INTERNATIONAL STANDARD

ISO/IEC 20968

First edition
2002-12-01

Software engineering — Mk II Function Point Analysis — Counting Practices Manual

Génie logiciel — Analyse des points fonctionnels Mk II — Manuel des pratiques de comptage

Reference number
ISO/IEC 20968:2002(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Table of Contents

TABLE OF CONTENTS III

FOREWORD V

1 1

INTRODUCTION 1

1.1 Definition and Purpose of MkII Function Point Analysis 1
1.2 Purpose of the Counting Practices Manual (‘CPM’) 2
1.3 Who should read this document ? 2
1.4 Albrecht/IFPUG Function Point Analysis 2
1.5 Applicability of Mk II FPA 3
1.6 Manual Structure 4
1.7 Metrics Practices Committee 5
1.8 Procedure for raising a Query or Issue with the MPC 5

2 7

THE MK II FUNCTION POINT ANALYSIS RULES 7

Rule 1 Boundary 7
Rule 2 Functional Size and Logical Transactions 7
Rule 3 Processing Component of Logical Transactions 8
Rule 4 Input and Output Components of Logical Transactions 8
Rule 5 Logical Transaction Size 8
Rule 6 Reporting a MkII Function Point Count 9

3 11

MEASUREMENT STEPS 11

Step 1 Determine the Viewpoint, Purpose and Type of the Count 12
Step 2 Define the Boundary of the Count 12
Step 3 Identify the Logical Transactions 13
Step 4 Identify and Categorise the Data Entity Types 13
Step 5 Count the Input Data Element Types, the Data Entity Types Referenced, and the Output Data Element Types. 13
Step 6 Calculate the Functional Size 13
Step 7 Determine Project Effort 13
Step 8 Calculate Productivity and other Performance Parameters 13
Step 9 Score the Degrees of Influence 13
Step 10 Calculate the Technical Complexity Adjustment 13
Step 11 Calculate Adjusted Size and Performance Parameters 14

4 15

GENERAL GUIDELINES FOR MKII FUNCTION POINT COUNTING 15

4.1 Determining the Viewpoint, Purpose and Type of the Count 15
4.2 Drawing the Boundary for a Count 16
4.3 Interfaces 18
4.4 Identifying Logical Transactions 21
4.5 Identifying Entity Types 39
4.6 Identifying Input and Output Data Element Types 43

5 49

MEASUREMENT GUIDELINES FOR SPECIFIC SITUATIONS 49

5.1 Counting Graphical User Interfaces (GUIs) 49
5.2 Approximate Sizing of Application Portfolios 54
5.3 Sizing Changes 55
5.4 Changes to make software Year 2000 compliant 58
5.5 Counting Application Packages 58

6 61

CALCULATING THE ADJUSTED SIZE (OPTIONAL) 61

7 63

MEASURING EFFORT 63

7.1 Project Start 64
7.2 Project End 64
7.3 Whose time included? 64
7.4 What time is included? 64
7.5 Project duration 64

8 65

MEASURING PRODUCTIVITY AND OTHER ASPECTS OF PERFORMANCE 65

8.1 Development Productivity 65
8.2 Change Productivity 65
8.3 Maintenance and Support Productivity 65
8.4 Measuring and Understanding Performance in Software Activities: The Wider Issues 66

9 68

ESTIMATING EFFORT USING MKII FPA 68

10 70

GLOSSARY OF MKII FPA TERMS 70

APPENDIX I 74

TECHNICAL COMPLEXITY ADJUSTMENT 74

APPENDIX II 84

DATA COLLECTION FORMS 84

Introduction 84
Record Sheets - a Possible Structure 84
Documentation Process 85

APPENDIX III 92

BIBLIOGRAPHY 92

The International Standard: 92
General texts on software measurement with MkII FPA: 92
Use of MkII FPA in Estimating 92
Other Relevant Publications 92

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

ISO/IEC 20968 was prepared by the United Kingdom Software Metrics Association (UKSMA) 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.

Software engineering — Mk II Function Point Analysis — Counting Practices Manual

1

Introduction

1.1 Definition and Purpose of MkII Function Point Analysis

For the purposes of this document, the abbreviation ‘Mk II FPA’ is used for ‘Mark II Function Point Analysis’.

Mk II FPA is a method for the quantitative analysis and measurement of information processing applications. It quantifies the information processing requirements specified by the user to provide a figure that expresses a size of the resulting software product. This size is suitable for the purposes of performance measurement and estimating in relation to the activity associated with the software product

In the context of Mk II FPA, ‘information processing requirements’ means the set of functions required by the commissioning user of the application software product (excluding any technical and quality requirements). ‘The activity’ could be the development, enhancement or maintenance of the software product needed to meet the requirements.

The MkII FPA method is intended to comply with ISO/IEC 14143-1: 1998, the International Standard for Functional Size Measurement (see Bibliography).

1.2 Purpose of the Counting Practices Manual ('CPM')

The Mk II Method of Function Point Analysis was defined by Charles Symons in "Software Sizing and Estimating: Mk II FPA" published in 1991. After development within KPMG in 1985/86, with the protected status of a proprietary method, the method is now in the public domain. The Metrics Practices Committee (MPC) of the UK Software Metrics Association is now the design authority for the method and is responsible for its continuing development.

The purpose of this Manual is to explain and promulgate the method, and to set out the rules for applying Mk II Function Point Analysis ('FPA'). Chapters 1 to 5 inclusive of this manual provide the authoritative standard of the Mk II FPA Method.

It is not the purpose of this manual to replace Charles Symons' or the other books in the Bibliography. The manual is not intended to provide a teaching introduction to MkII FPA, nor does it discuss the broader subject of software measurement.

This manual replaces all previous versions of the Counting Practices Manual.

The definition covers:

- The software domains for which MkII FPA may be applicable
- Application software requirement components recognised by Mk II FPA
- A process for applying the MkII FPA rules and documenting the result
- Interpretation of the rules for a variety of software technologies (e.g., GUI, client/server, objects, etc.)
- Basic Formulae used in Mk II FPA
- Terminology used in Mk II FPA.

An important aspect of this new version of the standard, in order to comply with ISO/IEC 14143-1:1998, is that the Technical Complexity Adjustment is no longer considered to contribute to the "Functional Size". Hence measurements previously expressed in MkII 'Unadjusted Function Points' should now be regarded as the MkII measures of the Functional Size, without further qualification. For the time being, the Technical Complexity Adjustment remains part of the method. If it has been applied then the result should be qualified as the "Adjusted Size".

1.3 Who should read this document ?

- Users of Mk II FPA
- Suppliers of tools, training or other services involving the method.
- Anyone interested in learning about the details of Mk II FPA

1.4 Albrecht/IFPUG Function Point Analysis

Allan Albrecht developed the original Function Point Analysis method. The design authority for the direct descendent of his approach is now the International Function Point Users Group ('IFPUG'). The briefest reference is made here to the relationship between the Mk II and the IFPUG FPA methods.

The two methods measure subtly but significantly different sizes of a software product (and therefore of the work-output of the processes of developing, maintaining and enhancing a software product).

In terms of the sizes produced, the major differences are that Mk II FPA, with its finer granularity, is a continuous measure whereas IFPUG limits component size once a threshold is reached and that the MkII method aims to better reflect the internal processing complexity of business 'data-rich' systems. As the concepts on which the size measure is based are logical transactions and entities, in which software requirements and functional specifications are typically expressed, a MkII Functional Size measure should be truly independent of the technology or methods used to develop or implement the software.

The weightings introduced by Charles Symons were designed to deliver a size scale of similar magnitude for the MkII method as for the IFPUG method. On average therefore, the methods give roughly the same software sizes up to around 400 function points (though there can be quite a scatter about the average for individual items of software). For larger sizes, Mk II FPA tends to produce increasingly higher sizes than the Albrecht/IFPUG method.

For some purposes, e.g. portfolio management, the methods may be regarded as equivalent. However, for the commonest purposes of performance measurement and estimating it is preferable to use one scale or the other consistently, only converting between them if needed, using a formula which shows the average relationship.

1.5 Applicability of Mk II FPA

MkII FPA is a method that assists in measuring process efficiency and managing costs for application software development, enhancement or maintenance activities. It measures a software product size independent of technical characteristics of the software, in terms relevant to users. It can be:

- applied early in the software development process
- applied uniformly throughout the software's lifetime
- interpreted in business terms, and
- understood by users of the software.

MkII Function Points can be used to measure the functional size of any software application that can be described in terms of logical transactions, each comprising an input, process and output component. The sizing rules were designed to apply to application software from the domain of business information systems, where the processing component of each transaction tends to be dominated by considerations of the storage or retrieval of data. The method may be applicable to software from other domains, but the user should note that the sizing rules do not take into account contributions to size such as from complex algorithms as typically found in scientific and engineering software, nor do the rules specifically take into account real-time requirements. To apply MkII FPA to these other domains may be possible or may require extensions to or new interpretations of the rules given in this manual.

MkII FPA can be used for sizing:

- a requirements specification or functional specification of a new application or of a change to an existing application
- the requirements met by an existing, operational application, whether it be a bespoke application or an implementation of a packaged business software solution, and whether a batch or on-line implementation.

Either directly, or coupled with effort, defect counts and other measures, MkII FPA can be used for a variety of purposes, including to:

- measure project or organisational performance (productivity, delivery rate and quality).
- compare internal and external IT performance
- compare application quality and reliability
- compare normalised development, maintenance and support costs of applications on different platforms
- estimate the resourcing requirements, duration and cost of projects
- contribute to the cost and risk elements of the business case for a new project
- assist in identifying all requirements before an application has been developed
- control "creeping elegance" or scope change during projects
- assign work to team members
- determine the size of the application asset base
- produce useful, high-level, functional documentation of old 'legacy' systems that lack up-to-date functional documentation
- determine the replacement value of applications.

MK II FPA is independent of the project management method to be used (e.g. waterfall, spiral, incremental) and of the development method employed (e.g. object-oriented, SSADM, Information Engineering, etc.). It is a measure of the logical, business requirements, independent of how they are implemented.

1.6 Manual Structure

This manual is split into 10 Chapters. After this Introduction, the Chapters address the following:

Chapter 2 describes the Rules to which all Mk II Function Point counts must conform.

Chapter 3 lists the main steps to be carried out in performing a count.

Chapter 4 provides general guidelines and illustrations of how to perform the tasks involved in a count for many situations faced in practice.

Chapter 5 provides measurement guidelines for some sizing specific types of software and for some specific types of sizing requirements.

Chapter 6 describes the process to adjust the Functional Size derived using the MkII method to account for non-functional requirements.

Chapter 7 gives a definition of effort for the calculation of productivity.

Chapter 8 describes how to measure productivity.

Chapter 9 gives a brief introduction to the use of MkII FPA for estimating.

Chapter 10 provides a glossary of terms and definitions.

Appendix 1 contains a detailed definition of the Technical Complexity Adjustment.

Appendix 2 provides some forms which may prove helpful in performing a count.

Appendix 3 contains a Bibliography of publications referred to in this document, and other useful references.

1.7 Metrics Practices Committee

The Metrics Practices Committee was established by the Committee of UKSMA to exercise delegated authority on a day-to-day basis over the Mk II Function Point method. The Committee acts on behalf of UKSMA and the user community with the following objectives:

- to act as the Design Authority and owner of the MkII method
- to maintain control over the rules, interpretation and documentation of the MkII method
- to ensure that the rules are interpreted in a consistent and valid way for both MkII and IFPUG methods
- to supply advice on the interpretation of both the MkII and IFPUG methods
- to foster consistency in the application of the methods, and thus comparability of results
- to promote the use of the Functional Size as a key component of software metrics
- to provide a reference point for interpretation
- to keep the user community informed of any developments via the UKSMA Committee and general meetings.

1.8 Procedure for raising a Query or Issue with the MPC

Should the reader wish to comment or need advice on any aspect of this manual, please contact the UKSMA Administrator. Suggestions for improvement to this document, and UKSMA services are warmly welcomed!

(Blank page)

2

The Mk II Function Point Analysis Rules

Below are listed the rules to which all Mk II Function Point Counts must conform. General guidelines to assist in conformance to these rules are given in Chapter 4 of the manual.

Rule 1 Boundary

1.1 Mk II FPA is used to measure the size of the functionality required by the users of an application, within a boundary defined for the purpose of the FP count.

1.2 The application or part of the application enclosed by the boundary must be a coherent body of functionality, comprising one or more complete Logical Transaction Types. (In the following, 'Type' is dropped for ease of reading.)

Rule 2 Functional Size and Logical Transactions

2.1 The Functional Size of an application is the sum of the sizes of each of the Logical Transactions whose input and output components cross the enclosing boundary.

2.2 A Logical Transaction is the lowest level of self-consistent process. It consists of three components; input across an application boundary, processing involving stored data within the boundary, and output back across the boundary.

2.3 It is triggered by a unique event that is of interest to the user, which, when completed, leaves the application in a self-consistent state in relation to the unique event. It may also be triggered by a user requirement to extract information from the application, leaving it unchanged.

2.4 A Logical Transaction is counted once when sizing an application, even though it may be executed from more than one point in the application.

2.5 Changes to are counted by summing the size of the added, changed, and deleted Logical Transactions.

2.6 In the case of changed Logical Transactions, the size counted is the size of the changes made to the Logical Transactions, not the overall size of the Logical Transactions. The size of the changes made is found by summing the counts of the added, changed and deleted input, process and output components of the Logical Transactions.

Rule 3 Processing Component of Logical Transactions

3.1 The processing component of a Logical Transaction is analysed by reference to its manipulation (i.e. create, update, delete, or read) of stored data.

3.2 The processing component is sized by counting the number of Primary Entity Types that are referenced by the Logical Transaction, plus the System Entity if referenced. See Section 4.4 for definitions of 'Entity-Type' and 'Primary'.

3.3 All accesses to Non-Primary Entity Types within the entity model are considered to be references to a single entity called the System Entity.

3.4 Within an application boundary there can be only one System Entity, to which a maximum of one reference may be included in the processing component of a Logical Transaction.

Rule 4 Input and Output Components of Logical Transactions

4.1 The input and output components of a Logical Transaction are sized by counting the number of Data Element Types crossing the application boundary, via each component respectively.

4.2 A Data Element Type is a uniquely processed item of information that is indivisible for the purposes of the Logical Transaction being sized. It is part of an input or output data flow across an application boundary.

Rule 5 Logical Transaction Size

5.1 The Functional Size of a Logical Transaction is the weighted sum of the input, processing, and output components of the Logical Transaction.

5.2 The industry standard weights are as follows: Input Weight is 0.58 (per Input Data Element Type), Processing Weight is 1.66 (per Entity Type Reference), and the Output Weight is 0.26 (per Output Data Element Type).

Rule 6 Reporting a MkII Function Point Count

6.1 A MkII FP count obtained according to the rules of this Counting Practices Manual should include reference to the CPM Version Number when it is reported, for example:

‘557 MkII Function Points (ISO/IEC 20968:2002)’

(Blank page)

3

Measurement Steps

The purpose of this section is to summarise the key factors in each of the steps 1 to 6 which comprise the Mk II FPA Counting Process, the additional steps 7 to 8 which are needed to determine the most commonly required performance parameters, and the optional steps 9 to 11 which are needed to take into account the Technical Complexity Factor. Reference is made to the Rules of Chapter 2, and to other parts of this manual which provide greater detail and guidance.

The Functional Size of an application is derived from the Functional User Requirements as expressed by the commissioning user.

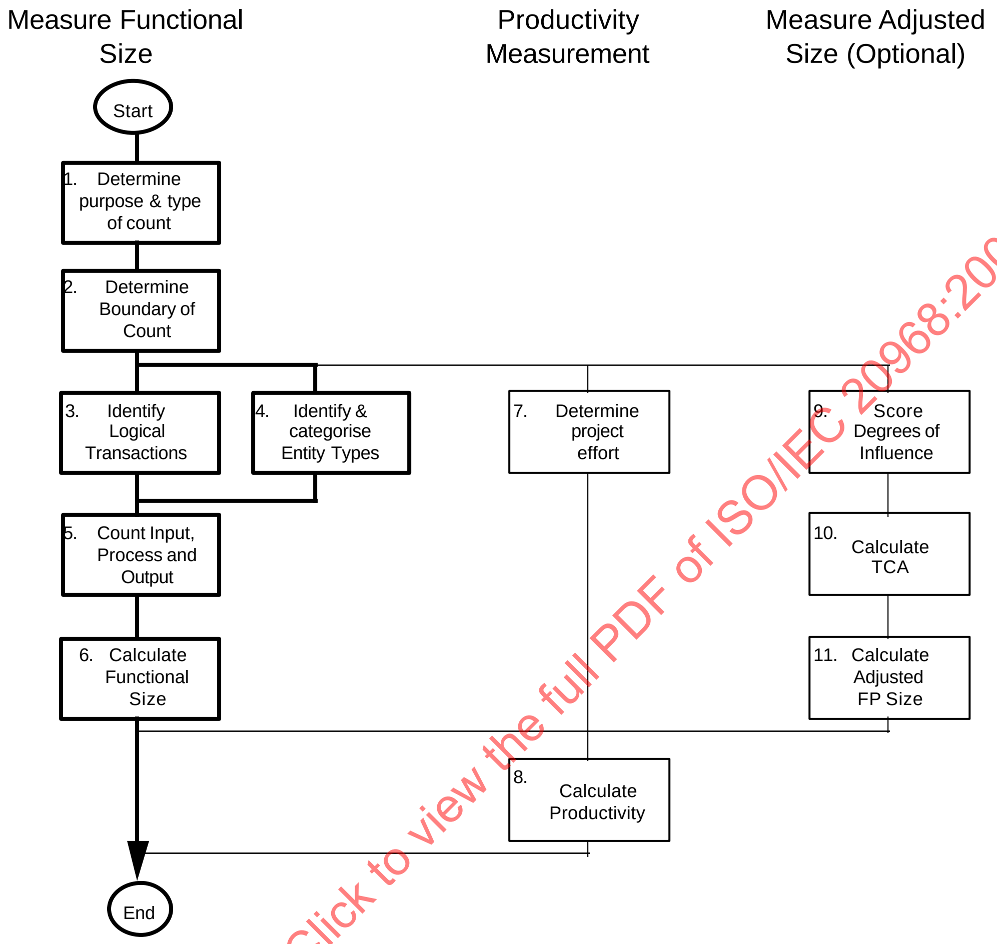
Any function that is provided, but which was not required by the user, is ignored for FPA purposes (Rule 1).

Functional requirements are usually modelled in the Requirements Specification and Logical System Design documents. They are an expression of what an application should do or does, rather than *how* it does it, and these are therefore good source documents for counting purposes.

Access to a member of the project team knowledgeable in what the system is supposed to do, and to a data analyst for the data entity model, is also an invaluable asset to a count.

The effort involved in a count typically takes 0.2% - 0.4% of the total project effort for a new large development project.

Mk II FPA Counting Process and its use for Productivity Measurement


Measure Functional Size
Productivity Measurement
Measure Adjusted Size (Optional)
Start
1. Determine purpose & type of count
2. Determine Boundary of Count
3. Identify Logical Transactions
4. Identify & categorise Entity Types
5. Count Input, Process and Output
6. Calculate Functional Size
7. Determine project effort
8. Calculate Productivity
9. Score Degrees of Influence
10. Calculate TCA
11. Calculate Adjusted FP Size
End

Step 1 Determine the Viewpoint, Purpose and Type of the Count

Identify the customer for the count, and the purpose. For example, is it to measure the work-output of a particular group of developers, or the functionality 'owned' by a particular user? Is the aim to count all of the functionality which was required, or the functionality which was delivered to the user? See Section 4.1 for guidelines.

Questions which may help determine what has to be counted include:

- Does the project involve development, enhancement, maintenance, or support?
- When did/does the project begin and end?

Is an accurate or approximate count needed? (If the latter, see Section 5.2.) Is it needed for performance measurement, or for estimating? (If the latter, see also Chapter 9.)

Step 2 Define the Boundary of the Count

This may be iterative with Step 1. Drawing the boundary determines the Logical Transactions to be included (the 'Scope') and which are to be excluded from the measurement.

See Sections 4.2 and 4.3 for guidelines.

Step 3 Identify the Logical Transactions

Logical transactions are the lowest level processes supported by the application, consistent with Rule 2

See Section 4.4 for guidelines.

Step 4 Identify and Categorise the Data Entity Types

It is usually highly desirable to have an entity-relationship data model for the requirements, to identify all the Data Entity Types. However, as only the Primary Entity Types are needed, a full Third Normal Form analysis is not needed.

See Section 4.5 for guidelines.

Step 5 Count the Input Data Element Types, the Data Entity Types Referenced, and the Output Data Element Types.

For each Logical Transaction, count the number of Input Data Element Types (Ni), the number of Data Entity Types Referenced (Ne), and the number of Output Data Element Types (No).

See Sections 4.5 and 4.6 for guidelines.

Step 6 Calculate the Functional Size

The Functional Size (FS) is the weighted sum over all Logical Transactions, of the Input Data Element Types (Ni), the Data Entity Types Referenced (Ne), and the Output Data Element Types (No).

So the Functional Size(FS) for an application is:

$$FS = Wi * \Sigma Ni + We * \Sigma Ne + Wo * \Sigma No,$$

where ‘Σ‘ means the sum over all Logical Transactions, and the industry average weights per Input Data Element Type, Data Entity Type Reference and Output Data Element Type are, respectively:

$$Wi = 0.58$$

$$We = 1.66$$

$$Wo = 0.26$$

For the sizing formulae to be used for Changes to applications, see Section 5.3

Step 7 Determine Project Effort

Determine the total effort and elapsed time for the project.

See Chapter 7 for guidance.

Step 8 Calculate Productivity and other Performance Parameters

Examples: Productivity = FS / Project Effort,

Delivery Rate = FS / Elapsed Time

See Chapter 8 for guidance.

Step 9 Score the Degrees of Influence

Optionally assess the Degrees of Influence of each of the Technical Complexity Adjustment characteristics.

See Appendix 1 for guidance.

Step 10 Calculate the Technical Complexity Adjustment

Optionally calculate the TCA.

See Appendix 1 for guidance.

Step 11 Calculate Adjusted Size and Performance Parameters

Optionally use the TCA calculated in Step 10 to calculate the Adjusted Size which can then be used instead of the FS to derive the associated performance parameters (e.g. productivity and delivery rate), as in Step 8.

4

General Guidelines for MkII Function Point Counting

The purpose of this chapter is to provide users of the MkII FP method with more detailed rules and practical cases which illustrate the application of the method for a variety of real counting situations. The structure of the chapter corresponds to the first five steps of the MkII FPA Counting Process described in Chapter 3.

4.1 Determining the Viewpoint, Purpose and Type of the Count

As defined in Rule 1, the first important step before starting any MkII Function Point count is to decide the 'boundary' of the application which has to be counted; this determines what functionality is included and what is excluded.

The choice of the boundary depends on the *viewpoint,* and perhaps the *purpose* of the count. There are three commonly encountered viewpoints.

- The Project Viewpoint. We wish to determine the size of the functionality delivered, i.e. the 'work-output', by a specific application development or enhancement project, or the functionality supported by a specific application maintenance or support activity. The purpose might be that we wish to use the resulting functional size measure as a component of measuring productivity of the project, or as a component of estimating the effort to develop the project. In this case the boundary is drawn around the functionality delivered by the project team or as maintained by the support team.

- The Application Manager Viewpoint. We wish to determine the total size of applications which support a specific business functional area, and which are managed as a unit by an Application Manager. The purpose might be to track the extent of automation of the business area as it progresses each year, or to measure the support productivity of the Application Manager's team for that whole area. In this case the boundary is drawn around the functionality supported by the Application manager's team.
- The Business Enterprise Viewpoint. We wish to determine the total size of the application portfolio of the enterprise for the purpose of asset valuation. In this case the boundary is drawn around the whole of the portfolio of the Enterprise.

The reason for distinguishing these different viewpoints is that the MkII FP counts arising from these different viewpoints are not necessarily related in a simple additive way. Usually, the whole is *less* than the sum of the parts. Examples will illustrate.

- The functional requirements of a Business area are developed in three successive application development projects. For the purposes of estimating and of measuring the productivity of the three separate projects, we need to count the FP's delivered by each of the projects separately. However, because of the architecture of the systems of this Business area in our example, each project had to deliver some temporary, 'throw-away' functionality to convert data from the previous generation of applications, and in addition some permanent interface files have to be created for the three applications to work coherently.

Clearly, to measure the productivity of the individual projects, we need to count the total functionality delivered from the Viewpoint of the Project Leader by each separate project. This would include the FP's of any 'throw-away' functionality, and of any interface files which that project had to provide. But when measuring the final total or 'nett' functionality from the viewpoint of the Business area when all three projects have been completed, we would not count the 'throw-away' functionality, and it *may* arise that not all of the functionality implied in the interfaces is credited to the 'nett' functionality (see further the discussion on interfaces below).

- A similar type of boundary problem arises when counting the FP's delivered for client-server systems. A client-server application could be developed by two sub-project teams, one specialising in PC client applications, and the other in main-frame server applications. The application boundaries for the two projects may well overlap, with some user functionality being dealt with by both projects. If we wish to measure the performance of each development sub-project team separately, then the total functionality delivered by each team would be counted separately. Alternatively, if we wish to size the integrated client-server application, to measure the functionality delivered to the user or project sponsor, then the overlapping functionality would not be double-counted.

4.2 Drawing the Boundary for a Count

As shown in the preceding paragraph, where the boundary of an application is drawn for FP counting depends on the intended viewpoint, purpose and use of the count.

Drawing the boundary determines what functionality is to be included in the count, and what excluded.

When the application boundary is drawn, it defines the conceptual border between the software and the 'user' (see further below). 'Input data' from the user crosses the boundary to enter the application. 'Output data' leaves the application and crosses the boundary to reach the user. Within the boundary are the Data Entity Types (hereinafter referred to as 'entities', for simplicity) that are processed by the Logical Transaction Types (hereinafter referred to as 'logical transactions' or just 'transactions') of the application.

In the context of FPA, examples of 'user' are:

- Business User - e.g. a manager, a clerk, someone at a terminal, or some other person who enters data into the application and receives output.
- Automated User - another application or automatic data collection device that supplies data to or receives data from the application being counted. (Note that an application can also be regarded as the user from the viewpoint of a 'lower-level' piece of software such as a file handler or network access device. But MkII FPA was designed to work at the business application level, and great care should be taken about applying the method at lower, infrastructure software levels.)

In all cases, data from the user which crosses the application boundary to be processed will:

- cause some part of the application to change state according to the requirements of the user.

and/or

- cause data to be made available to the user.

Drawing an application boundary must not result in incomplete logical transactions. Any logical transaction must have some user input from across the boundary, do some processing on data entity types within the boundary, and return output across the boundary.

Whether the requirements of the application are implemented in on-line or in batch mode is immaterial to the FP count; there is no logical difference between data crossing the application boundary on-line, and data crossing the boundary off-line (batch).

Figure 4.1 below shows a typical example of an application boundary from a business user perspective, encompassing on-line and batch transactions.

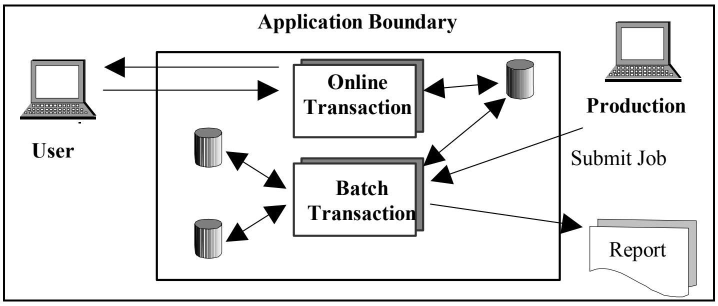


Figure 4.1 - Business user view of application boundary

Referring to figure 4.1, we can see that there is an on-line transaction that updates one database. This transaction involves the user at a terminal holding an input/output dialogue across the application boundary.

In figure 4.1 there is also a batch transaction which accesses three databases. This transaction is initiated in a production environment, under the control of the computer operators, and produces a report, which is sent to the business application user. The input from the operators, who initiate the job, crosses the application boundary, as does the output report that is sent to the user.

There are three databases shown within the application boundary in figure 4.1. It does not matter for the FP count of this application if one or more of these databases happens to be shared with other applications that may create, read, update or delete data on the databases. For the purposes of the function point count being described, the entities implied in all three databases are included within the application boundary as they are referenced by the two transactions shown in figure 4.1, and the relevant entity references will be counted in each of those transactions.

Databases can appear within the boundaries of any number of applications. The criterion for their inclusion is simply whether or not the data is referenced by one or more logical transactions included within the application being counted. Logical transactions, on the other hand, normally appear in only one application, unless functionality is deliberately duplicated across more than one application.

4.3 Interfaces

When an application boundary is drawn for the purposes of a particular FP count, it may result in identifying 'interfaces' to other applications. In common understanding, the word 'interfaces' is used for two physically different processes.

- a file of 'data' (e.g. master data records, 'transactions', a string of messages, etc.) is *made available* as an interface file from one application to be shared with another
- 'data' are *transmitted* via an interface, either one at a time or as a batch, from one application to another

N.B. When we use the term 'transactions' in the above, these are not complete logical transactions in the meaning of MkII FPA. Typically, these 'transactions' might be the input part of a set of logical transactions. For example a file (or stream) of orders, or bank terminal withdrawals, would be the input part of the logical transactions for the receiving application which is going to process them. (Remember, a complete logical transaction always involves an input, process and output component, which leaves the application in a self-consistent state.)

In the first type of interface above where one application shares a file with another, both applications, the one making the interface file available (i.e. the application maintaining the file) and the other using (i.e. reading) the same interface file, encompass this file within their boundaries. Therefore the entity types implied in the file should be counted in the relevant logical transactions of each of the applications.

In the second type of interface above, the data transmitted across the common application boundary form the output part of a logical transaction of the sending application, and similarly also the input part of a logical transaction of the receiving application. As in the first type of interface above, any entity types referenced by the transmitted data should be counted in both the sending and receiving applications, as both process such data.

At first sight, these two types of interfaces seem to discuss different physical implementations of the same logical requirement and thus might be expected to give identical MkII FP counts. But this is not necessarily so – the two types of interfaces may result from distinct logical requirements.

Three cases below will illustrate how to apply the counting rules when interfaces arise. In each case it is assumed that the Viewpoint requires each application processing the interface file to be sized separately. In the table headings, 'DET' means 'Data Element Type'.

Case 1. Orders are entered on-line in Application A and validated, and accumulated on a file which can be accessed directly by, i.e. shared with, Application B. The shared order file is within the boundary of both applications, and hence the orders do not have to 'cross an application boundary' to be passed from A to B. At some arbitrary time chosen by Application B, it processes the accumulated orders against a Stock file and produces a summary report. Application A has no influence on when Application B reads the file. Applications A and B both also read a master Customer file during their respective processing.

Applic.	Transaction	Input DET's	Processing ER's	Output DET's
A	Order Entry	20	2 (Ord, Cust)	1 (Error / Confirm),
B	Order Process	1 (Trigger)	3 (Ord, Cust, Stock)	10 (Ord Summary)

Case 2. As Case 1, but before Application B begins to process the interface file, Application A sends a notification that 'on-line processing has completed'. There is therefore a separate transaction sent by Application A, and received by Application B, that synchronises the two applications. If we are counting both applications separately, this transaction should be counted in both applications.

Applic.	Transaction	Input DET's	Processing ER's	Output DET's
A	Order Entry	20	2 (Ord, Cust)	1 (Error / Confirm)
A	Entry Complete	1	1	1
B	Begin Order Process	1 (Trigger)	3 (Ord, Cust, Stock)	10 (Ord Summary)

(Note: Case 2 gives a different count to Case 1. But in Case 1, Application A has no control regarding which processing cycle of Application B will handle any specific Order. There is also in Case 1, arguably, a requirement for a 'transaction' to Application B informing the latter that it may start its processing, but this originates from somewhere other than Application A and has not been shown as required in Case 1. It may, for instance, be a timed trigger, such as 'Time_Of_Day', or an operator intervention.

Case 3. As Case 1, but each order, immediately after entry and validation, is transmitted 'on-line' by Application A to Application B.

Applic.	Transaction	Input DET's	Processing ER's	Output DET's
A	Order Entry	20	2 (Ord, Cust)	1 (Error, etc) plus 'n' *
B	Order Process	5 (revalidated)	3 (Ord, Cust, Stock)	10 (Ord Summary)

* 'n', the count of the number of DET's transmitted out across the common boundary depends on the requirements for formatting agreed with Application B. If the 20 DET's transmitted out are treated as a block, unchanged in format from the data entered, count "one". If all 20 DET's are re-formatted to meet the requirements of Application B, count 20.

Case 3 also differs from Cases 1 & 2 in that each Order is processed discretely and sequentially by both Applications A & B. No additional notification messages are needed, as, upon receipt of each Order transaction, Application B processes it and waits for Application A to transmit the next.

In all of Cases 1, 2 & 3, Applications A & B may reside on the same or different processors. This is a physical implementation detail which does not affect the functional size.

The General Rules therefore are as follows:

(a) Input read from an Interface File: If information read from an interface file requires validation, then count the input DET's that require validation as part of a logical transaction of the receiving application.

(b) Output transmitted across an Interface: Count the DET's made available or transmitted across the interface that have to be specially formatted for the receiving application in the output component of a logical transaction of the transmitting application

(c) The files made available or transmitted across an interface comprise DET's which will may create or cause updates of one or more entities. Count any references to these entities in the relevant logical transactions of both the sending and receiving applications.

(d) Take care to understand any logical requirements to synchronise interacting applications and count the transactions made necessary.

(Note also that if the task were to produce the size of the combined system A+B as one unit, ignoring interface considerations, then the relevant transactions and their sizes would be as in Case 1 above, irrespective of how the requirements had been implemented.)

4.4 Identifying Logical Transactions

After deciding the purpose of an FP study, and choosing an appropriate boundary, the next most critical decision to make when performing FPA, is to correctly identify the set of logical transactions involved. This Section is intended to help the reader identify logical transactions correctly and clarifies some commonly encountered counting situations.

N.B. Tests of experienced MkII FPA practitioners have shown that the commonest mistake made is to *omit* to identify and hence size some of the Logical Transactions of the system. The advice of this Section on distinguishing Logical Transactions is therefore especially important.

Users of MkII FPA should note that early in a project's life-cycle, requirements are often stated in a general form such as 'the system shall be usable by anyone familiar with GUI conventions'. The temptation at this stage is to say that such a requirement can only be handled by the Technical Complexity Adjustment (which is no longer considered to contribute to the MkII FP size).

By the time the requirements are agreed in detail, however, much of this 'ease of use' requirement will have influenced the information processing content of the various Logical Transactions, and hence this requirement will be taken into account in the MkII Functional Size. The Logical Transactions to be counted thus correspond to the *finally agreed set of logical requirements allocated to the software being sized.*

4.4.1 What is a Logical Transaction ?

Logical transactions are the lowest level business processes supported by a software application. Each comprises three elements: input across an application boundary, some related processing, and output across the application boundary.

The definition says that:

> ***Each logical transaction is triggered by a unique event of interest in the external world, or a request for information and, when wholly complete, leaves the application in a self consistent state in relation to the unique event.***

This statement warrants some further explanation. The keywords are examined below.

triggered	Each logical transaction has an input part, some related processing and an output part – the processing and output occur in response to the stimulus provided by the input part

which itself corresponds to an event in the external world.

unique

In FPA, we are concerned only with unique types of event – synonyms and multiple occurrences are ignored, provided that the same set of processing is triggered in each case. That is, the same acquisition and validation, references to the same set of entity types, and the same formatting and presentation are performed. Variations in processing path resulting from different input values on distinct occasions are not regarded as giving rise to different logical transaction types. All the entities which could be referenced by all possible processing paths should be counted in the one logical transaction type.

event of interest

People are interested in information of varying kinds; such information originates when things happen in the 'real world' e.g. when an account is opened at a bank, or a lift reaches a specific floor, when a customer pays for goods bought at a retail store, or a television programme is cued to commence broadcasting. The 'events of interest' are those about which the software application is required to process information.

Events are:

- Atomic; events are indivisible and cannot be decomposed into two or more component events unless the level of abstraction (granularity) of the description you are making, and hence the purpose and requirement of the application, is changed;
- Consistency preserving and Instantaneous ; each event either completes successfully, or fails entirely – it is not possible for an event to have 'partially happened';
- Detectable; in order to store information about an event, the event must be observable, or at least able to be inferred from observations.

These are the ACID tests – if any candidate 'event' fails one or more of these criteria, reject it from the application's logical model.

In some circumstances we can contrive that the events of interest in the external world directly generate messages which form the input side of a logical transaction. In other circumstances, a software application must periodically inspect (i.e. 'poll') the status of the external world to determine whether an event of interest has occurred, generating an input message to document a positive result. In either case, the resulting message is regarded as the input side of a logical transaction (the detection mechanism is purely implementation detail). The polling mechanism is, of course, triggered by an event in the external world i.e. the passage of a specified amount of time.

external world

Everything that lies outwith an application's boundary is considered to be the external world of that application. All users of an application exist in this external world, as do the happenings that are recorded in the form of events, and the requests for information made by the application users.

Thankfully, these are the only components of the external world that need any description for the purposes of FPA. However, such a description is absolutely necessary in order to establish the context of the application.

request for information

People wish to store and remember information about events, as indicated above. In order to retrieve some of the stored information from an application, a request is made (by a user in the external world). This takes the form of an input message, which triggers some processing and a response (output), just as does an event. Individual requests are always atomic, consistency-preserving (as, by definition, they cause no change of state), instantaneous and detectable. Hence, requests for information (often called 'queries') provide an essential source of logical transactions. They differ from events in that events occur regardless of the existence of a software application; requests for information are themselves consequences of the existence of the application.

wholly completed

The definition of a logical transaction does not specify the time between receipt of the input message and the end of the output response; the duration may be a few microseconds or run into years (in theory). However, the logical transaction commences when the event is detected and is wholly complete only when the response has left the application boundary.

self-consistent state as regards the unique event

As indicated above, events are defined as instantaneous, and logical transactions are defined to record the fact that an event has occurred. As an event cannot be decomposed, nor partially happen, neither may the logical transaction that represents it. A transaction must be entirely successful or fail entirely if the information stored in the application is to remain correct, that is, self-consistent.

No single item of information stored by an application should contradict another item of information – if such a situation should occur as the result of a software fault, the application is said to have 'lost integrity'. (But this cannot normally be detected by FPA!)

4.4.2 CRUDL – Create, Read, Update, Delete, List

In an application that stores information about a set of entity types (i.e. a set of business objects), the presence of each entity type automatically implies the existence of a minimum set of logical transactions. This minimum set is colloquially referred to using the acronym 'CRUDL', which stands for 'Create, Read, Update, Delete and List'.

Instances of any entity type must follow a minimalist life history initiated by a create event, then consisting of a sequence of zero or more update events and terminating with a logical delete event. See Figure 4.2. At any time during this life history, application users may make requests for information, i.e. Reads, to view the details of one specific instance of an entity type, or require a List of all or of a selection of stored entity instances. During a first attempt to catalogue the logical transactions in an application, the FPA practitioner is advised to assume the presence of at least these five CRUDL transaction types - although not all types may be required by the same application.

Subsequent analysis is likely to expose more detailed information regarding the life history of the entity types. In particular, the practitioner should expect to find that a model using a single Update event is too simplistic. Most business entity types exhibit richer life histories, where each entity instance passes through a variety of states, undergoing a specific set of changes each represented by distinct logical transaction.

Note that a transaction termed, for example 'Maintain Customer Details', is almost certainly NOT the lowest level business process.

'Maintain' implies requirements for a series of logical transactions such as 'View' an existing item, if <not found> 'Add' a new item, 'Change' the viewed item, or 'Delete' it, 'Authorise' the change made by a junior member of staff, or 'List' the entire set of items available, etc.

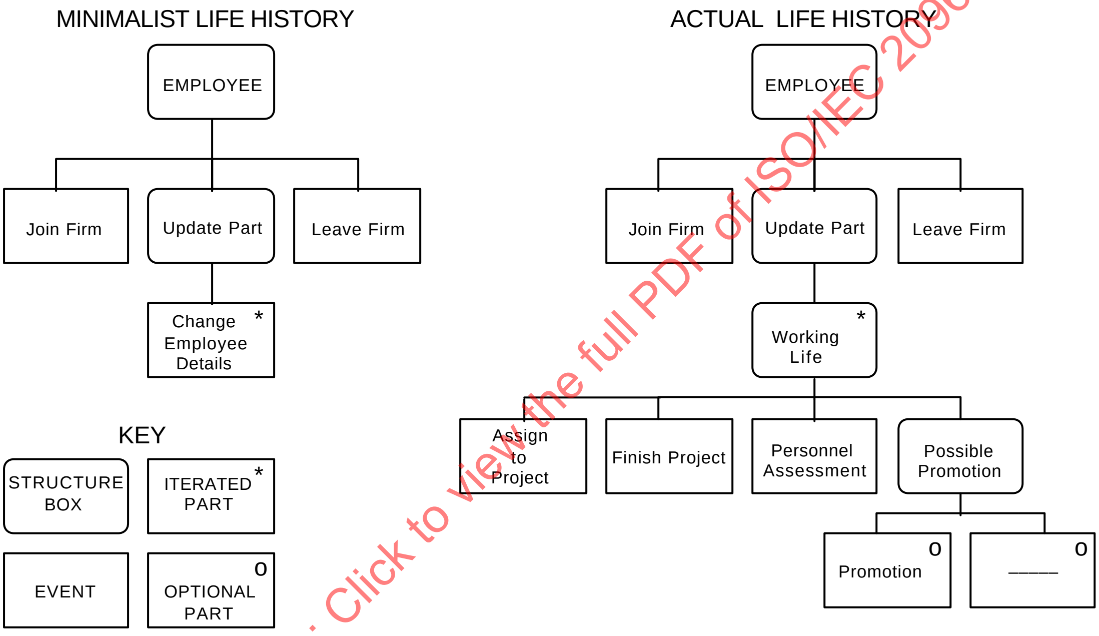


Figure 4.2: Minimalist & More Complex Entity Life Histories

Ask the following questions when identifying the set of logical transactions:

- 'Who uses it?'
- 'Why is it needed?'
- 'What does it do?'
- 'When is it initiated?'
- 'How is it different from other logical transactions?'

4.4.3 Cataloguing Logical Transactions

One approach that is recommended when analysing and defining software requirements is to produce successively refined versions of a catalogue of the logical transactions that are (or are required to be) fulfilled by the application (or project).

During the early stages of problem analysis, the transaction catalogue will consist of a simple list of all the candidate events and requests for information (where the term 'candidate' implies that a definite commitment has yet to be made regarding inclusion). See Figure 4.3. While recording candidate transactions, if other information such as for example, the required transaction response time, the transaction volumes or security criteria, becomes available, it may be helpful to record this too in the catalogue. This will aid calculation of the Technical Complexity Adjustment if required, and helps document both functional and related qualitative requirements.

ID	Transaction Name	Event or Query		Response Time	Volume
Transaction Cluster #1 – MAINTAIN CUSTOMER					
T001	Add_Customer	E		< 3 sec	120 / day
T002	Change_Customer	E		< 3 sec	15 / day
T003	Delete_Customer	E		< 3 sec	15 /day
T004	Query_Customer	Q		< 3 sec	50 / day
T005	List_Customers	Q		< 60 min	< 5 / day
Transaction Cluster #2 – MAINTAIN ACCOUNT					
T010	Add_Account	E		< 3 sec	200 / day
	etc.				

Figure 4.3: Catalogue of Clustered Candidate Logical Transactions

As work progresses, information becomes less ambiguous and requirements are agreed more clearly. Details can be added progressively to describe the input, process and output parts of each transaction. Decisions can be made regarding the confirmation or rejection of the candidate status of transactions, and new transactions may be found.

By the time software requirements are completely described in detail, the transaction catalogue will contain all the information required to perform an accurate and precise MkII FPA measurement. It can also provide the information necessary to control scope creep and to perform impact analysis when subsequent changes are proposed. For an example of a Catalogue of Transactions complete with the sample details, see Figure 4.4. (Again, response time and volumes are possibly only relevant if the TCA is to be calculated.)

ID	Transaction Name	Event or Query	No. of Input DET	Response	No. of Output DET	Entity Types Referred to	No. of ER	MkII FP	Response Time	Volume
Transaction Cluster #1 – MAINTAIN CUSTOMER										
T001	Add_Customer	E	20	OK/Error	1	Customer	1	13.52	< 3 sec	120 / day
T002	Change_Customer	E	20	OK/Error	1	Customer	1	13.52	< 3 sec	15 / day
T003	Delete_Customer	E	1	OK/Error	1	Customer	1	2.5	< 3 sec	15 /day
T004	Query_Customer	Q	4	Customer & Account_Details	25	Customer Account	2	12.14	< 3 sec	50 / day
T005	List_Customers	Q	1	Customer_List	19	Customer	1	7.18	< 60 min	< 5 / day
Transaction Cluster #2 – MAINTAIN ACCOUNT										
T010	Add_Account	E	15	OK/Error	2	Customer Account	2	12.54	< 3 sec	200 / day
	etc.									

Figure 4.4: Catalogue of Logical Transactions completed with details

4.4.4 The Three Elements of a Logical Transaction

Every logical transaction consists of the three elements of input, process and output. MkII FPA makes the following basic assumptions regarding the functional size of these three elements:

- the size of the input element is proportional to the number of uniquely processed Data Element Types (‘DET’s’) composing the input side of the transaction
- the size of the processing element is proportional to the number of Data Entity Types (or ‘entities’) referenced during the course of the logical transaction;

- the size of the output element is proportional to the number of uniquely processed DET's composing the output side of the transaction.

It is a convention of MkII FPA that, as a minimum, each logical transaction must have at least one input data element type, must make one reference to an entity type and must have one output data element type.

By examination of the catalogue of the logical transactions that compose a software application (or project), three *base counts* can be derived for the above; i.e. the Total Number of Input DET's, the Total Number of References to Entities and the Total Number of Output DET's. Detailed rules for identifying references to entity types and DET's are described in the next Sections 4.5 and 4.6 respectively.

The three elements input, process, output, each represent information processing of a distinct nature. Specifically:

- the input element consists of the ***acquisition and validation*** of incoming data either describing an event of interest in the external world, or the parameters of a request for information to be output from the application;
- the processing element consists of the ***storage and retrieval*** of information describing the status of entities of interest in the external world;
- the output element consists of ***formatting and presentation*** of information to the external world.

The functionality involved in providing each of these three distinct kinds of information processing is different. Hence, in MkII FPA, three weighting factors are used to enable these three kinds of functionality to be combined into a single value for a Functional Size. (The weights have been calibrated to industry-average relative effort to analyse, design, program, test and implement these components, so that the MkII FP Functional Size scale provides an industry-average relative measure of work-output for these activities.)

4.4.5 Logical Transactions at Application Program Interfaces

As discussed above, defining the application boundary clarifies the interfaces to the world external to the application and to the application users that exist in that external world. When considering the logical input and output of information, the source and destination of specific messages is of little consequence; for our MkII FPA purposes we are interested only in data ***as it crosses the application boundary***. However, some of the external users may be other software applications, and it is worth examining the kinds of situation than can arise.

Application Program Interface with Another Application

Two applications inter-operate such that information from one application is transmitted to the second application. See Figure 4.5.

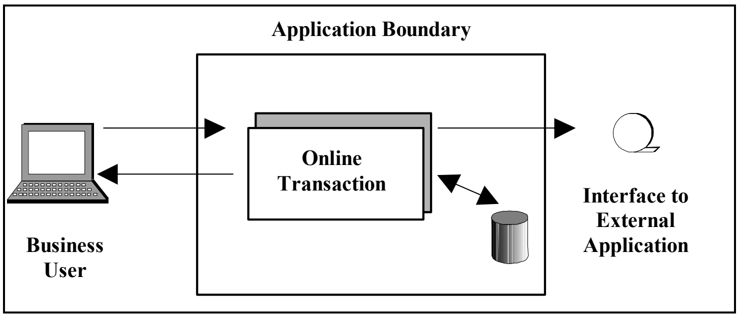


Figure 4.5 - Example of an interface between two applications

In this example, events in the business world trigger a logical transaction that is dealt with by one application, causing changes in the state of the data retained by that application, and a response to the business user. As a result of the transaction, data is written to a file. This file is subsequently read and processed by a second application. This second application is a separately developed, maintained and managed piece of software, having its own distinct application boundary.

Completion of the transaction leaves the first application in a consistent state *regardless of the success or failure of the processing provided by the second application*.

The second application reads the information contained in the file and treats the records it contains as a stream of input messages, each of which forms the input side of a logical transaction. To maximise reliability the receiving application should validate the data content of the file as it would information input from any other source. Hence, provided the structure and content of the interface file is published and adhered to, the internal organisations of the two inter-operating applications are independent one from another. Thus they comply with the software engineering principle of *information hiding* designed to encourage *low coupling* between the software components and hence maximise maintainability and minimise support costs.

For the purposes of MkII FPA, the information written to the interface file by the sending application is seen as output data elements of the logical transaction that triggers the processing. In addition, the receiving application will see the records contained in the interface file as the input side of a series of logical transactions, and one would expect the receiving application to contain corresponding processing and provide a suitable output in response.

Updating Files Nominally ‘Owned By’ Another Application

In a situation somewhat different from that above, the first application might directly create entity types stored as files more usually updated (and hence regarded as being *owned by* the second application).

In this circumstance, no application program interface exists, as the first application is intimately dependent on the internal structure of the second application. The second application would need to perform no validation of input, as all the processing related to the initial stimulus is performed by the first application. Hence, in MkII FPA, under these circumstances the creation of the files would be counted as references to entity types taking place during the processing part of the logical transaction concerned in the first application. Hence these files would be considered as being included within the boundary of the first application.

Confusion may arise here because the concept of the ‘application boundary’ is commonly used in two senses. One sense is the well defined MkII FPA concept of the application boundary. On the other hand, sometimes a set of files and related software programs are allocated as the responsibility of a particular maintenance team and are regarded as an ‘application’. This arises due to a political division of work and responsibility. In such a case, the political boundary of the application may not coincide with the MkII FPA concept of the ‘application boundary’.

User views of the application boundary

An application boundary can be drawn from a number of points of view. As components of transactions can be recognised by examining the inputs and outputs where they cross the boundary, understanding the nature of the boundary has a profound effect on the set of transactions identified. Consider the following client/server application shown in Figure 4.6

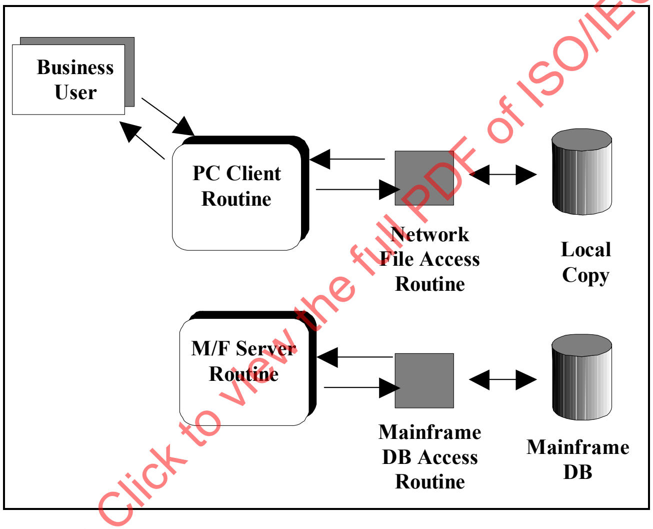


Figure 4.6 - Example of a client/server application

In this example there is a PC that runs a client routine providing a GUI front end to the business user. The routine accesses a network database residing on a local file server. This database contains a summary of commonly used data that resides in a more complete form on a mainframe machine (possibly geographically distant).

When the user enters an enquiry on the PC, the local database on the file server is used to provide information displayed on the screen. However, only summary data is shown. If the user wants more detailed information, double clicking with the mouse causes the detailed data to be extracted from the master database that resides on the mainframe. All updates that the user makes cause the summary database on the network to be updated, as well as the mainframe master.

The two databases are used to improve response times. The network database, holding only summary data, is quite small, and the response time for the majority of transactions, which only require summary information, is very fast. On the few occasions when detailed information is required, the mainframe is accessed, and the response time of the transaction is much longer.

The business user view of the system is shown in Figure 4.7, below.

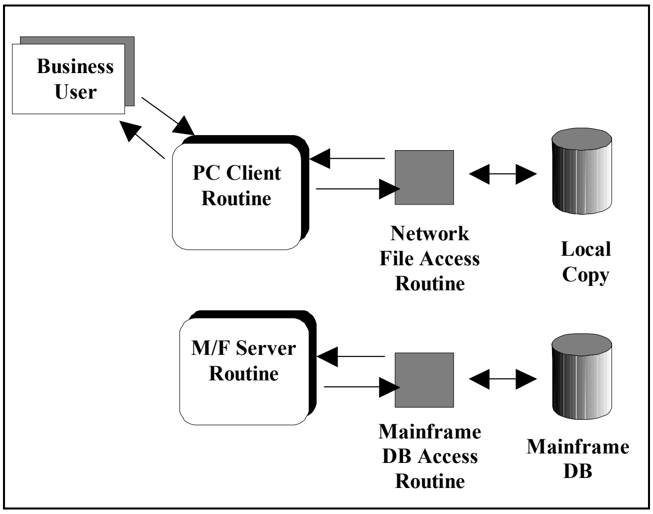


Figure 4.7 - User view of a client/server application

The physical location of the data is of little concern to the business user. The user enters data and the required responses are displayed. The fact that two physical databases are used is a technical solution chosen for performance reasons. From the user perspective there is only one logical database. Duplicate entity types in the two physical databases are perceived as single logical entity types. As there is only a single boundary enclosing both the local and mainframe databases, logical transactions are recognised only as their inputs and outputs cross this boundary. Inter-operation of the application components is invisible to the business user and no logical transactions are recognised in the messages passing between PC and mainframe.

Example transactions might be…

ID	Transaction Name	Event or Query	No. of Input DET	Response	No. of Output DET	Entity Types Referred to	No. of ER
T001	QUERY_CUSTOMER_SUMMARY_DATA	Q	2	Display Summary Data	5	Customer	1
T002	QUERY_CUSTOMER_DETAIL_DATA	Q	2	Display Detail Data	20	Customer	1

However…

…three groups of development staff were involved in building this application. And it is easy to imagine a scenario whereby IS management is interested in the size of the application attributable to each team. That is, how much did each team build?

Figure 4.8 shows the view of the project team that was involved in the development of the PC client aspect of the application.

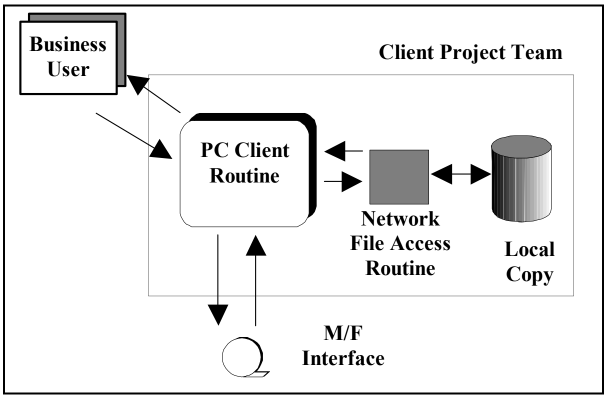


Figure 4.8 - The PC Client Project Team's View of the Boundary

In this scenario the user is the business user who initiates logical transactions triggering processing of data held on the network file server.

As described previously, when the user double clicks the mouse button, detailed information is retrieved from the mainframe database, via a mainframe server routine. From the PC Client project team viewpoint, this is achieved via an interface with the mainframe application. The PC client outputs some data across the application boundary, sending it to the mainframe server routine. This, in its turn, processes the message passing back across the application boundary the data requested from the mainframe.

Example transactions might be…

ID	Transaction Name	Event or Query	No. of Input DET	Response	No. of Output DET	Entity Types Referred to	No. of ER
T001	QUERY_CUSTOMER_SUMMARY_DATA	Q	2	Display Summary Data	5	Customer	1
T002	QUERY_CUSTOMER_DETAIL_DATA	Q	2	Request to mainframe	2	Customer System	2
T003	RECEIVE_DETAILS_FROM_MAINFRAME	E	20	Display Customer details	20	System	1

In MkII FPA, and from the perspective of the project team, the size of the PC Client application includes the DET's output across the application program interface with the mainframe as part of the output side of the triggering transaction. Additionally, the response from the mainframe is regarded as the input side of a distinct logical transaction, triggering processing and the corresponding output to the business user.

Figure 4.9 shows the project from the viewpoint of the mainframe project team.

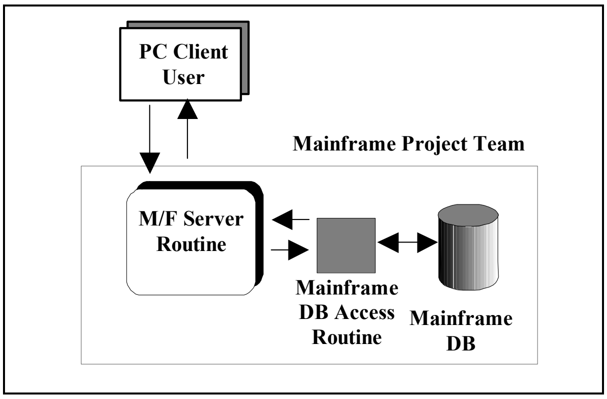


Figure 4.9 - The Mainframe Project Team's View of the Boundary

To the mainframe project team, the user is the PC Client. It is the PC Client that initiates logical transactions requiring data to be processed on the mainframe.

Input from the PC Client crosses the application program interface i.e. the boundary, is processed by the mainframe server routine, which accesses the mainframe database, and causes output to be sent across the application boundary back to the PC Client. In MkII FPA, and from the perspective of the project team, each distinct kind of message received from the PC Client is regarded as the input side of a separate logical transaction, with corresponding processing (in the form of references to logical entity types stored on the mainframe database) and an output message (sent back to the PC Client).

Example transactions might be…

ID	Transaction Name	Event or Query	No. of Input DET	Response	No. of Output DET	Entity Types Referred to	No. of ER
T001	QUERY_CUSTOMER_DETAIL_DATA	Q	2	Pass Customer Details to PC	20	Customer	1

In addition to the PC Client and mainframe development teams, there was an infrastructure team that developed the routines which were used for file access on the network server, and on the mainframe. These routines were built so that they could be used by a number of future projects, that is, the infrastructure team was responsible for designing and constructing re-usable components. Figure 4.10 shows the infrastructure team's view of the project.

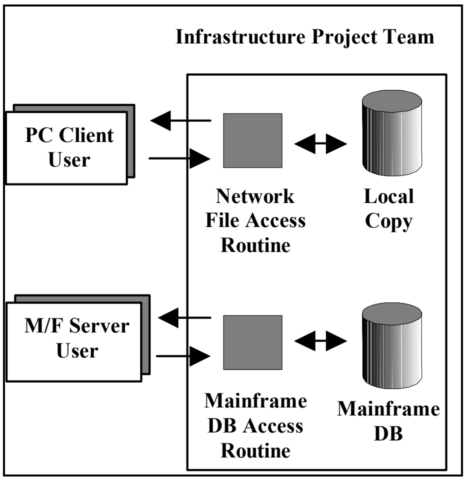

Infrastructure Project Team
PC Client User
Network File Access Routine
Local Copy
M/F Server User
Mainframe DB Access Routine
Mainframe DB

Figure 4.10 - The Infrastructure Project Team's View of the Boundary

The infrastructure team produced two potentially re-usable routines. One routine provides database access to the local summary database stored on the network server; the other routine provides database access to the detailed data stored on the mainframe database.

In MkII FPA, and from the perspective of the PC Client and Mainframe project teams, both these routines are extensions to the programming language used for software development. They are *bottom-up components* on a par with other programming language instructions such as IF…THEN…ELSE, MOVE, READ…FILE, WRITE…RECORD, ADD A TO B, etc. Hence they are invisible to the business users, and to the PC Client and the Mainframe application project teams.

They would not normally be sized by MkII FPA as application functionality.

However, if it is desired to measure the potentially re-usable functionality developed by an infrastructure support team, such routines can be measured using MkII FPA but great care should be taken. MkII FPA was designed for use with software in the business applications domain, and applying it to infrastructure software is extending its design envelope. Certainly, FPA values obtained from these distinct domains should not be mixed together.

Note, that the level of granularity (i.e. the abstraction used) is more finely grained for the infrastructure components than that used when drawing the application boundaries for the PC Client and Mainframe components (which is itself a more finely grained abstraction than that used to draw the boundary from the viewpoint of the business user). Also, separate boundaries should be drawn around each distinct set of related bottom-up components, and disparate and unrelated components should not be lumped together. This is because the logical model used to describe the context of one component will be different from the logical model used to describe a separate, unrelated component. Any similarity is by chance and is liable to lead to future confusion. Where different things are described, making clearly distinct descriptions leads to less eventual ambiguity.

Hence, in the case above, the routine used to provide access to the network database is enclosed in a distinct boundary, and is sized separately from the routine used to provide access to the mainframe database.

Distinguishing Deployed Size and Delivered Size

The foregoing examples illustrate the impact of the user view and the corresponding understanding of the application boundary.

- The business user viewpoint results in a size measure of the business functionality independent of the application architecture and is sometimes referred to as the ***deployed size***.
- The project team viewpoint results in a size measure of each application that is separately designed, developed and maintained, and the sum of such measures is sometimes referred to as the ***delivered size***.
- The infrastructure team's viewpoint results in a size measure of all the software components that extend the functionality delivered by the underlying operating system, middleware, database management systems, network communications systems, etc., delivering this functionality not to the business user, but to applications developers.

Each of the three measures above are derived using distinct abstractions and levels of granularity in the descriptions used to describe their respective contexts. The terminology used in these descriptions will invariably be different. Hence, such measures should be kept distinct. It rarely makes any sense to add them together.

4.4.6 Accounting for Housekeeping

4.4.6.1 Database Management and Integrity Functions

Database management and integrity functions are only included in a MkII FPA size when they represent agreed *business* requirements, i.e. not technical or quality requirements. So a good question to ask is, 'Is this function specifically required by the application user as part of the business requirements?'. If the answer is TRUE, then it is justifiable to include it in the FP count.

For instance, it is normally expected that a software application is complete and internally consistent. Hence any functions required to check and maintain database integrity and performance (e.g. database rebuild routines) are regarded as implicitly required implementation detail, and dictated by the choice of a software solution. Such functions would be expected to be provided as a part of the operational environment and to be executed by IS staff. They would not be included in a MkII FPA study as distinct logical transactions.

However, if the business user requires specific functions to enable the end user to perform integrity checks, to determine the percentage free space available and to initiate database rebuild processing, then such functionality would be counted as logical transactions. The difference here is that, in the second case, the logical transactions have to be provided with an interface usable by business users, and have to be subjected to the same development disciplines as any other business functionality intended for staff without special IS knowledge, skills and tools.

If the application provides a 'deletion routine' or tool for error correction, then this is a legitimate function that can be measured. Generic tools automatically provided by the operational environment should not be counted

4.4.6.2 Operational Logs

If a log is a necessary part of maintaining the integrity of the application e.g. it provides information necessary to enable database back-up and recovery routines to execute successfully, then it is normally regarded as implicit in the choice of a software solution to the business problem and is not counted.

However, if the log is required by the business user as an additional, user controlled function e.g. to provide management with a means of determining the volume of transactions processed per period, then it is regarded as user-specified functionality and any related logical transactions would be recognised.

Example 1. A transaction is required to process many records and produce the sum total of certain numeric values. If the business requirement is only to calculate the total, any logs that are produced as a by-product are not '...of interest to the user...' and are not counted.

Example 2. If, in the above transaction, the user specifies that an audit log is required to show that all records have been processed, then the output is '...of interest to the user...' and is counted.

4.4.6.3 Security Access

If the Logical Transactions involved in gaining user access are provided by the operating environment, they should not be counted. Additional requirements for security allocated to the software being sized (i.e. giving rise to Logical Transactions specific to and provided by the software) should be counted.

4.4.7 Batch Programs

MkII FPA is concerned with the measurement of the Functional Size of software. The Functional Size excludes, by definition, consideration of the qualitative (i.e. performance) characteristics of the software. Hence, MkII FPA makes no distinctions regarding speed of response, volume of transaction throughput, etc. It is concerned only with the logical functions.

Batch programs are a physical implementation of a logical function, and batch mode is chosen usually due to issues such as the need to process large volumes of input with little user intervention. So far as a logical model is concerned, whether a function is implemented in batch or on-line mode is of no concern to MkII FPA. Hence, the same process is used to identify logical transactions and to calculate the size for functionality implemented in batch or on-line.

Although batch program flows often contain many steps, implemented as distinct programs connected by intermediate files, usually only a few of these programs process any input or output across the logical application boundary. Typically, these include the first and last programs in the flow, and any programs that generate printed reports. Figure 4.11 illustrates a batch flow containing several program steps.

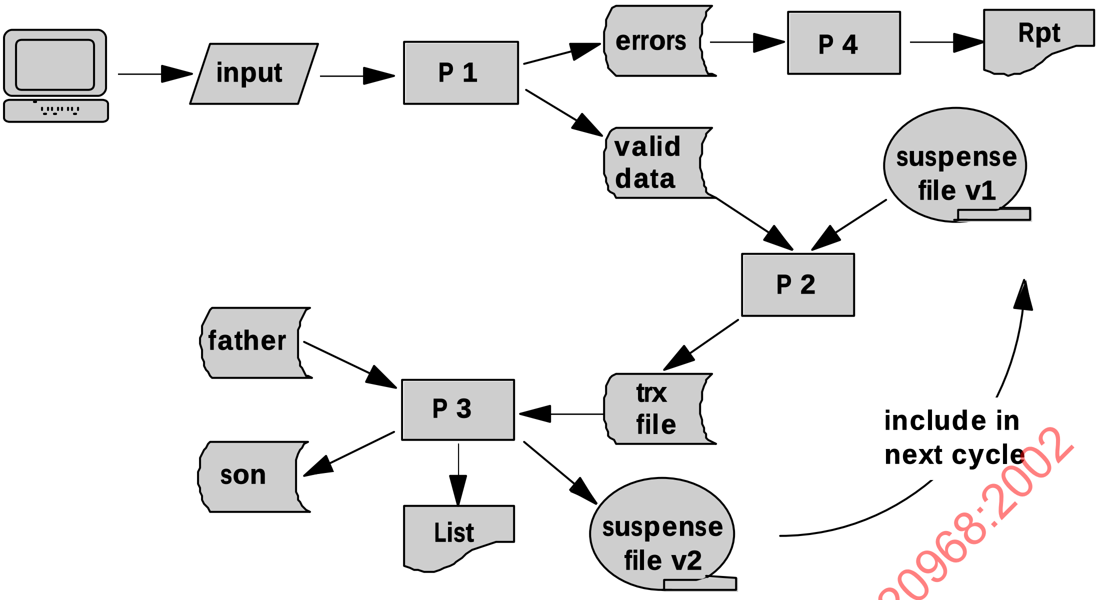


Figure 4.11 - A Typical Batch Flow – Transaction & Master File Collation

In the illustration, data about events are input and validated by program P1. Input errors are output by a print program P4. Valid transaction data is sorted and merged in program P2 with transaction data suspended from the previous execution of the batch flow due to mismatches with the masterfile. The sorted file of transactions is read by a collation program P3; this performs a father/son update producing a new (son) generation of a masterfile, a list reporting all the updates, insertions and deletions made, and a new version of the suspense file for use the next time the batch flow is executed.

In the example flow, data input for all logical transaction types crosses the boundary as the input to P1. Hence this input needs careful analysis to identify the distinct logical transactions. Output for all transaction types crosses the boundary as the list from P3 and the error report from P4. These two reports need careful analysis so the outputs can be properly attributed to the correct logical transactions. To understand the entity types, examine and perform a third normal form data analysis on the father/son masterfile.

Batch flows have the characteristic that they are concerned with small numbers of logical transactions (e.g. Insert_Master, Credit_Master, Debit_Master, Delete_Master), processed in relatively large volumes. Although the processing of the component logical transactions is distributed across a number of program steps, the functional size of batch suites typically is quite small.

4.4.8 Camouflaged Transactions

4.4.8.1 Retrieve Before Create, Update or Delete

Often in practice, an Update transaction may be required to be preceded by a Retrieve transaction. The Retrieve is provided to help the user decide whether to proceed with the Update or not, after sight verification that the correct record has been selected for update. This **'Retrieve-before-Update'** transaction may be identical to a normal Read transaction (in which case it is only counted once for the software), or it may be required to be different from a normal Read, in which case it should be counted as a separate transaction from the normal Read.

Examples of differences would be if the 'Retrieve-before-Update' did not show all the information available via a normal Read, or if use of the normal Read transaction was restricted in certain ways, e.g. so that the data were presented in non-updateable form, or if it had other distinguishing processing.

The same considerations would apply to an Add or Delete function which may be preceded by the need for a confirmatory enquiry. However, where the 'Retrieve-before' process is identical before an Add, Update and Delete transaction, count only one 'Retrieve-before' transaction-type.

4.4.8.2 Transaction Chaining and Nesting

For the convenience of some users, logical transactions may be linked by the software so that, on completion of one, the application automatically navigates to another.

Such chaining together of several transactions typically is arranged in response to usability requirements, e.g. to facilitate rapid navigation for telephone sales staff. Often, information entered as input to one transaction is stored locally and reused as input in subsequent transactions, without requiring the user to re-enter it. One transaction may be said to ***inherit*** data input to an earlier transaction.

In MkII FPA, logical transactions are distinguished as the lowest level business processes that leave the data content of the application self-consistent. So, although transactions may be chained navigationally, if successive processing steps leave the application self-consistent, then each step is regarded as a distinct logical transaction.

Navigation is not counted in MkII FPA, so irrespective of any chaining, all the DET's necessary as input for any one transaction are counted, irrespective of efforts to maximise usability and of whether devices such as inheriting data from earlier transactions are used.

Similarly, in some applications, requirements may be resolved such that, during the course of one transaction it is necessary for the user to initiate (and possibly complete) a distinct separate transaction. For example, part way through a Create_Order transaction, it may prove necessary to perform a Query_Stock enquiry. The Query_Stock enquiry transactions is logically distinct; it may be reached by navigation from several parts of the system. Hence it is counted as a separate logical transaction with unique processing, distinct from the Create_Order transaction.

4.4.8.3 Synonyms

Large software projects can involve large numbers of people, both from the user and the supplier side of the project. During software requirements capture, the disparate requirements of a wide variety of users with a variety of purposes are considered. Also, several different people may provide their personal views of the requirements of a single user group. Under these circumstances, it is likely that different people will use distinct terminology. The result can be that the catalogue of logical transactions may contain several transactions with different names but requiring exactly similar processing. That is, one transaction exists with several synonyms.

It is the analysts' responsibility to identify such situations and resolve them, clarifying whether there are really multiple transactions or a single logical transaction with multiple names. Note however, that MkII FPA measures functional requirements, and similar requirements may be imposed by distinct user groups for their own purposes. If distinct user groups require similar functionality, this functionality is, by definition, delivered to distinct customers. Hence, it is admissible to keep the requirements of each user group distinct and to size the relevant logical transactions due to each user group, if they are really different.

Example 1:

The user wants a report of sales performance, with a separate report of the sub-totals for:

- Area
- Department
- Salesman.

All the information is reported at a single point in time and the various elements of the reports are synchronised, aggregated, and delivered to a single user group on one report. The reports fulfil a single user need, the provision of Sales Performance information. Therefore a single logical transaction is recognised, the output size of which consists of the entire set of DET's on the report, including the sub-totals.

Example 2:

Alternatively, if there were three different customers (e.g. Area Manager, Department Manager, Sales Manager) each with a distinct requirement for a report on Sales Performance, then there would be 3 separate logical transactions, even though the requirements of all three had been satisfied by the developers by providing copies of the same shared physical report. In this case the size of each report should in theory include only the DET's requested by each respective customer.

4.4.8.4 Distinguishing logical requirements combined into a single physical implementation

A frequent failing in understanding user requirements derives from inadequate functional decomposition of the problem. In MkII FPA, the measure of Functional Size is very sensitive to the recognition of all the required logical transactions.

Therefore, when sizing requirements during development, care should be taken to ensure that functional requirements are first decomposed to the lowest level of business process required by the user.

Example 1:

A user group requires that customer data be updated either on-line, or stored and updated automatically at the end of the working day. As a first attempt, the requirement is described by the transaction

- Update Customer Record.

If the user is really unconcerned regarding the precise implementation and it makes no difference to the business when customer details are updated, then this may be sufficient.

However, if the user wishes to be provided with a choice to be made at run time, enabling the user to specify whether a specific customer's details should be updated immediately or the update delayed until later, then a single transaction would be insufficient. It is not the 'lowest level business process...'. A further decomposition of the requirement results in the recognition of the following:

- Update Customer Details on-line
- Update Customer Records off-line

On even closer analysis, it will be realised that in order to perform an off-line update, new customer details must be captured and temporarily stored and a subsequent update triggered by some stimulus – in this case, the tick of the clock or some other signal indicating the end of the working day. Therefore, in the final analysis, there could be a total of three logical transactions as follows:

- Update Customer Details on-line
- Capture and Store Customer Details
- Apply Customer Updates at end of day

There are obvious performance differences between these later requirements and the earlier single update transaction. Not least, the degree to which the customer details accurately reflect the events that have occurred in the external world is different between the two cases. In the second case, the deferred update can result in customer details lagging the external world by one whole day. This may or may not be a desirable feature – only the user can comment adequately.

4.4.8.5 Reused Components

How to identify the transactions represented by modules that are used in many applications or many places in a single application may cause uncertainty.

A logical transaction may be implemented so that it makes use of a common module, for instance retrieving address details for a client. This re-use does not affect the size of the logical transaction. How the logical transaction is physically implemented is not at issue: MkII FPA is concerned with the logical view, so the reused module is 'invisible' to this view. The input DET's, references to entity types and output DET's are counted in the usual way.

Alternatively, an entire logical transaction may be implemented so that it is available from multiple places within the application. In this case also, the transaction is counted only once, regardless of the number of different ways in which it is possible to navigate to it.

If software which implements one or more logical transactions is utilised in more than one application, then its functionality should be counted in as many applications as it appears in (once per required transaction per application) as it is to be assumed that it implements distinct user requirements in each such application. The fact that implemented software is reused in this way does not affect the Functional Size, which is a measure of the requirements. Reuse will, of course, affect the effort, time and cost of software development, which will typically be lower than if reuse where not an option. The benefit therefore, will show up as improved capability on the part of the developer. This is entirely reasonable – a business problem of measured size is being solved more efficiently.

4.5 Identifying Entity Types

4.5.1 Basic Rules for Counting Entity Type References

In MkII FPA, as a measure of the size of the processing phase of each logical transaction, we count the *primary* entity *types* referenced in the processing.

Count the number of entity *types* referred to. Do not count the number of times they are referred to within a transaction; e.g. if the customer entity-type is both read and updated within a transaction, count only one entity reference. Similarly, count only one entity-type, regardless of the number of occurrences of the entity-type. (Hereinafter we will use 'entity' and 'entities' for simplicity, where there is no risk of misunderstanding.)

Count *primary* entities, not non-primary entities; the latter are 'lumped together' into the 'System Entity'. The latter should also be counted as a single entity reference, if any of its non-primary entity components are referenced in a Logical Transaction (see further in 4.5.3 below), even though the System Entity is not considered to be a primary entity.

It is recommended always to carry out entity-relationship analysis, or relational data analysis to produce an entity diagram for the application before starting the MkII FP count. Do not count physical files, whether permanent or transient, as these could be highly misleading.

4.5.2 Entity Types

An entity type is something (strictly, some type of thing) in the real world about which the business user wants to hold information. Data element types hold information about entity types. For example:

> 'Employee' would be an entity-type in a personnel system. ('Fred Bloggs' might be an entity occurrence.)
>
> 'Employee date of birth' is a Data Element Type ('DET') which holds information about the employee. ('Employee date of birth' is highly unlikely ever to be an entity type. We do not want to know anything about it.)

If relational data analysis has been carried out leading to a set of tables, then the subject of each table will be an entity type (but not necessarily a primary entity type).

4.5.3 Distinguishing Primary & Non-Primary Entity Types: the 'System Entity'

Primary entities are defined as '**the main things in the real world about which the application being sized is required to hold data**.'

There are several other hints and tips which can help distinguish primary and non-primary entity types. The following list is meant to be an aid to decision-making, in no particular order of importance, to supplement the main definition above.

Criteria	Primary Entity	Non-Primary Entity
No. of Attributes	Several, with values changing frequently (e.g. quantities)	Very few, e.g. only code, name, description; values rarely changing
Occurrence Frequency	Count of occurrences may change often, e.g. no. of orders or payments in a system changes all the time	Permanently fixed, or rarely changed
Ownership	Probably owned by the application being counted	May well be owned by the organisation, and used by several applications
Time of change of attribute values	During normal system operation	Usually outside normal system running
Authorisation for change of attribute values	No special authorisation; performed by normal system user	Performed by a system administrator
Entity life-cycles	Usually many stages or status's	Usually only 'live' or non-existent; maybe last-period and current attributes are maintained

Note that there is nothing absolute about the distinction between primary and non-primary entity types. There can also be borderline cases, for example the 'Grade' entity in a Personnel system, which has attributes such as Grade_Code, Grade_Name, Lower_Salary_Limit, Upper_Salary_Limit, Date_of_Validity, etc. This has few attributes, not changing frequently, and requiring special authorisation for changing, but being fundamental to the personnel business may well need to be considered as a Primary entity.

Within an application, the primary and non-primary entities are classed as such for the entire application, and this classification should not vary by transaction.

However, an entity type which is non-primary for one application could be primary for another. For example, in an application to maintain bank accounts, 'customer-type', could be an attribute of the entity type 'customer'. Third Normal Form analysis of the whole business area would show 'customer-type' to be an entity, but as the 'maintain bank accounts' application does not want to know anything about 'customer type', it is a non-primary entity type in this application. However, another market analysis application might want to accumulate a lot of information each month about each customer type, and hence for the latter application, 'customer type' would be a primary entity.

The allowed values of attributes of non-primary entities are usually maintained in 'Master Tables', which are usually referenced for validation purposes by many types of transactions.

As a simplifying convention of MkII FP counting, all non-primary entity-types are 'lumped together' in what is known as the 'System Entity'. The System Entity is counted once in any Logical Transaction which references any of its component non-primary entity-types. The System Entity should ***not*** be regarded as a Primary Entity for the purposes of other rules in this CPM (e.g. see Chapter 5, ‘Drop-down/pop-up List Box’).

4.5.4 Entity Sub-Types

In some instances, it is necessary to distinguish ‘sub-entity types’, and count them separately.

A sub-entity type (or ‘sub-entity’) is a specific type or variation of a primary entity, for which there are distinct processing rules e.g.

ENTITY TYPE	SUB ENTITY TYPE
Account	Savings Account, Deposit Account
Employee	Permanent, Temporary

The characteristic type/sub-type relationship is, for example, ‘all savings accounts are accounts, but not all accounts are savings accounts’.

Sub-Entities are counted separately when different processing logic is used in the transaction in which the entities are involved e.g.

- Transfer of funds from a "Savings Account" to a "Deposit Account".

  Each is a sub-entity of "Account" but different processing logic is involved for each sub-entity in this transaction. Count 2.

- Create a new occurrence of ‘Permanent Employee’.

  Count 1 for the reference to ‘Permanent Employee’

- Count all Employees, referencing both Permanent and Temporary.

  No distinction between Permanent and Temporary, therefore count 1 for the reference to ‘Employee’.

- List the names of all Temporary Employees and their individual pay rates (the Temporary sub-entities have additional attributes for individual pay rate, probation and progress).

  Logically, we are only dealing with one entity-type, so count 1 for the reference to the Temporary Employee sub-entity.

- Add up total costs for all Employees.

  Count 1 for the reference to the Permanent and 1 for the reference to the Temporary sub-entity types (different processing of pay rates). Total count 2.

4.5.5 Involuted Entity Types

(Also known as ‘recursive’ or ‘self-referential’ entity types.)

There is a further exception to the above entity counting rules. If an entity-type is related to itself, e.g. as in a parts list or customer hierarchy, then for a transaction that traverses up and down the hierarchy, count two entity references, one for the initial reference and one for the repetitive loop, irrespective of how many times the hierarchy is traversed.

4.5.6 Logical Entities in Batch Systems

MkII FPA is concerned with counting references to *logical* entity *types*, not physical files. Distinguishing these in batch systems requires care. In the example batch system of Fig. 4.11, the transaction stream may contain several logical transaction types (e.g. create, update, delete), and the processing could involve references to, for example two entity-types, such as Order_Header and Order_Item. The example shows several physical files in the processing sequence, but in this case there are only two entity references involved in each of the logical transactions. When analysing the system using MkII FPA, examine physical files to help identify entity types, but only count the entity type references for each transaction, not the physical files involved.

4.6 Identifying Input and Output Data Element Types

The general rules for identifying and counting input and output Data Elements Types ('DET's') are as follows:

Types versus Occurrences

Count uniquely processed types of data element, not individual occurrences.

Single, Composite, Multi-use, etc., Data Elements

Count composite DET's, e.g. name & address, as a single DET, unless there is a requirement to handle (e.g. validate) the separate parts (e.g. post code), in which case count the individual DET's actually processed separately.

Count dates shown as day/month/year for example as one DET, unless there is a specific processing need to distinguish the separate parts of the date.

Count inherited input the number of times that it is used. Inherited input is where a DET is input once, stored locally, and provides input to two or more transactions.

If the application is required to provide default values for input DET's and displays them to the user for acceptance or over-write, count once as normal data inputs.

Where an input DET is required to be re-displayed as output or after validation failure, count both as input and output.

Regard all required error messages as an occurrence of element type 'error message', i.e. count as 1. NB. Operating system or network error messages not created or maintained by the project team do not attract any function points if they are part of the technical environment used. But if the error message contains embedded business data which varies with the occurrence of the error message, count the DET's involved separately.

Arrays

Where data can be presented as an array, the count = (number of column types) x (number of row types).

For example, if the user wants to display payments for 12 successive months , this could be represented as an array with 2 columns and 12 rows:

	month	value
payment	April	12
	May	57
	June	56
	July	78
	August	23

September	121
October	68
November	83
December	55
January	60
February	23
March	92

The count would be:

2 column types [month & value]) x (1 row type [payment]) = 2.

This would be the count for this type of information, however many row occurrences were added.

If in this example, a total was also required for the value column:

	month	value
payment	April	12
	May	57
	...	
	March	92
total payments	-	728

The count would be increased by the size of the new type of data as follows:

(1 column types [value]) x (1 row types [total payments]) = 1,

giving a total count for the whole table of 3.

Menus, and Transaction Initiation

Do not count menus or "PF Keys" as part of any logical transaction when used simply for navigation purposes. MkII FP’s measure the user information processing functions provided. The menu structure simply allows the user to move around between the functions provided; it does not contribute to any information processing function itself.

However, sometimes a menu is used not only to select and invoke a transaction but also to enter data, e.g. parameters passed to the transaction through the menu. Count any such parameters passed as though they had been input directly to the transaction. A similar exception is where a PF Key or menu selection directly initiates the transaction. Data that can be used to initiate a transaction can include selection, comparison or permissible range parameters and these are counted as input DET’s.

When transactions are "automatically" initiated the real trigger must be found. For example a date may be a valid trigger. A change of date, for example the end of the month, is a valid external event and is treated as an input to the application, triggering a transaction. It is counted as one input DET. Similarly, transactions that are run periodically are also initiated by temporal events (date or time changes).

The Transaction Type Identifier

Where the input part of a logical transaction includes a ’transaction type identifier’ which is necessary for the system to know what to do with the input, this should be counted as one DET. Consider the following: an application has two transaction-types, each with an input message consisting of the fields Customer_ID, Account_ID and Amount. Unless the need for a ‘transaction type’ field is recognised, the application has no way to determine that these two transactions are, respectively, Credit_Account and Change_Loan_Limit.

Field Headings, Headers, Footers, etc

Do not count any field labels, titles, or underlines that contain no variable information Do not count standard headers or footers that automatically appear on every report or screen, e.g. copyright messages.

Physical Screen Limitations

Ignore physical screen limitations, for example many applications have output that consists of lists of data that spread over several physical screens, requiring paging, and scrolling to access the whole output. The count is the number of unique DET’s displayed, regardless of the number of physical screens.

Granularity and Uniquely Processed DET’s within Input/Output Streams

If input/output data is in the form of a data stream whose structure comprises more than one DET, then the number of input or output DET’s to be counted is identified by applying the following logic:

1. If the logical transaction processes the data as a single DET, regardless of the structure, there is one DET: count 1 input or output.

Example. Archiving of data records. The record structure would comprise many DET’s, but the whole of the record would be stored without validation or changes to the formatting.

2. If the logical transaction processes (validating or formatting) individual DET’s or groupings of DET’s within the structure then there is one DET counted for each distinctly processed DET or grouping of DET’s.

Example (a) Update an existing record without validation. This would typically involve the extraction of the DET’s that represent the unique key. The unique key would be counted as one DET and the remainder of the DET’s would also be counted as 1 DET.

Example (b) Update an existing record with validation before the update. Each unique DET, or group of DET’s, required to be validated would be counted .

Input/Output in Different Forms

When the same information is required to be presented in different ‘forms’ in the input or output parts of logical transactions, great care is needed in interpreting the MkII FPA counting rules.

By different ‘forms’ of presentation of the same information, we mean, for example

- output in both numerical tabular form or in graphical form (e.g. bar chart, pie chart, etc.)
- input and output in different natural languages

- input and/or output in both black and white, and in coloured presentations.

As the information is the same in all cases, the interpretation might be that the MkII FP size should only count one ‘form’ and ignore requirements for presentation in other forms. But the stated aim of the MkII FPA method is to produce a size which is ‘suitable for the purposes of performance measurement and estimating in relation to any activity associated with the software product’.

Being required to produce information in different forms requires more functionality of the software, so ignoring the ‘extra size’ associated with producing the ‘extra forms’ (e.g. presentations in different languages, etc.) would produce an unfairly low size measure. The following examples are of several different ‘forms’ of presenting information, designed to illustrate how the MkII FP counting rules should be applied.

Example 1: An array of values is required to be printed in two forms, as a numeric table and as a pie chart. Three cases can be distinguished, depending on what are the finally agreed requirements.

a) One LT, with simultaneous display of the data in the two forms. Count two output DET’s, one for each form.

b) The requirements are for a separate LT for each form. Count each separate LT as per the normal counting rules.

c) The requirement is to give the user the choice of which output form he wants, depending on his selection of the value of an input DET. Count the input DET and two output DET’s.

Example 2: A product code is required to be printed by one LT on a label in both normal numeric form, and as a bar code. Treat as Example 1a).

Example 3: All existing output in black on white is to be converted to colour for aesthetic reasons. Do not count any added or changed DET’s in the Change project. This type of work cannot be reliably sized with FPA.

Example 4: All existing output in black on white is to be converted to colour which adds significant information, e.g. red indicating share prices are falling, green that they are increasing. Count all changed DET’s in the size of the Change.

Example 5: Colour is introduced to highlight which fields are in error. Do not count more than once - the normal error message rules apply.

Example 6: All field labels in one language are to be made available in a second language, via a sign-on option. The sign-on input DET may be counted, but labels are literals and are not normally counted in MkII FP rules. This type of work cannot be reliably sized with FPA.

Example 7: Field labels are required to be stored as variables, with different values for different languages. Count the field labels as DET’s.

Example 8: DET values are required to be displayed in more than one language. These DET values have to be stored as attributes of the same entity, so each language variation of the DET is counted, on input or output, as appropriate. Note that only one entity reference is counted for the retrieval of these DET’s, however many languages.

Example 9: A list of debtors with amounts owing is required to be printed in one LT either in alphabetic order of debtor name, or as a user option, in order of increasing debt. Count only one set of DET’s for the output, as the other sequence provides the same information. Count one input DET for the control of the sort sequence.

Example 10: A club membership reporting program enables lists of members to be printed, sorted in various ways, and with selected sub-sets of information ranging from a full listing, to selected names and addresses. The size depends on the agreed user requirement:

a) The user requirement is for each report to be pre-programmed. Count one LT for each report.

b) The user requirement is for a tool with which the user can generate his own *ad hoc* reports. Count all the input DET's which can be used to determine the output, all the entity references which could be required, and all the output DET's which can be produced. The viewpoint here is of the logical requirements for the tool, not the application of the tool.

Example 11: A report has been formatted for a 132 character line length, and has to be re-formatted for an 80 character line length. Count all the DET's affected by the Change.

Example 12: A report is normally sent to a laser printer, but the Change requires it to be sent to another laser printer for which a new device driver is needed. Do not count any additional DET's at the application level, because nothing is changed at that level. (If the device driver software has to sized, develop a logical model for the transactions of the device driver.

(Blank page)

5

Measurement Guidelines for Specific Situations

5.1 Counting Graphical User Interfaces (GUIs)

5.1.1 Introduction

This Section supplements Step 5 of the Mk II FPA process contained in Chapter 3 of the Counting Practices Manual. It contains the guidelines to be used when applying FPA to an application which utilises a GUI. The Section describes the rules for measuring the logical requirements of the application implemented with the GUI, and not for measuring the functionality of the GUI software environment itself.

5.1.2 Basic Principles

A fundamental principle of MkII FPA is that it sizes the logical requirements, irrespective of how they are implemented. A GUI is primarily a means of implementing a solution to the logical functional requirements to facilitate ease of use. Therefore in principle use of a GUI does not affect the functionality nor change the size of the requirements. FPA counts the underlying logical requirements represented in the GUI.

However, the features that a GUI environment provides may lead the developer to include increased functionality that was not specifically requested. For example, what could have been a single input Data Element Type ('DET') in a conventional (non-GUI) system could evolve into an input DET plus a single input enquiry in a GUI environment. It is customary to include such additional enquiries as additional logical transactions in the count, even though they may not have been explicitly specified as required by the user.

A GUI provides elements enabling human-computer interaction. The table below lists a number of such elements. Alongside each item is an indication of the interaction it supports. The remainder of this chapter describes how the element should be counted, if at all. Some GUI controls do not support any logical function specific to the application being counted but contribute solely to what are referred to in this chapter as 'the non-functional' characteristics. These are never counted at all.

The usual operation of the various elements is summarised below. However, new approaches and new elements are frequently being produced by GUI tool providers, so always carefully examine a GUI element before counting and apply the basic principles.

5.1.3 GUI Table

GUI Element	Counting Area
Check Box (not exclusive)	Input/Output/Non-Functional
Combo Box (Edit Box + Drop Down)	See Edit Box/Drop Down
Control Bar (i.e. a series of icons)	See Icon
Control Button (OK, Cancel etc.)	Non-Functional
Dialog Box	Input/Output
Drag and Drop	See Icon
Drop Down/Pop Up List Box	Output/Input/Non-Functional
Edit Box (enter or display box)	Input/Output
Icon	Input/Output/Non-Functional
Menu	Non-Functional
Non Text Annotation	Input/Output
Radio Button (mutually exclusive)	Input/Output
Screen Hotspot	Non-Functional
Scroll Bar	Non-Functional
Text Box	Input/Output
Touch Sensitive Screen Areas	See Icon
Window Control (Min, Max, Restore etc.)	Non-Functional

The GUI elements listed must be used to size an application in the same way that screen formats, print layouts etc., are used in character-based, batch and other non-GUI implementations.

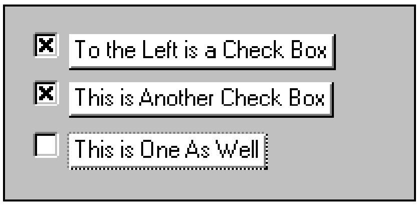
Many GUI elements govern aspects of the GUI environment itself and thus relate to usability, not application functionality.

- When sizing an application, the scope is restricted to the GUI elements that input or display business information.
- Where a GUI element physically represents both input and output the input and output parts are counted separately with their appropriate transactions.
- Check Boxes and Radio Buttons can be used in arrays and when used as such should be counted as defined above in Chapter 4.

5.1.4 Counting Guidelines

This section describes the counting guidelines for those GUI elements which are involved in a logical transaction. Those elements which are purely part of the non-functional characteristics have not been listed.

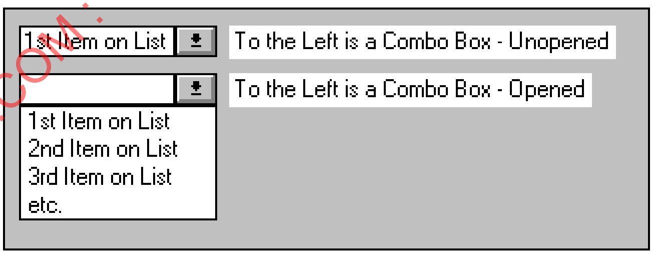
5.1.5 Check Box



Check Boxes allow data to be entered or output for a characteristic which has a Yes/No answer, for example, ‘does the customer have a valid driving licence?’ They can also be used as navigation i.e. the use of a check box can act as a menu selection to control the display of subsequent screens.

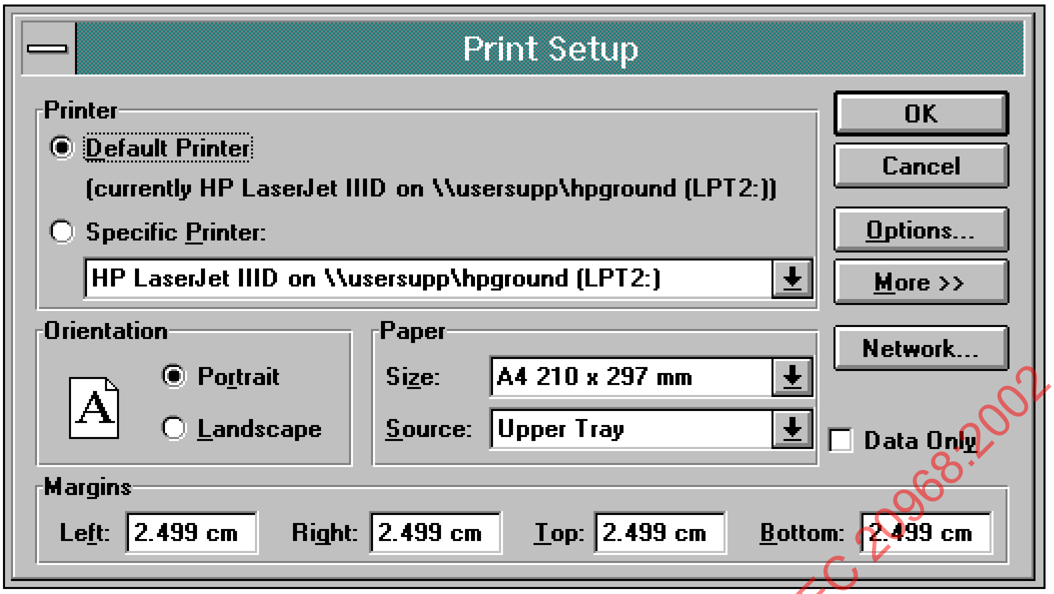
- Count each check box used to input business data for a transaction as an input DET.
- Count each check box used to output business data for a transaction as an output DET.
- If a check box is acting as a form of application navigation then it should not be counted as part of any transaction.

5.1.6 Combo Box


1st Item on List
To the Left is a Combo Box - Unopened
To the Left is a Combo Box - Opened
1st Item on List
2nd Item on List
3rd Item on List
etc.

A Combo Box is a combination of an Edit Box and a Drop Down List. Each of these parts should be counted separately according to its own rule.

5.1.7 Dialog Box



Print Setup

Printer

● Default Printer

(currently HP LaserJet IIID on \\usersupp\hpground (LPT2:))

○ Specific Printer:

HP LaserJet IIID on \\usersupp\hpground (LPT2:)

Orientation

● Portrait

○ Landscape

Paper

Size: A4 210 x 297 mm

Source: Upper Tray

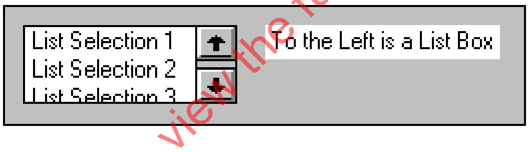
□ Data Only

Margins

Left: 2.499 cm Right: 2.499 cm Top: 2.499 cm Bottom: 2.499 cm

A Dialog Box includes a number of control items such as edit boxes and control buttons. Each part should be counted separately according to its own rule. (In the above illustration, the functions provided by the GUI environment, and not by the application, should not be counted.)

5.1.8 Drop Down/Pop Up List Box



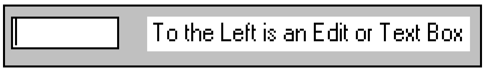
List Selection 1
List Selection 2
List Selection 3

To the Left is a List Box

List Boxes can act as input and output for different logical transactions and in some cases they may form part of the non-functional characteristics. A Drop Down may act as basic validation or help e.g. a Miss/Mr/Mrs prompt. It may also be used to display logical data e.g. a list of current branches for a banking application.

- The display of a list box with no reference to a primary entity is acting as a form of basic validation or help. No separate transaction should be counted for the list displayed (but count one input DET for the box).
- The display of a list box with reference to a primary entity almost certainly indicates a separate logical transaction.
- The display of a list box counted as a transaction is counted only once in an application no matter how many times it appears.
- A list box used to select an item for input to a transaction is counted as an input DET. If there is also a text box for typing in the data only one DET is counted for the ability to enter the data either way.

5.1.9 Edit Box



To the Left is an Edit or Text Box

- An Edit Box is used to enter data and is often used in conjunction with a drop down list in the form of a Combo Box. Entering data via an edit box

either by selection from a list or typing the data is counted as input data for the corresponding transaction.
- An edit box which displays business data is counted as output data.
- Count each text box used to enter or output data in the same way DET's are counted for a non-GUI application.

5.1.10 Icon

An Icon, or use of an Icon with drag and drop, may provide input data to a transaction e.g. dragging data to a report. It can also be used for data output e.g. display of an icon to indicate an element of business data. It may also be used as a form of application navigation i.e. non-functional.

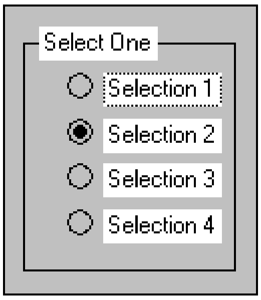
- The use of an icon to provide input data to a transaction is counted as an input DET for the transaction.
- The use of an icon to provide output data for a transaction is counted as an output DET for the transaction.
- An icon used as application navigation is not counted as it is part of the non-functional requirements.
- An icon could be a trigger to start up a logical transaction, e.g. an enquiry. If so, count one input DET, if there is no other input.

5.1.11 Non Text Annotation

Examples of this are audio annotation, video annotation, and the scanning in of documents. These can all be used as both input and output DET's. Almost invariably a scanned document treated as a single unit will count as one input DET.

- For each non text annotation used to input data to a transaction analyse the item and count each unique DET as an input DET.
- For each non text annotation used to output data to a transaction analyse the item and count each unique DET as an output DET.

5.1.12 Radio Button



Radio Buttons are for mutually exclusive items i.e. for each set only one button can be selected. They are used to provide a selection of options to a user, or can be used to display the current selection of an enquiry.

- Count one input DET for each set of mutually exclusive radio buttons.
- Check Boxes and Radio Buttons can be used in arrays and when used as such should be counted as defined in the rules for array counting in chapter 4.

- Count one output DET for each enquiry radio button display box.

5.1.13 Text Box

A Text Box is counted in the same way as an Edit Box.

5.1.14 Non-Functional Controls

Non-Functional Controls, such as Scroll Bars, Screen Hotspots, Control Buttons, etc. provide ease of use capability to an application, but do not provide any business application functionality. They are not counted for FPA purposes.

5.2 Approximate Sizing of Application Portfolios

As always the sizing method employed must be appropriate to the purpose in view. The most common reasons for measuring a portfolio are to monitor maintenance and/or support productivity, or to help estimate for a conversion programme. Before embarking on the sizing of an application portfolio one should, therefore, explicitly determine the degree of accuracy required. The degree of accuracy of the results should subsequently be stated (e.g. + or - 10%, 20%, 50%).

The principle of materiality also applies. The cost of a measurement exercise can exceed the value gained from the information. The cost of conducting a rigorous function point count of a live portfolio of ‘legacy’ systems, in the absence of up to date requirements documentation, can be very much higher than if there is high quality, up-to-date documentation. Note that the documentation gathered and understanding of those systems gained from such an exercise may be valuable in future maintenance: however, a cost benefit analysis should be undertaken before any major undertaking.

A simple and cheap-to-apply sizing method, eschewing function points, may be appropriate where the application portfolio is quite homogeneous. For example, if the business domains involved are comparable, and the technical architectures are similar and there is no requirement for external comparisons, then for such purposes measurement of the size of the source lines of code (‘SLOC’) according to some convention, or even of the compiled code, for example, may be sufficient.

Another approximation method which can be used where there is less homogeneity is Capers Jones' "backfiring". This is described in his publications and elsewhere. It depends upon a count of the source lines of code (SLOC) in each programming language for the systems involved. Definitions of lines of code are available from various sources. A number of tools (principally for IBM MVS mainframes) support automated SLOC counting. The results may be very variable and dependent on design and programming style. Best results will be obtained if the portfolio is predominantly 3GL rather than 4GL and using a limited number of languages. For some 4GLs the concept of a SLOC is meaningless. SLOC counts can be converted into function points using tables produced by Capers Jones and others, and then using other approximate conversion relationships to Mk II function points, if required.

A further alternative to a full function point count which will be more accurate than straightforward backfiring is to use ‘sampling and extrapolation’. In this approach the following steps would be taken:

i. Identify groups of systems with similar characteristics (e.g. financial, MVS COBOL).

ii Within those groups identify discrete areas of functionality and their corresponding physical implementation.

iii Identify readily countable characteristics of the physical implementations that may have a plausible correlation with functional size (e.g. size of source libraries, lines of code in various languages, numbers of screens, numbers of modules, numbers of database tables, numbers of logical entities).

iv Perform a function point count on a random sample of those areas of functionality (or even better of a carefully chosen representative sample of functionality).

v Perform multiple regression analysis (MRA) to determine for those areas the correlation(s) between the readily countable characteristics and the corresponding function point counts, and the confidence levels that may be had in the correlations. MRA is built into various spreadsheets. It may be desirable to have the assistance of a colleague with some statistical or mathematical expertise to advise.

vi Apply the correlation relationships thus determined for those readily countable characteristics to the areas that have not been function point counted in order to extrapolate and derive the total portfolio size in function points. If SLOC have been used then this constitutes an Change of the backfiring method.

vii Results should be published with the confidence level derived from the use of MRA attached to the results.

Finally, another alternative approach to a full FP count is to estimate the functionality using a scheme similar to the following. All the logical transactions are identified and classified in some way, e.g. as ‘simple’, ‘average’ or ‘complex’. An average FP size for each of the chosen classes is determined locally by sampling for the application area concerned.

Process / Function	Transaction	Create & Update			Enquiry			Delete
		s	a	c	s	a	c	a
Total Transactions								
multiplied by		4	12	20	3	10	17	8
Total Function Points								

s = simple a = average c = complex

Note that the choice of classes and the calibration of the multiplying factors (average FP size per transaction-type for each class) should be determined in the local environment.

5.3 Sizing Changes

(N.B. The term ‘Change’ is now preferred and is used for any project involving functional changes, i.e. additions, modifications and deletions to existing software for whatever reason; in CPM 1.2 this was referred to as ‘Maintenance’ work, and in CPM 1.3 as ‘Enhancements’)

Input, process and output functionality which is part of or affected by a Change project is identified and categorised according to the same rules as any other functionality. Only the rules for adding up the functionality to get the total sizes associated with the Change differ from the normal rules.

For each logical transaction required to be added, or to be changed or deleted for the Change, identify the affected input DET’s, output DET’s, and entity references. Categorise each affected component as added, changed or deleted.

The functional size of Change work (i.e. the work output of the project)

= added + changed + deleted functionality.

The size of the software after the Change

= the size of the software before the Change + added - deleted functionality.

The steps for measuring a Change to an existing application are as follows.

Step 1 - Isolate Logical Functionality

Step 2 - Identify and Count New Logical Transactions Required

Step 3 - Identify and Count Existing Logical Transactions No Longer required

Step 4 - Identify the Existing Logical Transactions Affected

Step 5 - Count the Individual Inputs, Outputs and Entity References Affected

Step 6 - Calculate the Functional Size of the Change, and of the changed software

Step 7 (Optional) - Determine the Technical Complexity Adjustment

Step 1 - Isolate Logical Functionality

Identify the logical task elements of the Change project being measured, then identify the logical transactions involved. They may not be specifically stated in the requirements document. It is the responsibility of the analyst to identify them from the available information.

For example - Business Requirement - "Client address information to include postcode"

Work element - capture Postcode in all new addresses, and updates to existing records

Transaction - Enter new client details

Transaction - Update client details

Work Element - Prints involving addresses including postcode

Transaction - Produce reminder letter 1

Transaction - Produce policy schedule.

The collection of all the above elements defines the scope of the project being measured and will identify the new transactions needed to satisfy the requirement, the transactions that are no longer needed and the transactions that already exist but must be modified in some way.

Step 2 - Identify and Count New Logical Transactions Required

From the scope defined in Step 1, identify and count those transactions which are wholly new.

For example, the user needs to capture details about a new type of insurance policy. This Policy Type did not exist before, it does now, and it satisfies a separate and unique business need. A new transaction is therefore required.

In contrast, in an application that already holds information about clients, the addition of Postcode to a client's address does not need a new transaction. The facilities to capture and update addresses were already included in the application; all that has been added is one input type to an existing transaction.

Step 3 - Identify and Count Existing Logical Transactions No Longer Required

From the scope defined in Step 1, identify and count those transactions which must be entirely removed from the application.

For example, the user no longer requires a monthly report of new sales leads, as this information now forms part of a monthly sales performance report. Therefore, the transaction which used to produce the report is to be deleted from the application.

Step 4 - Identify the Existing Logical Transactions Affected

A transaction is affected by the project if an input data element type, an output data element type or an entity reference contained within that transaction has changed as the result of the Change work. A change includes the addition, deletion or modification of any input DET, output DET or entity reference, for whatever reason.

For example, the addition of Postcode to the client's address information affects transactions which capture new clients' addresses, update existing addresses, and print addresses. Requirements for modifications to position or formatting of input and output DET's are considered as changes.

Any alteration of the business rules is also regarded as a change, even if the actual counts of input or output data element types and of entity types referenced do not change. For example, if the business requires a change of validation of an input data element type range from £100 - £499, to £500 - £999, then for function point counting purposes this is counted as a change of one input data element type, even though no new data element types are introduced and no existing data element types are deleted.

It is not essential that there be a change in either an input or output component. A change to an entity reference alone is possible. For example, the user might wish to keep a record of aggregated stock sales , instead of recording the items sold by each salesperson individually. An input, "Quantity Sold", was previously an attribute of one entity ("Salesperson") and is to become an attribute of a different entity ("Stock"). The input is unchanged, but one entity reference, to Salesperson, has been deleted and one, to Stock, has been added, so two entity references are affected by the change.

Step 5 - Count the Individual Inputs, Outputs and Entity References Affected

This step is carried out only for transactions affected by the Change project.

For each logical transaction affected by the changes, count the number of individual inputs, outputs and entity references which are affected: that is, added, changed, or deleted.

For example, in the transaction "Add New Client", the inputs might be Name, Address and Telephone Number. The Address input has changed, by having a Postcode added to it. Telephone Number is a new input: total count is thus 2 input data element types. The input "Name" has not been changed; therefore this input should not be counted, even though it is part of the transaction affected.

Analysis involves recording, for each individually affected input, output and entity reference, whether it has been added, modified or deleted as a result of the change.

Step 6 - Calculate the Functional Size of the Change project and of the enhanced software

See formulae given above.

Step 7 (Optional) - Determine the Technical Complexity Adjustment

(N. B. The TCA is now no longer recommended as part of the MkII FP size.)

If the TCA has changed in the course of the Changes then there will be 2 TCA figures. The old TCA should be applied to the Functional Size for deleted transactions. The new TCA should be applied to the Functional Size for new and amended transactions. This subtlety may be regarded as somewhat of a false refinement. In any case the TCA is frequently unaffected by changes to an application.

5.4 Changes to make software Year 2000 compliant

In principle the MkII FP method may be used to size Changes needed to make software Year 2000 compliant, for estimating purposes. Users are strongly recommended to carry out their own calibration of the method in their own environments for this purpose. Note that modifications for Y2000 compliance will not change the Functional Size of an item of software

5.5 Counting Application Packages

The following assumes that we are dealing with packages brought in by the developers, which may be tailored, enhanced and complemented by bespoke software before being delivered to the users. This section is concerned with *application* packages, i.e. vertical market software and not general purpose software like spreadsheets, memory management, communication packages, operating system utilities, etc.

In all stages of the life cycle there is likely to be a need to distinguish between functionality obtained from the package and bespoke functionality, for which separate counts and baselines will be maintained.

A general observation on the accuracy and repeatability of counts: a lower level of accuracy will frequently be sufficient, although care will be needed where there are contractual considerations. The reasons are (i) that the requirements tend not to be known in detail and (ii) that the size of the package component of the application is a much less reliable indicator of the likely size of the system development project than it can be for a conventional bespoke development. For guidance on approximate function point counts see Section 5.2 of the Counting Practices Manual.

The earliest stage at which a FP count may be required is typically at the package evaluation stage. A possible approach is to size the available packages to compare against a breakdown (transaction by transaction) of the function point count of the user's requirements specification. This would reveal what percentage of the total functionality required would be satisfied by the various packages, and therefore what percentage of the functionality would need to be delivered by bespoke development or Change of the base package.

From this point it is essential to include only function points relating to functionality required by the user. The list of functions provided by the package will frequently be listed in the table of contents of the user guide. Ticking those functions believed to correspond to user requirements should enable one to abstract the part of the package that is required. (Be very wary of FP counts provided by the package supplier - these may bear little relation to a Mk II FP count based on the Logical Transactions of the user's requirements.)

The principles involved in deciding what should be included in the count are the same for application packages as for other bought-in or reused functionality. The general rule is that the FP's should be counted if they were (to quote Symons paraphrasing Albrecht) "selected, maybe modified, integrated, tested, or installed by the project team". Package functionality that is irrelevant to the application needed by the user should be ignored in the FP count.

There will be the following categories of components that could or should be separately sized:

(a) transactions from the package that do not require modification

(b) additional transactions that have to be created by the project team

(c) Changes required to the package (the logical sum of additions, changes and deletions)

(d) the net change (an increase or decrease) in the overall application size as a result of (c)

(e) transactions from the package requiring deletion

The solution delivered to the customer = (a) + (b) + (d) - (e).

The size of the software development aspect of the project = (b) + (c) + (e).

The size maintained of a package-based system depends on the manner in which the package component is supported If, for example, a proportion of the system is unmodified and maintained by the third-party supplier then that part of the system should not be counted as part of the size of the maintenance workload of the in-house support team.

End user report generators and query tools introduce further considerations. Given the growing importance of such tools they should not be left out of any measurement programme. There are several possible ways of approaching the sizing of such tools. The solution chosen should be auditable and seen to be fair by both developers (who will be installing and supporting it) and the end users.

The ideal way is to count the sizes of the transactions resulting in reports produced by or for the users, perhaps summing these over a period, and dividing by the effort involved to get an average performance measure. This may, however, require a lot of work to record all the FP counts and time in relation to the value of the information obtained. A sampling approach over a trial period may therefore give an adequate answer to illustrate the benefit of the tool.

Other possibilities have to be examined therefore, including (neither of these is particularly satisfactory):

- consider the use to be made of such a tool - in other words ascribe a size to it equal to the size of all the reports that the users actually produce
- count one single report transaction, whose size will be equal to the sum of all possible inputs, all entity references and all possible outputs - in other words count all the data items, but only once.

The problem is that these report generators and query languages are *tools*, not applications. (It is the use of these tools which is an application, which can be sized with MkII FPA.) Tools are function-rich, whereas business applications, for which MkII FPA was designed, are data-rich. A tool has a functional size, dependent on the functions it provides, but existing FPA methods do not adequately deal with the types of functions found in tools.

6

Calculating the Adjusted Size (Optional)

The MkII FPA method measures the functional size of the application as seen by the user. As well as the functional size, there is often a need to take into account the technical complexity and certain quality requirements of the application. These are sometimes referred to as the 'non-functional requirements'. One common approach is multiply the Functional Size by the Technical Complexity Adjustment, the result is called the Adjusted Size. Note that measurement is not a measure of functional size according to ISO/IEC 14143-1:1998. The approach is no longer recommended by UKSMA, however this Chapter has been retained to maintain continuity with previous versions of the method. This Chapter will be removed completely in the next major reissue of the CPM.

This approach attempts to measure the influence on the application of each of 19 (or more) technical characteristics on a scale of 0 to 5. (These technical characteristics are listed in Appendix 1 to this manual.)

The sum for all characteristics is used to compute the factor to be applied. This factor is known as the Technical Complexity Adjustment (TCA) .

If the total of the scores for each of the 19 (or more) characteristics is called the Total Degrees of Influence (TDI), then:

$$TCA = (TDI * C) + 0.65$$

where the current industry average value of C is 0.005.

The TCA can thus vary between 0.65 and 1.15 (if all the characteristics have the minimum or maximum influence, respectively).

The adjusted size calculation is expressed as:

$$AS = FS*TCA$$

where

AS = Adjusted Size

FS = Functional Size

TCA = Technical Complexity Adjustment

NB The Technical Complexity Adjustment (or the 'Value Adjustment Factor' as it is known in the IFPUG method), was introduced originally by Albrecht, then modified only slightly for the MK II FP method. Recent statistical analysis suggests that neither the VAF nor the TCA represent well the influence on size of the various characteristics they try to take into account. The TCA definition is included here for continuity with previous versions, but it would be preferable for now when sizing applications within a single technical environment (where the TCA is likely to be constant) to ignore it altogether. Where size comparisons are needed across applications built with very different technical and quality requirements, it may be preferable to use a locally calculated adjustment until such time as a revised TCA is demonstrated to be valid.

7

Measuring Effort

Organisations have many different ways of measuring effort. In order to satisfactorily measure productivity, (see Chapter 8 on productivity measurement), it is essential to have a consistent measure of effort.

Some organisations use person-days, and some use person-months to measure effort. Both of these measures can have various definitions, dependent on the number of practical work-hours in a month, or in a day. It is more likely that we will get consistent measures across different groups and time-recording systems if we use 'work hours' as the unit of measure for effort.

Given that we will use work-hours as our unit of measurement, there are four further elements of effort we have to define:

- When does a project start?
- When does it end?
- Whose time do we include?
- What time do we include?

7.1 Project Start

A project starts when it has been agreed and approved that work should be undertaken to deliver an IS solution. This is usually after a Feasibility Study, or Project Investigation has been carried out.

The clock starts when one or more staff have been allocated to the project, and start working on it.

7.2 Project End

The measurement of a development or enhancement project effort ends, when the application is made "live" by moving it into the production environment for the first time, and the users make use of it.

7.3 Whose time included?

The project effort should include that of staff who are clearly allocated to the project and who work according to the project requirements. The time of a user being interviewed, for instance, would not be included, whereas the time of a user allocated for a significant amount of time to the project team would be included. (‘Significant’ is of course a vague criterion; the first criterion above, that the user is simply ‘clearly allocated...........project requirements’, is more precise, and to be preferred.)

7.4 What time is included?

The work effort should be measured in units of a "productive" or "net" work-hours, which is defined as:

‘One hour of work by one person, including normal personal breaks, but excluding major breaks, such as for lunch’.

Productive work-hours therefore excludes

- time away from work (vacation, public holidays, absence through illness, etc.).
- time at work, but away from the project (receiving education not specific to the project, at office-wide meetings, on activities related to other projects, etc.).

Where effort is not recorded systematically, it may be possible to derive project effort for a member of a project team from the elapsed time the team member spends on the project.

We can derive "net" or "productive" work-hours from "gross" or "employed" work-hours if we have an average utilisation, that is, we know the average percentage productive work-hours across the year. For example, suppose the employment is for 52 weeks per year at 40 hours per week, i.e. 2080 gross work-hours/year. If the average utilisation across the year is 75%, then the average available effort is 1560 work-hours per year, i.e. 0.75 * 2080.

7.5 Project duration

Project duration is the calendar time from project start to project end.

8

Measuring Productivity and Other Aspects of Performance

8.1 Development Productivity

Productivity is defined in terms of what you get out divided by what you put in, that is:

Productivity = Output / Input

Examples often quoted include Number of Cars produced per worker per year in the car industry, or Amount of Profit generated per square foot per annum of retail space in the supermarket business.

In the Information Systems world, our output, what we produce, is application software, and what we put in, our input, is effort. The definition of effort has been covered in Chapter 7, and the size of an application produced, the output, is represented by the FS (FS).

Application development productivity is therefore defined as:

Productivity = FS/Effort

As we discussed in Chapter 7, effort is best given in work-hours, therefore, productivity is measured in Function Points per Work-Hour.

8.2 Change Productivity

As well as development productivity, productivity of change projects can also be measured. In this instance the work-output is the size of the change work carried out, as defined in Rule 2.5 of Chapter 2, and elaborated in Section 5.3.

Note that more refined measures of change productivity are possible which would take into account, for example, that the productivity of adding, changing, or deleting transactions may not be equal.

8.3 Maintenance and Support Productivity

Support work, i.e. maintaining the application according to the existing functional specification, carrying out perfective maintenance and supporting users, can be measured by considering the size of the application being maintained and the effort required, i.e.

Application Size / Effort per period

8.4 Measuring and Understanding Performance in Software Activities: The Wider Issues

The general subject of measuring performance in software development, change and maintenance and support activities is beyond the scope of this manual, but a few words of elaboration are necessary.

Productivity, defined as Size / Effort, is the most commonly quoted performance measure, but it is just one aspect of overall performance in software activities, which really have three main 'dimensions', namely, effort, time and quality. To gain a fuller picture of performance therefore, we need to measure not only productivity, but also other performance measures such as:

- Delivery Rate (= FS / Elapsed Time)
- Quality measures such as Defect Density (= no. of defects found per period / FS)

Depending on the business objectives, it is quite possible that speed of delivery could be more important than productivity. For some applications, very high quality is of paramount importance.

To understand why performance measures vary across different application projects, we need not only the above performance measures, but also measures or assessments of many factors which may be the cause of the observed variations, such as the experience of the project team, the methods and tools used, the stability of the user requirements, etc.

For example, the technical environment for a 4GL or PC development is likely to be more productive than that for a 3GL environment. Vague and shifting requirements is a common cause of reduced productivity.

Change (or 'Enhancement') projects are likely to be less productive than development projects because of the need to understand and test the application being changed, and not just the changed part. The degree of productivity loss will depend on, for example, the available tools and documentation.

Caution: FPA is a powerful normalising factor for generating measures of productivity, quality and other performance factors. All the authorities, including the major software services providers, concur that the results of FPA should be used to provide *process* measures, not *individual* productivity measures capable of use for staff appraisal.

(Blank page)

9

Estimating Effort using MkII FPA

Determining the size of an item of application software in MkII FP's from a statement of its requirements or from its functional specification, can be a valuable first step in a process of estimating the effort and time needed for its development. A full process is given in the MkII FP Estimating method (see Bibliography).

In outline this process is as follows.

1 Estimate the software size, in MkII FP's

2 Determine the 'normative' effort and elapsed time for the project, using historical relationships of development productivity (size/effort) and delivery rate (size/elapsed time) versus software size, for the technology environment to be used ('Normative' means that which we would expect from past experience.)

3 Break down the normative effort and time so as to obtain their distribution over the project phases, using historical distributions of development effort and time, for the technology environment to be used

4 Review the effort and time associated with each phase for the risk involved, and any other factors which might speed up or slow down the process by comparison with the normative effort and time, and adjust the effort and time appropriately

5 Finally, use a historically established trade-off relationship between effort and time, or other knowledge, to allow for any delivery date or staffing constraints, to arrive at the final best estimate for effort and time. (This step might, alternatively, be carried out after step 2.)

The accuracy of this process can be improved by local calibration, that is using weights in the MkII formulae that are specific to the technology which will be used to develop the software. When the MkII formulae are used in this way with locally calibrated weights, the weights can be adjusted so that the formulae predict the software size in units of *standard-hours* needed to develop the software ('standard' in the local environment).

Note that for consistency with the International Standard ISO/IEC 14143-1:1998 on Functional Size Measurement (see Bibliography), software sizes obtained using the MkII formulae, but with local weights (rather than the industry-standard weights) .1) the result shall not be referred to as the functional size as measured using the MkII method.

2) The result shall be referred to using the identification of the local customisation method.

10

Glossary of MkII FPA Terms

Adjusted Size	A size based on the Functional Size multiplied by the Technical Complexity Adjustment. (This measure does not represent Functional Size)
Albrecht 1984	Original document of the function point concept, written by Alan Albrecht in November 1984.
Application	A coherent collection of automated procedures and data supporting a business objective. Frequently a synonym for "system", the word "application" is preferred in this manual since it expresses more precisely the nature of the subject matter of Functional Size Measurement.
Application Function Point Count	A count that provides a measure of the functionality the application provides to the end-user.
Attribute	A unique item of information about an entity. For the purposes of FPA, attributes are generally synonymous with Data Element Types (DET's)

Baseline Function Point Count	Application Function Point Count taken of the functionality at a point in time, from which changes can be measured.
Boundary	The conceptual interface between the software under study and its users. The boundary determines what functions are included in the function point count, and what are excluded.
Change	The modification of an existing application comprising additions, changes and deletions.
Change Project Function Point Count	A count that measures the work-output arising from modifications to an existing application that add, change or delete user functions delivered when the project is complete.
Data Element Type (DET)	A unique user recognisable, non-recursive item of information. The number of DET's is used to determine the input and output size of each Logical Transaction.
Degree of Influence (DI)	A numerical indicator of the impact of each of the 19 (or more) Technical Complexity Adjustment Factors, ranging from 0 (no influence) to 5 (strong influence, throughout). These indicators are used to compute the Technical Complexity Adjustment.
Development	The specification, construction, testing and delivery of a new application or of a discrete addition to an existing application.
Development Project Function Point Count	A count that measures the functionality provided to the end users with the first installation of the software developed when the project is complete.
Enhancement	See 'Change'
Entity (or Data Entity Type)	A fundamental thing of relevance to the user, about which information is kept. An association between entities that has attributes is itself an entity.
Entity Subtype	A subdivision of an Entity Type. A subtype inherits all the attributes and relationships of its parent entity type, and may have additional unique attributes and relationships.
Function Point Analysis (FPA)	A form of Functional Size Measurement (FSM) that measures the functional size of software development, enhancement and maintenance activities associated with Business Applications, from the customer's point of view. The MkII FPA Method complies with ISO/IEC 14143-1:1998 - Functional Size Measurement
Functional Size	(ISO Definition) A size of the software derived by quantifying the Functional User Requirements

Functional Size Measurement (FSM)	(ISO Definition) The process of measuring Functional Size.
Functional User Requirements	(ISO Definition) A sub-set of the user requirements. The Functional User Requirements represent the user practices and procedures that the software must perform to fulfil the user needs. They exclude Quality Requirements and any Technical Requirements
General System Characteristics	IFPUG terminology for the Technical Complexity Adjustment Factors.
Installed Function Point Count	An 'Application Function Point Count' related to a set of installed systems.
IFPUG	The International Function Point User Group, which maintains the definition of the direct descendent of the Albrecht 1984 FPA method.
Interactive	When the user communicates with the computer in a conversational-type manner, e.g. by use of menus, command lines etc.
Logical Transaction	The basic functional component of Mk II FPA. The smallest complete unit of information processing that is meaningful to the end user in the business. It is triggered by an event in the real world of interest to the user, or by a request for information. It comprises an input, process and output component. It must be self-contained and leave the application being counted in a consistent state.
Navigational Aids	Features of software that help the user to navigate around a computer application, e.g. shortcut keys to move through a dialogue faster.
Non-Primary Entity	A data entity-type arrived at by Third Normal Form analysis which is not one of the main entity-types for which the application in question has been built. Non-primary entities have only very few attributes, e.g. code, description - see also System Entity, and Section 4.5 of this CPM.
Primary Entity-Type	In Mk II FPA one of the main entity-types which has the attributes that the application has been designed to process and/or store.
Quality Requirements	(ISO Definition) Any requirements relating to software quality as defined in ISO 9126
Scope Creep	Additional functionality that was not specified in the original requirements but is identified as the requirements and scope are clarified and the functions are defined.

SLOC	Source Lines of Code, the number of lines of programming language code in a program before compilation.
System	See Application.
System Entity	In Mk II FPA a contrivance which 'lumps together' all the non-primary entities of an application.
Technical Complexity Adjustment	A factor which attempts to take into account the influence on application size of Technical and Quality Requirements, which may be used derive the Adjusted Size. Note that if this is done, the result is not the Functional Size. (The TCA is not included within the ISO standard 14143, nor is its use generally recommended).
Technical Complexity Adjustment Factors	The set of 19 factors that are taken into account in the Technical Complexity Adjustment (TCA). Each factor has a Degree of Influence (DI) between 1 and 5.
Technical Requirements	(ISO Definition) Requirements relating to the technology and environment for the development, maintenance, support and execution of the software. Examples of technical requirements include programming languages, testing tools, operating systems, database technology and user interface technologies
User	(ISO Definition) Any person that specifies Functional User Requirements and/or any person or thing that communicates or interacts with the software at any time.

Appendix I

Technical Complexity Adjustment

There are 19 characteristics that contribute to the Technical Complexity Adjustment for an application. This step involves a value judgement representing the 'Degree of Influence' of each of them according to the following scale:

0	=	Not present, no influence
1	=	Insignificant influence
2	=	Moderate influence
3	=	Average influence
4	=	Significant influence
5	=	Strong influence, throughout

This is a general guide. To assist in evaluating the individual characteristics, more specific rules are given below for each of the characteristics.

1. Data Communication - Data and control information use communications facilities

For terminals connected remotely or locally, score as: