity types that are either an FPA table or (a part of) a logical file does not cover all possible cases. When in doubt, evaluate entity types within the context of this International Standard.*

Do the following to determine the complexity of the FPA tables ILF and the FPA tables ELF:

— Count the number of different FPA tables that belongs to the group as the number of record types

— Count the number of different data elements of all the FPA tables together as the number of data element types

Additionally, one external input, one external output, and one external inquiry are always counted for the FPA tables *ILF*. No external inputs, outputs, or inquiries are counted for the FPA tables *ELF*.

Do the following to determine the complexity of the standard external input, external output, and external inquiry for the FPA tables ILF:

— count the number of different entity types that belong to the FPA tables ILF as the number of referenced logical files;

— count the number of data elements of all the entity types that belong to the FPA tables ILF as the number of data element types.

4.21 Deriving logical files (data functions) from a normalized data model

4.21.1 Introduction

In FPA, a logical file (an internal logical file or an external logical file) is a conceptual entity type. A conceptual entity type is made up of one or more entity types from a data model in third normal form that, together, are considered one logical unit by a user.

The term "entity type" in this subclause refers to an entity type in a data model in third normal form.

If a data model in third normal form (or equivalent) is at your disposal, you will be able to determine the logical files (data functions) using the rules stated below.

Pay attention to the following matters when applying these guidelines:

— Logical files must sometimes be counted that are not in the normalized data model (e.g. historical files containing aggregated data). See subclause 5.2.j.

— Entity types can appear in the normalized data model that should never have been included in it (e.g. temporary files). Even though such entity types appear in the data model, they are not logical files.

— Always look critically at the data model and at the nature of the relationships, particularly at the cardinality and optionality.

4.21.2 Denormalization rules

To derive the logical files from a data model in third normal form one shall carry out the following steps in this sequence:

1. Determine which entity types in the data model are FPA tables, and so belong to the FPA tables ILF or the FPA tables ELF. FPA tables are valued in a specific way (see subclauses 4.20, 5.2.k and 6.2.g).

2. Determine which entity types are a "Intersection entity: see key-key entitykey-key entity" *without* other attributes. These represent an n:m relationship in the normalized data model and are not valued at all. The referring attribute (foreign key) is counted as a data element type for both logical files connected by this key-key entity.

3. Determine which entity types are a "key-key entity" *with* other attributes. Notice that two situations can arise here as a result:

   a. The additional attributes are technical in nature (not requested by the user, e.g. a date/time stamp) and are not counted as data element types. If they are the only data element types, then the entity type should be dealt with as indicated in step 2 above.

   b. The additional attributes are functional in nature (required by the user), in which case, they should be treated as indicated in step 4.

4. Examine the remaining entity types as to whether they are a logical file on their own or whether together, with one or more related entity types, they make up a logical file. Determining factors are:

   — The nature of the relationship(s) with another entity type (cardinality and optionality)

   — The dependence or independence of the entity type's existence Both of these ideas are examined further below (see subclauses 4.21.3 and 4.21.4).

After the nature of the relationship(s) has been determined, you can assess how the entity types involved should be considered using the table in subclause 4.21.5.

4.21.3 The nature of the relationship (cardinality and optionality)

Two entity types, Project and Employee for example, can be connected to each other via a relationship (e.g. "has").

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

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

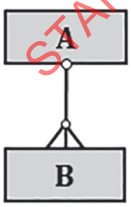
If the business rule was that an employee does not have to work on a project, the "1-side" of the relationship would be optional, and we would denote the relationship between Project and Employee as (1):N

In other situations, the "N-side" could also be optional: 1:(N) or (1):(N).

4.21.4 Independence or dependence of an entity type

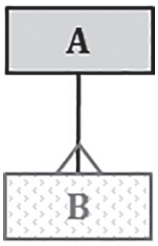
Entity independence is the degree to which an entity type is meaningful in and of itself without the presence of other entity types. Below we show how you can determine whether an entity type is independent or not within the context of different situations that bear on optionality and cardinality.

Entity independence in a (1):(N) relationship



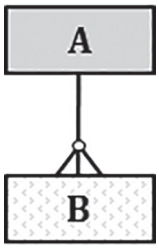
If a relationship between two entity types (A and B) is bilaterally optional, it means that the entity types can exist independently. In such a case, FPA sees each entity type as *entity independent* with regard to each other. As the table in subclause 4.21.5 indicates, entity types A and B each form a logical file in FPA.

Entity dependence in a 1:N relationship



If a relationship between two entity types (A and B) is bilaterally mandatory, it means that each entity type cannot exist without its counterpart, which is they cannot exist independently of each other. In this case, FPA sees the entity types as *entity dependent*. As the table in subclause 4.21.5 indicates, entity types A and B together form one logical file in FPA.

Entity dependence or independence in a 1:(N) relationship



If a relationship between two entity types (A and B) is optional, it means that an A may exist to which no Bs are linked (e.g. as in the 1:(N) relationship between Project and Employee).

In such a situation, the idea of *entity independence* for entity type B plays a role in FPA. What happens to the optional side of the relationship (in this case B) when we want to delete the non-optional side of the relationship (A) when Bs are still linked to it?

Two essentially different situations are distinguished here:

1. The deletion of A is allowed and all Bs linked to it are deleted in the same action (possibly after a message requests confirmation in which it is stated that all Bs will be deleted automatically with the deletion of A).
2. The deletion of A is not allowed as long as Bs are still linked to it.

In situation 1, the Bs are apparently not significant to the business unless they are related to an A, whereas in situation 2 they *are* significant.

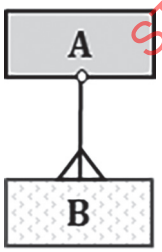
Before you are allowed to delete A in the latter case, you will first have to delete all the related Bs on purpose or link these Bs to another A.

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

As the table in subclause 4.21.5 indicates, entity types A and B in FPA form one logical file in situation 1, whereas in situation 2 they each form a separate logical file.

In the example citing the relationship between Project and Employee, Employee is normally entity independent, because it is not desirable to have employee data deleted if the data of a project is deleted.

Entity dependence or independence in a (1):N relationship



The concept of entity independence as it pertains to the relationship between A and B in a (1):N relationship can also be dealt with in a similar way. These kinds of relationships, however, seldom appear in practice. The question you now have to ask yourself is, "what happens to the optional side of the relationship (in this case, A) when we want to delete the final B (the non-optional side of the relationship) linked to A?"

Two essentially different situations are distinguished here as well:

1. The deletion of a final B linked to A is allowed in which A is also deleted in the same action (possibly after a message requests confirmation in which it is stated that A will be deleted automatically with the deletion of B).

2. The deletion of B is not allowed as long as A is still linked to B.

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.

Before you are allowed to delete the final B in the latter case, you will first have to delete the related A on purpose, or you must link this A to one or more other Bs.

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

As the table in subclause 4.21.5 indicates, entity types A and B in FPA form one logical file in situation 1, whereas in situation 2 they each form a separate logical file.

4.21.5 Conversion table: from normalized entity types to logical files

In the table on the following page, A and B are two entity types (not an FPA table and not a key-key entity) (see subclause 4.21.2 point 1, 2, and 3) from the normalized data model that are connected to each other via a relationship.

Table 3 — Conversion table from normalized entity types to logical files

Type of relationship between A and B	How to count A and B	Condition
(1) : (N)	2 LFs	
1 : N	1 LF, 2 RETs, sum DETs	
1 : (N)	1 LF, 2 RETs, sum DETs	When B is entity dependent on A
	2 LFs	When B is entity **in**dependent of A
(1) : N	1 LF, 2 RETs, sum DETs	When A is entity dependent on B
	2 LFs	When A is entity **in**dependent of B
(1) : (1)	2 LFs	
1 : 1	1 LF, 1 RET, sum DETs	
1 : (1)	1 LF, 2 RETs, sum DETs	When B is entity dependent on A
	2 LFs	When B is entity **in**dependent of A

Legend: LF = Logical file (ILF or ELF)
RET = Record type
DET = Data element type

Note *When in doubt, choose entity independent.*

Bear in mind that more than two entity types can also form a logical file sometimes. In such a case, count the total number of entity types as the number of record types. In the event of a bilateral mandatory (1:1) relationship, one logical file with one record type is always the rule.

4.22 Shared use of data

The situations below provide guidelines for identifying functions when two applications share the use of data.

Situation 1

Application A has an internal logical file *Customer*. The current data in it is available to application B.	**Application B** uses the current data from the logical file *Customer* in application A.

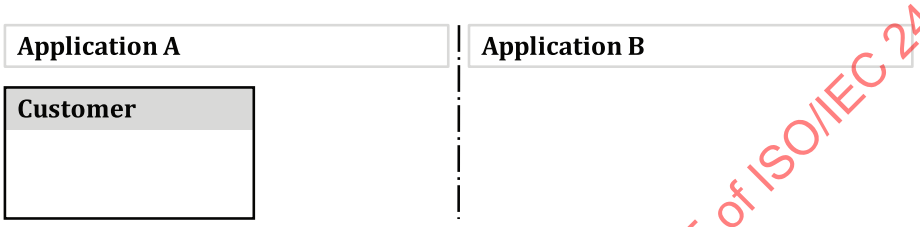


Figure 4.1 — Shared use of data, situation 1

Solution

For **Application A**, *Data* is an internal logical file.	For **Application B**, *Data* is an external logical file.

Situation 2

Application A makes a copy of the logical file *Customer* for application B. This copy exits application A.	**Application B** processes the copy supplied to one or more of its own internal logical files. The data from *Customer copy* is used one time only (is consumed).

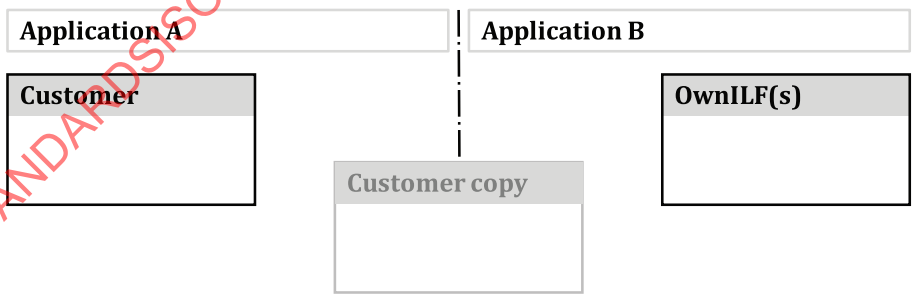


Figure 4.2 — Shared use of data, situation 2

Solution

For **Application A**, *Customer* is an internal logical file. Creating *Customer copy* is an external output. *Customer copy* is not a separate logical file.	For **Application B**, *Customer copy* is used only once and is apparently not a permanent file. *Customer copy* is therefore not a logical file for application B. Reading in and processing *Customer copy* is an external input.

Situation 3

Application A periodically makes, for functional reasons, an extract of the data from the logical file *Customer* (*Customer extract*) so that other applications can reference it. *Customer extract* remains within the boundary of application A. Differences can occur between the current data in *Customer* and the data in *Customer extract*.	**Application B** makes use of *Customer extract*.

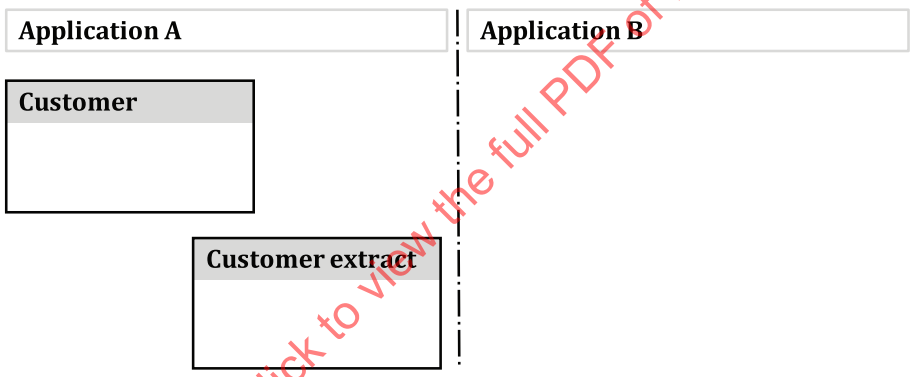


Figure 4.3 — Shared use of data, situation 3

Solution

For **Application A**, *Customer* is an internal logical file. *Customer extract* is also an internal logical file for application A. Creating *Customer extract* is an external input.	For **Application B**, *Customer extract* is an external logical file.

Situation 4

Application A periodically makes an extract from the logical file *Customer* (*Customer extract*) that is subsequently distributed to other applications. *Customer extract* exits application A.	**Application B** reads *Customer extract* in and the file is saved into a file *Customer copy* without undergoing any additional processing.

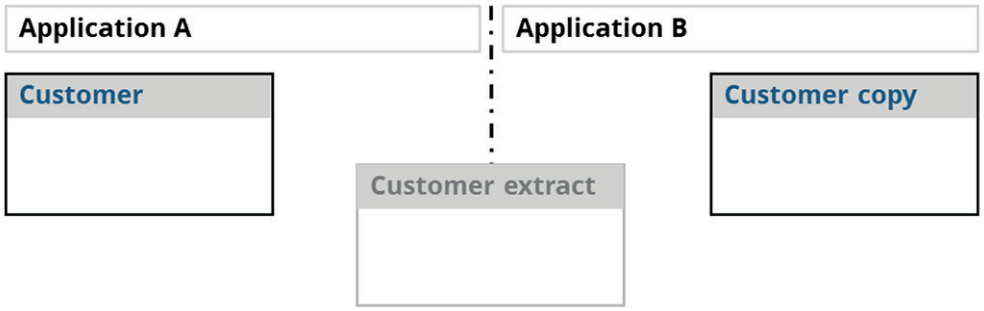


Figure 4.4 — Shared use of data, situation 4

Solution

For **Application A**, *Customer* is an internal logical file. Creating *Customer extract* is an external output. *Customer extract* is not permanent data for application A and is not counted as a logical file.	For **Application B**, *Customer copy* is an internal logical file. Reading in *Customer extract* is an external input. *Customer extract* is not permanent data for application B and is not counted as a logical file.

Situation 5

Application A makes an extract of the data from the logical file *Customer* (*Customer extract*) that is subsequently distributed to application B. *Customer extract* exits application A.	**Application B** processes the data from *Customer extract* in order to update one or more of its internal logical files. The data is used one time only (is consumed).

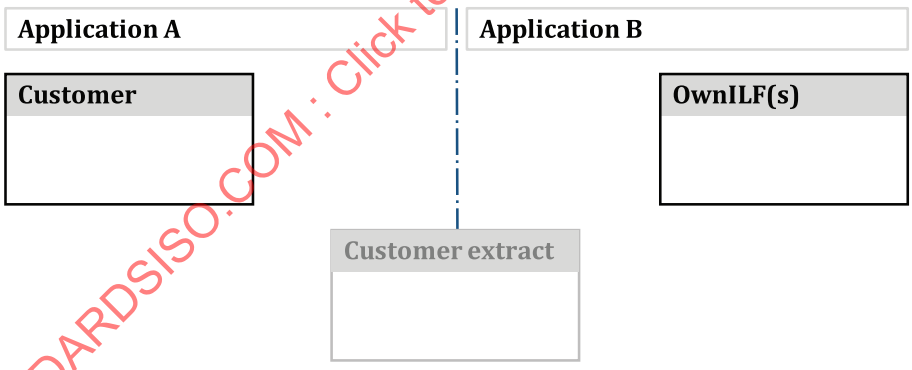


Figure 4.5 — Shared use of data, situation 5

Solution

For **Application A**, *Data* is an internal logical file. Creating *Data extract* is an external output. *Data extract* is not permanent data for application A and is not counted as a logical file.	For **Application B**, reading in and processing *Data extract* is an external input. *Data extract* is used only once and therefore does not contain any permanent data. *Data extract*, then, is not a logical file for application B.

Note *In order to be clear, and in keeping with the above mentioned guidelines, a file created several times or stored at different physical locations via the same logical processing and with the same layout, should be counted as one transaction or as one logical file.*

4.23 Generic rule for counting data element types

In Clauses 7.3, 8.3 and 9.3 guidelines are stated for counting data element types (DET) for transactional function types. These are concrete expressions of one generic rule for counting data element types. In this clause the generic rule is stated and explained.

Generic rule

For all transactional function types one generic rule applies to determine the amount of data elements:

Count all functional data that crosses the boundary of the application (inbound or outbound)

For optimal recognition and usability the Clauses 7.3, 8.3 and 9.3 connect to the characteristics of each transactional function type (EI, EO, EQ). These clauses provide guidelines from the perspective of that particular transactional function type. These guidelines seem to be specific rules, but in fact they are concrete expressions of this generic rule.

Due to the above rule, the following data are counted as DET:

— Editable fields (provided they cross the boundary of the application).

— Displayed fields from the logical files and FPA tables.

— Initiation trigger/functional control data: for initiating a transaction exactly one DET (no more, no less) shall be counted, regardless of the number of locations, ways or steps from which a transaction can be started (e.g. multiple menus or function keys/buttons on the screen or window associated with the transaction itself or function keys/buttons in screens belonging to other transactions). This initiation trigger must originate from outside the application, so that control data crosses the boundary of the application. Background thought is here that the way the transaction is initiated is an implementation decision. The data element type is the same.

— Messages: regardless of the number of computer system messages and function messages in a transactional function, exactly one DET shall be counted for all messages. This is because it is one type.

— Entered selection data.

— Displayed derived data.

— Other entered data such as printer selection, filename, etc.

As a result of the above generic rule the following data are *not* counted as DET:

— Page-up/page down keys

— Column headers/descriptions

— Headers

— Field labels

— Page Number

— System Date

— Internally updated fields

5 Internal Logical Files

This clause further examines the data groups that the user considers to be a single logical unit and that the application to be counted maintains. Usually these data groups are recorded in a conceptual data

model. The clause shows which internal logical files must be identified within the context of FPA. The number of internal logical files and their complexity contribute to the functional size.

5.1 Definition of an internal logical file

An internal logical file (ILF) is a logical group of permanent data seen from the perspective of the user that meets each of the following criteria:

— it is *used* by the application to be counted *and*

— it is *maintained* by the application to be counted.

***A logical group of data seen from the perspective of the user* is:**

a group of data that an experienced user considers as a significant and useful unit or object.

An equivalent to this kind of logical group of data is an object type in data modeling.

***Permanent* means:**

that the file remains in existence after the application has used it so that it can be used again, unlike data in other instances that is "consumed", used once.

***Used* means:**

that the data is also actually made use of in the processes of the application.

***Maintained* indicates:**

that it is possible to add, change, or delete data.

5.2 Identifying internal logical files

5.2.a When identifying internal logical files, start with a conceptual data model in which data groups (object types) have been specified in a usable, identifiable, significant, and comprehensible way for the user. You cannot simply assume a data model in third normal form. The entity types from a normalized data model should be grouped at a conceptual level here (see the guidelines in subclause 4.21).

5.2.b The maintenance of data (adding, changing, or deleting) is of decisive significance. Only data groups used and maintained in the application under consideration shall be counted.

5.2.c If there are no maintenance functions established for a candidate internal logical file in your function point analysis, there are three possible reasons for this:

— The file is a technological solution and, therefore, shall not be counted as a logical file.

— The file is maintained by another application. When such is the case, the file is not an internal logical file, but rather an external logical file, or an input transaction file to be counted as an external input.

— The file is an internal logical file, but no maintenance functions have been defined for it when there should have been. This is usually an omission in the functional user requirements.

5.2.d For each internal logical file, expect at least one external input, in addition to one external output or one external inquiry.

5.2.e If an internal logical file is not accessible to a user by means of an external input, determine whether it has been identified correctly as an internal logical file.

5.2.f Files introduced for technological reasons should not be counted (e.g. work files, temporary files, interim files, sort files, print files, spool files, and so on).

5.2.g If the user requests a restart-recovery mechanism for which (for example) a check-point data file is needed, do not include this file in the count. Additionally, do not count this file when determining the complexity of the functions.

5.2.h The presence of different user views, access paths, and/or indexes for a file identified as an internal logical file does not mean that several internal logical files are counted for this file.

5.2.i Historical files are counted as an internal logical file only if the set of data element types of a historical file is unique in relation to other files.

5.2.j Stay alert for historical files. The user commonly requires them, but does not always specify them on time.

5.2.k Entity types with constants, text, decoding, and so on within an application are counted together once as a group in the application as one internal logical file (the FPA tables), provided that each of these entity types can be maintained in the application. Additionally, one external input, one external output, and one external inquiry are counted by default for the FPA tables ILF. If one of the entity types of the kind listed above cannot be maintained in the application, then it shall not be counted as part of the FPA tables ILF (see the guidelines in subclause 4.20).

5.2.l Be alert for files in which only derived data is maintained (e.g. a running register). These files shall be counted as an internal logical file only if the user has specified them explicitly as a data group. If the file has been included for technological reasons (e.g. performance), then it shall not be counted.

5.2.m Sometimes it is clear from the information requirements specified by the user that a file is necessary even though it does not appear in the conceptual data model. In such a case, the file should be counted as an internal logical file if it is the result of a functional requirement and can be maintained in the application. If a file is needed for technological reasons, it shall not be counted.

5.2.n The internal logical files of an existing application that remain unchanged are not counted as internal logical files or as external logical files for the *project* functional size of an enhancement project.

5.3 Determining the complexity of internal logical files

Data element types

5.3.a Only those attributes (intended to be) used and/or maintained in the application to be counted are counted as data element types. This means that not all attributes are counted per se.

5.3.b All the data element types of the FPA tables involved in the FPA tables ILF are counted together, in so far as they are (intended to be) used and/or maintained in the application to be counted.

5.3.c If several entity types in a normalized data model together form an internal logical file, then the number of data element types of the internal logical file corresponds to the sum of the data element types (attributes) of the normalized entity types. In order to prevent double counting, the referring attributes in the entity types compiled into the one internal logical file are not counted.

Record types

5.3.d The number of record types for determining the complexity of an internal logical file is:

— equal to the number of enclosed entity types for all other internal logical files (see sub-clause 4.21);

— equal to the number of enclosed FPA tables for the FPA tables ILF.

Complexity matrix

The following table is used to determine the complexity of an internal logical file:

Table 4 — Complexity matrix internal logical files

Record Types	Data Element Types		
	1 - 19	**20 - 50**	**51 or more**
1	Low	Low	Average
2 - 5	Low	Average	High
6 or more	Average	High	High

The availability of data regarding the number of record types and the number of data element types depends on the phase of the application's life cycle. If this data is not known, or when you perform a high level analysis, an identified internal logical file should be valued as *low*.

6 External Logical Files

This clause further examines the data groups that the user considers to be a single logical unit and that the application to be counted uses, but which another application maintains. Usually these data groups are recorded in a conceptual data model. The clause shows which external logical files must be identified within the context of FPA. The number of external logical files and their complexity contribute to the functional size.

In previous releases of this guideline, this type of logical file was known as external interface file. Since this term can be associated with a certain technical implementation, this term has been changed to external logical file, indicating that it is similar in nature to an internal logical file, but maintained by another application.

6.1 Definition of an external logical file

An external logical file (ELF) is *a logical group of permanent data seen from the perspective of the user* that meets each of the following criteria:

— it is used by the application to be counted *and*

— it is not maintained by the application to be counted *and*

— it is maintained by another application *and*

— it is directly available to the application to be counted.

***A logical group of data seen from the perspective of the user* is:**

a group of data that an experienced user considers as a significant and useful unit or object.

An equivalent of this kind of logical group of data is an object type in data modeling.

***Permanent* means:**

that the file remains in existence after the application has used it so that it can be used again, unlike data in other instances that is "consumed", used once.

***Used* means:**

that the data is also actually made use of in the processes of the application.

***Not maintained by the application to be counted* means:**

that it is not possible to add, change, or delete data in the application to be counted.

***Directly available to the application to be counted* means:**

that the application concerned always has the current data from the logical file at its disposal, even though another application maintains this logical file.

6.2 Identifying external logical files

6.2.a When identifying external logical files, depart from a conceptual data model in which data groups (object types) have been specified in a usable, identifiable, significant, and comprehensible way for the user. You cannot simply assume a data model in third normal form. The entity types from a normalized data model should be grouped at a conceptual level here (see the guidelines in subclause 4.21).

6.2.b A file is counted as an external logical file only when it is maintained by another application than the one to be counted, but its current data is always available to the application to be counted (see subclause 4.22 for a further explanation).

6.2.c If an exchange of data takes place between applications via a transaction file, the transaction file is not counted as an external logical file but as an external input and/or external output instead. Remember, an external logical file must contain functionally permanent data that can be used in the application more than once, unlike a transaction file whose data is consumed (used only once) by an application (see subclause 4.22 for further explanation).

6.2.d Expect at least one external output and/or one external inquiry for each external logical file. Occasionally, however, an external logical file is used only to permit the edit, audit or validation of the data element types on an external input.

6.2.e An external logical file must be an internal logical file of another application.

6.2.f The presence of different user views, access paths, and/or indexes in a file identified as an external logical file does not mean that several external logical files are counted for this file.

6.2.g Entity types with constants, text, decoding, and so on that are referenced in the application but maintained by another application are counted together once as a group as one external logical file (the FPA tables ELF) (see the guidelines in subclause 4.20).

6.2.h Sometimes it is clear from the information requirements specified by the user that a file is necessary even though it does not appear in the conceptual data model. If such a file is made available by another application and the application to be counted always has the current data available from that file, then the file must be counted as an external logical file. If the file has been included for technological reasons (e.g. performance), then it shall not be counted.

6.2.i A logical file shall be counted as an external logical file in an application only when it is not an internal logical file for this application that is in an application that does not maintain the logical file.

6.2.j When determining the application functional size, count a logical file used communally by several sub-systems of the same application once, either as an external logical file or as an internal logical file. A logical file can be counted as an external logical file only if the boundary of the application has been crossed.

6.2.k When determining a project functional size, count a logical file used communally by several sub-systems of the same application either as an external logical file or as an internal logical file for each of these sub-systems, if these sub-systems are realized in a corresponding quantity of sub-projects implemented more or less in parallel, and these sub-systems have been constructed in such a way that they must be able to exist independently (e.g. because of a phased implementation of the application or because of functional reasons).

6.2.l If the application is maintained and supported as a whole after the completion of the separate projects, this logical file is counted as indicated in subclause 6.2.j when the application functional size is being determined.

6.2.m The external logical files of an existing application that remain unchanged are not counted as external logical files for the project functional size of an enhancement project.

The following table provides an overview to help you distinguish between internal logical files and external logical files.

Table 5 — Overview to distinguish between internal logical files and external logical files

Situation		Count as	
In the application to be counted	**In the other application**	**In the application to be counted**	**In the other application**
Use only	Use and maintenance	**ELF**	**ILF**
Use and maintenance	Use only	**ILF**	**ELF**
Use and maintenance	Use and maintenance	**ILF**	**ILF**
Use only	Use only	*Not applicable**	*Not applicable**

ILF = Internal logical file

ELF = External logical file

* = Logical files cannot just be *used*. There must always be an application that is responsible for its maintenance.

6.3 Determining the complexity of external logical files

Data element types

6.3.a Only those attributes (intended to be) used in the application to be counted are counted as data element types. This means that not all attributes are counted per se.

6.3.b All the data element types of the FPA tables involved in the FPA tables ELF are counted together so far as they are (intended to be) used in the application to be counted.

6.3.c If several entity types in a normalized data model forms together an external logical file, then the number of data element types of the external logical file corresponds to the sum of the data element types (attributes) of the normalized entity types. In order to prevent double counting, the referring attributes in the entity types compiled into the one external logical file are not counted.

Record types

6.3.d The number of record types for determining the complexity of an external logical file is:

- equal to the number of enclosed entity types for all other external logical files (see sub-clause 4.21);
- equal to the number of enclosed FPA tables for the FPA tables ELF.

Complexity matrix

The following table is used to determine the complexity of an external logical file:

Table 6 — Complexity matrix external logical files

Record Types	Data Element Types		
	1 - 19	**20 - 50**	**51 or more**
1	Low	Low	Average
2 - 5	Low	Average	High
6 or more	Average	High	High

The availability of data regarding the number of record types and the number of data element types depends on the phase of the application's life cycle. If this data is not known, or when you perform a high-level analysis, an identified external logical file should be valued as *low*.

7 External Inputs

This clause further examines functional user requirements pertaining to the maintenance of internal logical files and the processing of control information in the application to be counted. It shows which external inputs must be identified in FPA. The degree to which external inputs contribute to the functional size depends on their quantity and their complexity.

7.1 Definition of an external input

An external input (EI) is a *unique* function recognized by the user in which *data* and/or *control information* is entered into an application from outside that application. It is a function that the user must see as an *elementary process*.

***Data* is:**

data that causes an addition, change, or deletion of data in one or more internal logical files

***Control information* is:**

data (e.g. a signal) that activates or deactivates one or more processes of an application or that influences the effect of a transaction

An external input should be considered *unique* when it:

— consists of a set of data element types different than all other external inputs, or

— consists of the same set of data element types, but requires a *different logical way of processing*

A *logical way of processing* is:

a method specified by the user in order to effect a desired result. This means the following within the context of external inputs:

— Maintaining and possibly referencing logical files

For example: When a new order is added, the file "Product" is referenced in order to see whether the product being ordered really can be ordered.

— Carrying out algorithms, calculations, and validations

For example: When a new order is added, logical processing consists of validations (among other things) that should be carried out to determine whether the data entered is correct.

A logical way of processing is considered *different* when:

— other logical files are maintained or referenced

— the same internal logical files are maintained, but there are different algorithms, calculations, and/or validations

Different technological solutions chosen to realize the same logical processing do not mean that the logical way of processing is different.

A function is an *elementary process* when two criteria are satisfied:

— The function has an autonomous meaning to the user and fully executes one complete processing of information. In other words, it is self-contained.

For example: A user enters a new employee, including the employee's salary data and his family status. From the user's perspective, entering the employee is a single and complete process. Entering the salary information and the employee's family status is a part of this process and is not separate.

— After the function has been executed, the application is in a consistent state.

For example: A user has required that an employee's salary data must be recorded when an employee's name is being entered. Entering only the employee's name or only the salary data of the employee results in an application in an *in*consistent state.

An external input can entail both the data and control information that a user enters directly and the information that has been received from other applications.

7.2 Identifying external inputs

7.2.a Count each input of data in so far as it:

- is an elementary process *and*
- has been specified by the user *and*
- is unique in the application to be measured *and*
- crosses the external boundary of the application *and* (usually)
- results in an addition, change, or deletion of data to, in, or from an internal logical file of the application.

7.2.b When an external input maintains several internal logical files (adds, changes, or deletes data), it is counted as a single external input if the user sees it as a whole and the opportunity to maintain each of the different files individually is not provided.

7.2.c If internal logical files can be maintained individually, then at least one external input is identified for each internal logical file.

7.2.d Count both the input of data the user enters directly and the input of data originating from other applications (e.g. in the form of an input file or a message). An external input can be activated by a user or by another application. It can also be activated automatically (e.g. a batch function started automatically).

7.2.e Files with functionally permanent data from another application shall not be counted as external inputs, but as external logical files.

7.2.f Expect at least one external input for each internal logical file, though usually there will be several. Consult with the user when external inputs are lacking.

7.2.g An input screen or window on which different functions can be effectuated (add, change, or delete) counts as different individual external inputs.

7.2.h An external input (e.g. to change data) that requests confirmation prior to the execution of the command entered is counted as one external input, because it pertains to a single elementary process.

7.2.i Count two functions when data is added, changed, or deleted in two steps (e.g. in order to allow another employee to authorize input). The initial input is counted as one external input as is the authorization function. Each step is an elementary process. An external inquiry might appear here if it has been explicitly specified as such.

7.2.j A duplicate external input (i.e. when the set of data element types and the logical processing are the same) counted earlier is not counted again.

7.2.k An external input which has been introduced exclusively for technological reasons (e.g. because of the technology used), but which a user has not requested, is not counted as an external input.

7.2.l Only those functions specified by the user count as external inputs.

7.2.m The menu structure is counted as indicated in subclause 4.15.

7.2.n If data to be changed or deleted is presented as part of the change or delete function of an internal logical file, then this presentation of data is considered to make up a part of the external input and is not counted separately as an external inquiry.

7.2.o The same applies when data is made visible automatically on a change screen or window (e.g. when data in an input file is presented successively). It is not an external inquiry, but a part of the external input.

7.2.p If the external inquiry is specified separately, then functionality is supplied to the user, in which case an external inquiry *is* counted. If retrieved data can then be changed using a change function, both an external inquiry and an external input are counted.

7.2.q If an external input consists of several input screens or windows because not all the data required can be entered onto one screen, count this as one external input. If each input screen or window can also be used separately without having to go through a pre-arranged sequence, however, then count a separate external input for each input screen or window, provided that each screen can be considered an elementary process in and of itself.

7.2.r A time-delay between the input and processing of data is not a reason to identify additional external inputs.

7.2.s Entering data to control an external input, output, or inquiry (e.g. selection data) is not counted as a separate function, but is counted as a part of the function involved.

7.2.t The processing of a transaction file (a file with temporary data) supplied by another application results in several external inputs if different kinds of logical processing have been specified as a result of the data. In other situations, this can occur when several record types have been defined within the transaction file or when different processing codes have been defined within one record type.

7.2.u Count one external input when the input of data for the same logical processing takes place via different media.

7.2.v External inputs that seem to be the same from the outside, but that maintain other internal logical files, must be counted as separate external inputs, because they entail a different logical processing.

7.2.w If a header record and/or a trailer record with check data appears in a transaction file (e.g. total amount or total number of records), the processing of this/these record(s) is not counted as an individual external input. The check data, however, *is* counted as data element types (if requested by the user).

7.2.x Sometimes the opportunity is given to select data that needs to be changed or deleted via a non-unique selection criterion. This means that characteristic data is shown of the items of the entity type that satisfy the selection criteria entered. The desired item of the entity type can then be selected. The act of displaying this data is seen as additional functionality. Count one external output for this (see subclause 4.16).

7.3 Determining the complexity of external inputs

Data element types

The following guidelines for counting data element types are a concrete expression of the generic rule as stated in Clause 4.23.

7.3.a Count all data element types (data and/or control information) that cross the boundary of the application to be counted.

7.3.b Count the way to get to a function and the way to start it (e.g. menu selection and function keys) as one data element type, the initiation trigger.

7.3.c If several functions can be effectuated on an input screen or window (e.g. add, change, and delete), then count the relevant number of data element types for each function.

7.3.d If several screens are used for an external input (e.g. first a screen into which selection data is entered and then a change screen), the combination of the screens together must be considered as a whole when counting data element types, because the function is a single elementary process. If, however, the selection data is not uniquely identifiable and the user must first choose the desired item from a number of items of the entity type selected, then count a separate external output (see subclause 7.2.x).

7.3.e If there are data element types for error messages as well as for other messages, or if there is a separate screen or window to display messages, then count as indicated in subclause 4.14.

7.3.f If additional data is to be displayed in response to an input, then the data element types displayed must be counted as a part of the external input. An example of this kind of additional data is a function "Update customer data" in which a customer number is entered and, for the purpose of verification, the application displays the customer's name and address, after which a user can enter the rest of the data.)

7.3.g If a header record and/or a trailer record with check data appear(s) in a transaction file (e.g. total amount or total number of records), then this data is also counted as data element types (if requested by the user).

File types referenced

7.3.h The number of referenced logical files for each external input is determined by establishing the number of referenced logical files involved in the validation of the input and/or in the execution of the external input. The files can be either internal logical files or external logical files.

7.3.i The FPA tables ILF and the FPA tables ELF are not counted as referenced logical files when the complexity of external inputs is being determined. This also applies to files introduced for technological reasons (see subclause 5.2.f).

7.3.j If a process has been defined in which a logical file with permanent data is maintained on the basis of a transaction file (input), count this as one or more (in the event of several processing codes) external inputs (the processing of the transaction file) with a minimum of one internal logical file (the permanent file).

7.3.k A transaction file (input or output) cannot be a "referenced" logical file. In other words, an external input which, for example, uses an input file in order to process its data, has no referenced logical files unless, of course, internal logical files are updated. This also applies to temporary files.

7.3.l Files such as temporary files, sort files, and print files introduced for technological reasons are not counted. Determine whether these kinds of files are an alternative for an internal logical file or an external logical file. If such is the case, count these underlying logical files to determine the complexity.

Complexity matrix

The following table is used to determine the complexity of an external input:

Table 7 — Complexity matrix external input

File Types Referenced	Data Element Types		
	1 - 4	**5 - 15**	**16 or more**
0 - 1	Low	Low	Average
2	Low	Average	High
3 or more	Average	High	High

The availability of data regarding the number of data element types and referenced logical files depends on the phase of the application's life cycle. If this data is not known, or when you are performing a high level analysis, an identified external input should be valued as *average*.

8 External Outputs

This clause further examines functional user requirements pertaining to output that an application produces. This output varies in size or requires further data processing. The clause shows which external outputs must be identified in FPA. The degree to which external outputs contribute to the functional size depends on their quantity and their complexity.

8.1 Definition of an external output

An external output (EO) is a *unique* output recognized by the user that crosses the application boundary. It varies in size and/or *further data processing* is needed for it. It is a function that the user must see as an *elementary process*.

An external output must be considered *unique* if:

— the output product has a different *logical layout* than all the other output products, or

— the output product has the same *logical layout*, but requires a different *logical way of processing*, or

— any input part of the external output consists of a different set of data element types than the input part of all the other external outputs.

An output product has the same *logical layout* if

the set of data element types is the same. The following is allowed:

— Output product data element types with a different order.

Note *Grouping data element types in different ways (e.g. by means of intermediate headings) is seen as a different logical layout.*

— Data element types appearing in the output product, but without a value. The reasons for this may be that:

a. the data element type does *not* have a value in the logical file (as a result of which the data element type is not present optically), or

b. the data element type *does* have a value in the logical file, but it is not relevant to the user.

***A logical way of processing* is:**

a method specified by the user in order to come to a desired result. This is understood to be the following within the context of external outputs:

— Referencing logical files

For example: When a list of employees is made, the logical file “Employee” is referenced.

— Providing further data processing such as algorithms, calculations, and/or validations.

For example: When reporting about all the employees in an organization, the logical way of processing contains the algorithms needed to calculate the total number of fixed employees, employees under hourly contracts, and all employees.

A logical way of processing is considered *different* when:

— other logical files are referenced

— the same logical files are referenced, but there are other algorithms, calculations, or validations

Selecting with a *different selection value* (whether or not with different kinds of *wild cards*) and selecting on the same field(s), but with a different *operator*, are *not* seen as a different logical way of processing.

Different technological solutions chosen in order to realize the same logical processing do not mean that the logical way of processing is different.

A *different selection value* is:

a selection on the same field(s), but with a different content. Do not confuse this with a selection on different fields.

A *wild card* is:

a specific sign indicating that different values can appear in a particular position. Examples of wild cards commonly used are the question mark ("?", representing an arbitrary character) and the asterisk ("*", representing a connected string of arbitrary characters).

An *operator* is:

a symbol that represents a logical operation.

For example: Commonly used operators are equal to (=), greater than (>), smaller than(<), greater or equal to (≥), smaller or equal to (≤) and not equal to (≠) and arithmetic operators like add (+), subtract (-), multiply (*) and divide (/).

***Further data processing* is:**

the execution of algorithms or calculations made on data that has been retrieved from logical files before information is shown.

A function is an *elementary process* when two criteria are satisfied:

— The function has an autonomous meaning to the user and fully executes one complete processing of information. In other words, it is self-contained.

For example: Consider a function in which all employees hired in a certain year can be printed. Entering the selection data and displaying or printing the employees selected is one complete processing from the perspective of the user.

— After the function has been executed, the application is in a consistent state.

For example: An application for an invoicing system creates an output file containing data about the consumption of users who live in a certain area. Selecting the users who live in the area

desired, searching for the consumption data, constructing the output file, and sending the file are considered a whole. From the user's perspective, the application is in a consistent state only when all the steps have been executed.

External outputs encompass both output products that go directly to the user in the form of reports and messages, as well as data flows that go to other applications via an on-line link or via an output file.

Only the domain of the output product is fixed for external outputs. The size of the output product, however, *does not have to be constant at all*, unlike external inquiries in which the size of the output product is constant.

8.2 Identifying external outputs

8.2.a Count each output of data in so far as it:

- is an elementary process *and*
- has been specified by the user *and*
- is unique in the application to be measured *and*
- crosses the external boundary of the application *and*
- varies in size or requires further data processing (see the definition in subclause 8.1).

8.2.b Count both output products supplied directly in the form of reports and messages to the user and output products supplied as output files and messages to other applications.

8.2.c Use the criteria below to distinguish between an external output and an external inquiry:

- The output of an external output may vary in size.
- An external output does not require input for the selecting of data.
- The output of an external output may contain data that has come about with the help of further data processing (such as the calculation of data).

8.2.d The output part of an external inquiry shall not be counted as a separate external output.

8.2.e Entering data for controlling an external output (e.g. entering selection criteria, the sort sequence desired, or the printer desired) is seen as a part of the external output and shall not be counted as external input.

8.2.f Expect at least one external output for each internal logical file. Consult the user when external outputs are lacking.

8.2.g An output product can comprise several external outputs. A single output product contains several external outputs when:

- the output product contains different logical layouts (see the definition of "logical layout" in subclause 8.1) and these logical layouts can be *retrieved individually*, or
- the output product contains different logical layouts (see the definition of "logical layout" in subclause 8.1) that have been established by different logical ways of processing and are combined for ease-of-use.

Retrieved individually means that the user has the opportunity to control or select which parts are going to be printed.

Individual logical processes are said to be activated when the different parts report about a different object or when they come about as a result of other logical files. In this case, we talk about individual logical processes that result in one combined output product that can be retrieved with one command for the sake of user friendliness.

8.2.h A report that has the same logical layout but that can be sorted in several ways counts as one external output, unless a different or an additional logical processing is needed for each sort sequence.

8.2.i Output can be intended directly for a user, a different application, or an external storage device. If the logical layout and the logical processing are identical, count it as one external output.

8.2.j It may be necessary to count a transaction file for another application as several external outputs. This can be the case, for example, when several record types appear in the file or when the output of different logical processes has been physically compiled in one external output file.

8.2.k Count only the external outputs that the user has asked for. Output products not requested by the user but introduced simply because of the technology are not counted (e.g. spool files).

8.2.l A file with functionally permanent data that the application to be counted maintains and that other applications use is counted as an internal logical file and not as an external output. Other applications that use the file, however, count this data as an external logical file and, therefore, not as an external input.

8.2.m If the user requests an overview of possible error messages, it is not seen as a separate external output, because the file with error messages is an FPA table (see subclauses 4.14 and 4.20).

8.2.n Messages (e.g. error messages) that say something about the execution of one function are linked to that one external input, output, or inquiry. They are not counted as individual external outputs, but are counted together as one data element type for the function concerned (see subclause 4.14).

8.2.o An output product with error messages or messages pertaining to the execution of different functions or to the repeated use of the same function is counted as one or more external outputs.

8.2.p An output product that is the logical result of maintenance to internal logical files (e.g. a transaction report or a processing report) is counted as one or more external outputs.

8.2.q Count an output on the basis of several selection criteria as follows:

When the user has more options (i.e. an "and/or situation"), count the selections that mutually exclude each other. Each selection or combination of selections that exclude all others is counted separately.

8.2.r If a header record and/or a trailer record with check data appear(s) in a transaction file (e.g. total amount or total number of records), the processing of this/these record(s) is not counted as an individual external output. The header record is comparable to the message header of a report and the trailer record to the summarizing totals in a report. The data element types from the header record or trailer record *are* counted (if requested by the user).

8.2.s If the user has the option to start several functions (either individually or in combination), in which the combination of the functions is more than the sum of their parts, you will have combination effects to contend with. Deal with the situation in the following way:

— A separate external output is counted for each function with a different processing that can be started separately.

— In principle, only one additional external output is counted for all possible combinations, unless there are different logical processes for certain (groups of) combinations. In such a case, an additional external output is counted for each (group of) combination(s) for which a different logical processing is needed.

8.2.t Showing a list from which a user can make a selection (e.g. a selection screen, window, pick function, pick list, or pop up function) must be counted as an external output if explicitly requested by the user, provided that the data originates from a logical file and not from an FPA table. Showing a list is not an external inquiry because the size of the list is not known beforehand. Any selection opportunity present is not counted as a separate function (see subclause 4.16 for a further explanation).

8.2.u Do not count any additional functions or data element types for being able to browse or scroll through produced output (see subclause 4.17 for a further explanation).

8.2.v If a function results in output products whose contents are the same but that have been stated in a different language, then the function is counted as one external output because the output products, although different in language, nevertheless consist of the same set of data element types, and processing is the same for all output products.

8.2.w Even though there may be several output products, there can nevertheless be only one external output. There is one external output when:

a) the output products have the same logical layout (see the definition in subclause 8.1) *and*

b) the output products have come about as a result of the same logical way of processing (see the definition in subclause 8.1).

8.3 Determining the complexity of external outputs

Data element types

The following guidelines for counting data element types are a concrete expression of the generic rule as stated in Clause 4.23.

8.3.a All data element types (not their possible values) that appear in the output product generated by the external output are counted.

8.3.b All control information (e.g. selection criteria, desired medium, desired printer, sort sequence, or time period of printing) at the level of data element type that must be entered to produce output are counted as data element types for the external output. Control information appearing on the output itself is counted twice.

8.3.c All address data at the level of data element type that indicates for whom or for which device or medium the output is meant is counted.

8.3.d All process data making up a part of the output product (e.g. averages, results of calculations, subtotals, and totals) are counted as data element types.

8.3.e If logical files must be referenced for an external output, the data element types appearing and referenced there are not counted. Only data element types that cross the boundary of the application are included to determine the complexity.

8.3.f Standard data such as system date and page number are not counted as data element types.

8.3.g Fixed data such as message headers, column descriptions, literals, and constants are also not counted as data element types.

8.3.h If there are data element types for error messages as well as for other messages, or if there is a separate window to display messages, then count as indicated in subclause 4.14.

8.3.i Navigation elements are not counted as data element types.

8.3.j If a header record and/or a trailer record with check data appear(s) in a transaction file (e.g. total amount or total number of records), then this data is counted as data element types (if requested by the user).

File types referenced

8.3.k The number of referenced logical files for each external input is determined by establishing the number of referenced logical files for validating the input and/or for producing the output. A referenced file can either be an internal logical file or an external logical file.

8.3.l When an external output is bound inextricably to an external input, count the number of referenced logical files for the data processing as a whole and not just the number referenced for the external output itself. Consider, for example, the external output for making a report about the processing of a number of transactions: this external output is bound inextricably to the external input for processing transactions.

8.3.m The FPA tables ILF and FPA tables ELF are not counted as a referenced logical file when the complexity of external outputs is being determined.

8.3.n Files such as temporary files, sort files, and print files introduced for technological reasons are not counted. Determine whether these kinds of files are an alternative for an internal logical file or an external logical file. If such is the case, count these underlying logical files to determine the complexity.

Complexity matrix

The following table is used to determine the complexity of an external output:

Table 8 — Complexity matrix external output

File Types Referenced	Data Element Types		
	1 - 5	**6 - 19**	**20 or more**
0 - 1	Low	Low	Average
2 - 3	Low	Average	High
4 or more	Average	High	High

The availability of data regarding the number of data element types and referenced logical files depends on the phase of the application's life cycle. If this data is not known, or when you perform a high level analysis, an identified external output should be valued as *average*.

9 External Inquiries

This clause further examines functional user requirements pertaining to output that an application produces. This output has been fully determined in size beforehand and does not require further data processing. The clause shows which external inquiries must be identified in FPA. The degree to which external inquiries contribute to the functional size depends on their quantity and their complexity.

9.1 Definition of an external inquiry

An external inquiry (EQ) is a *unique* input/output combination recognized by the user in which the application distributes an output fully determined in size without *further data processing*, as a result of the input. It is a function that the user must see as an *elementary process*.

An external inquiry must be considered *unique* if:

— the input part of the external inquiry consists of a different set of data element types than the input part of all other external inquiries, or

— the output product produced by the external inquiry consists of a different set of data element types than the output part of all other external inquiries, or

— the set of data element types of both the input part and the output product is the same, but the external inquiry requires *a different logical way of processing*

A *logical way of processing* is:

a method specified by the user in order to come to a desired result. This is understood to be the following within the context of an external inquiry:

— Referencing logical files

For example: When displaying employee data, the logical file "Employee" is referenced.

— Validations

For example: When employee information is being retrieved, the application checks whether the user has been authorized to query this information.

A logical way of processing is considered *different* when:

— other logical files are referenced

— the same logical files are referenced, but there are other validations

Different technological solutions chosen in order to realize the same logical processing do not mean that the logical way of processing is different.

The data of the input part of an external inquiry must identify the desired output uniquely. Additionally, the size of the output must be fully determined in size.

***Further data processing* is**

in this context, the execution of algorithms or calculations made on data that have been retrieved from logical files before information is shown.

A function is an *elementary process* when two criteria are satisfied:

— The function has an autonomous meaning to the user and fully executes one complete processing of information. In other words, it is self-contained.

For example: Consider a case in which data of a product is inquired about on the basis of its product number. The entering of the product number and the displaying of its corresponding product-data is one complete processing from the perspective of the user.

— After the function has been executed, the application is in a consistent state.

For example: The execution of only one of the two steps above would leave the application in an *in*consistent state, seen from the user's perspective.

9.2 Identifying external inquiries

9.2.a Count each combination of input and output data as an external inquiry when the input of data leads to the direct generation of output and the input-output combination:

- is an elementary process *and*
- specified by the user *and*
- is unique in the application to be measured *and*
- crosses the boundary of the application *and*
- distributes an output fully determined in size without further data processing (see the definition in subclause 9.1).

9.2.b External inquiries can pertain to inquiries originating directly from the user or from other applications.

9.2.c The distinction between an external inquiry and an external input can be explained in more depth by the following: the input part of an external inquiry coordinates a search action only and does not change the internal logical files whatsoever.

9.2.d Do not confuse a query facility with an external inquiry. An external inquiry is a direct search action for specific data in which a single key is used generally. A query facility on the other hand is an organized structure of external inputs, outputs, and inquiries used in order to be able to formulate several inquiries with many keys and operations. FPA considers such an organized structure as an application that must be counted as such when it must be developed separately. In such a situation, therefore, external inputs, outputs, and inquiries, have to be counted in order to measure the query facility (see subclause 4.11).

9.2.e An external inquiry is counted as an external inquiry only when the user has specified it as such. Therefore, displaying the data before editing the data is not counted as an external inquiry.

9.2.f The input part of an external inquiry shall not be counted as an external input.

9.2.g The output part of an external inquiry shall not be counted as an external output.

9.2.h An external inquiry must include the entering of data in order to control data processing (e.g. the entering of selection criteria). By definition, uniquely identifying data must always make up a part of the data entered.

9.2.i Use the features below to distinguish between an external inquiry and an external output:

- the size of an external inquiry's output shall be *completely determined and*
- the input of an external inquiry shall consist of a search argument that is unique in its identification *and*
- the output of an external inquiry shall *not* contain any data that has come about as a result of *further data processing and*
- changes to internal logical files shall not occur when an external inquiry is executed.

See subclause 9.1 for more about *further data processing*.

9.2.j Do not count any additional functions or data element types for being able to browse or scroll through produced output (see subclause 4.17 for a further explanation).

9.3 Determining the complexity of external inquiries

Data element types

The following guidelines for counting data element types are a concrete expression of the generic rule as stated in Clause 4.23.

9.3.a All data element types (not their possible values) that appear in the output product generated by the external inquiry are counted.

9.3.b All control information (e.g. selection criterion, desired medium, desired printer, or time period of printing) at the level of data element type that must be entered to manage the output are counted as data element types for the external inquiry. Control information appearing on the output itself is counted twice.

9.3.c All address data at the level of data element type that indicates for whom or for which device or medium the output is meant is counted.

9.3.d Count all data element types (data and/or control information) that cross the boundary of the application to be counted. Take all data element types into account that are relevant to the input or the output part.

9.3.e If logical files must be referenced for an external inquiry, the data element types appearing and referenced there are not counted. Only data element types that cross the boundary of the application are included to determine the complexity.

9.3.f Standard data such as system date and page number are not counted as data element types.

9.3.g Fixed data, such as message headers, column descriptions, literals and constants are not counted as data element types.

9.3.h If there are data element types for error messages as well as for other messages, or if there is a separate window to display messages, then count as indicated in subclause 4.14.

9.3.i Navigation elements are not counted as data element types.

9.3.j Count the way to get to a function and the way to start it (e.g. menu selection and function keys) as one data element type, the initiation trigger.

File types referenced

9.3.k The number of referenced logical files for each external inquiry is determined by establishing the number of referenced logical files involved in the validation of the input and/or in the execution of the external inquiry. The files can either be internal logical files or external logical files.

9.3.l The FPA tables ILF and the FPA tables ELF are not counted as referenced logical files when the complexity of external inquiries is being determined.

9.3.m Files such as temporary files, sort files, and print files introduced for technological reasons are not counted (see subclause 5.2.f). Determine whether these kinds of files are an alternative for an internal logical file or an external logical file. If such is the case, count these underlying logical files to determine the complexity.

Complexity matrices

The following table is used to determine the complexity of an external inquiry: