

Australian/New Zealand Standard™

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



AS/NZS ISO/IEC 20968:2004

This Joint Australian/New Zealand Standard was prepared by Joint Technical Committee IT-015, Software and Systems Engineering. It was approved on behalf of the Council of Standards Australia on 19 March 2004 and on behalf of the Council of Standards New Zealand on 7 May 2004. It was published on 4 June 2004.

The following are represented on Committee IT-015:

Australian Computer Society
Australian Information Industry Association
Australian Society for Technical Communication (NSW)
Australian Software Metrics Association
Griffith University
New Zealand Organisation for Quality
Quality Society of Australasia
Software Engineering Australia (QLD)
Software Quality Association (ACT)
Software Quality Association (NSW)
Software Verification Research Centre
Sydney SPIN Group (Software Process Improvement Network)
Systems Engineering Society of Australia
The University of New South Wales
University of South Australia
University of Technology, Sydney

Keeping Standards up-to-date

Standards are living documents which reflect progress in science, technology and systems. To maintain their currency, all Standards are periodically reviewed, and new editions are published. Between editions, amendments may be issued. Standards may also be withdrawn. It is important that readers assure themselves they are using a current Standard, which should include any amendments which may have been published since the Standard was purchased.

Detailed information about joint Australian/New Zealand Standards can be found by visiting the Standards Web Shop at www.standards.com.au or Standards New Zealand web site at www.standards.co.nz and looking up the relevant Standard in the on-line catalogue.

Alternatively, both organizations publish an annual printed Catalogue with full details of all current Standards. For more frequent listings or notification of revisions, amendments and withdrawals, Standards Australia and Standards New Zealand offer a number of update options. For information about these services, users should contact their respective national Standards organization.

We also welcome suggestions for improvement in our Standards, and especially encourage readers to notify us immediately of any apparent inaccuracies or ambiguities. Please address your comments to the Chief Executive of either Standards Australia International or Standards New Zealand at the address shown on the back cover.

This Standard was issued in draft form for comment as DR 04079.

STANDARDS AUSTRALIA/STANDARDS NEW ZEALAND

RECONFIRMATION

OF

AS/NZS ISO/IEC 20968:2004

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

RECONFIRMATION NOTICE

Technical Committee IT-015 has reviewed the content of this publication and in accordance with Standards Australia procedures for reconfirmation, it has been determined that the publication is still valid and does not require change.

Certain documents referenced in the publication may have been amended since the original date of publication. Users are advised to ensure that they are using the latest versions of such documents as appropriate, unless advised otherwise in this Reconfirmation Notice.

Approved for reconfirmation in accordance with Standards Australia procedures for reconfirmation on 30 October 2015.

Approved for reconfirmation in New Zealand on behalf of the Standards Council of New Zealand on 18 May 2016.

The following are represented on Technical Committee IT-015:

Australian Computer Society
Australian Society for Technical Communication (NSW)
Charles Sturt University
Department of Defence (Australian Government)
Griffith University
Institute of IT Professionals New Zealand
IT Service Management Forum Australia
La Trobe University
National Association of Testing Authorities Australia
NEHTA
New Zealand Organisation for Quality
NSW Business Chamber
Quantitative Enterprise Software Performance
Systems Engineering Society of Australia
University of Technology, Sydney

NOTES

Australian/New Zealand Standard™

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

First published as AS/NZS ISO/IEC 20968:2004.

COPYRIGHT

© Standards Australia/Standards New Zealand

All rights are reserved. No part of this work may be reproduced or copied in any form or by any means, electronic or mechanical, including photocopying, without the written permission of the publisher.

Jointly published by Standards Australia International Ltd, GPO Box 5420, Sydney, NSW 2001
and Standards New Zealand, Private Bag 2439, Wellington 6020

ISBN 0 7337 6001 5

PREFACE

This Standard was prepared by the Joint Standards Australia/Standards New Zealand Committee IT-015, Software and Systems Engineering.

This Standard is identical with, and has been reproduced from ISO/IEC 20968:2002, *Software engineering—Mk II Function Point Analysis—Counting Practices Manual*.

The objective of this Standard is to provide 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.

The terms ‘normative’ and ‘informative’ are used to define the application of the annex to which they apply. A normative annex is an integral part of a standard, whereas an informative annex is only for information and guidance.

As this Standard is reproduced from an international standard, the following applies:

- (a) Its number appears on the cover and title page while the international standard number appears only on the cover.
- (b) In the source text ‘this International Standard’ should read ‘this Australian/New Zealand Standard’.
- (c) A full point substitutes for a comma when referring to a decimal marker.

None of the normative references in the source document have been adopted as Australian or Australian/New Zealand Standards.

CONTENTS

	<i>Page</i>
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

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.

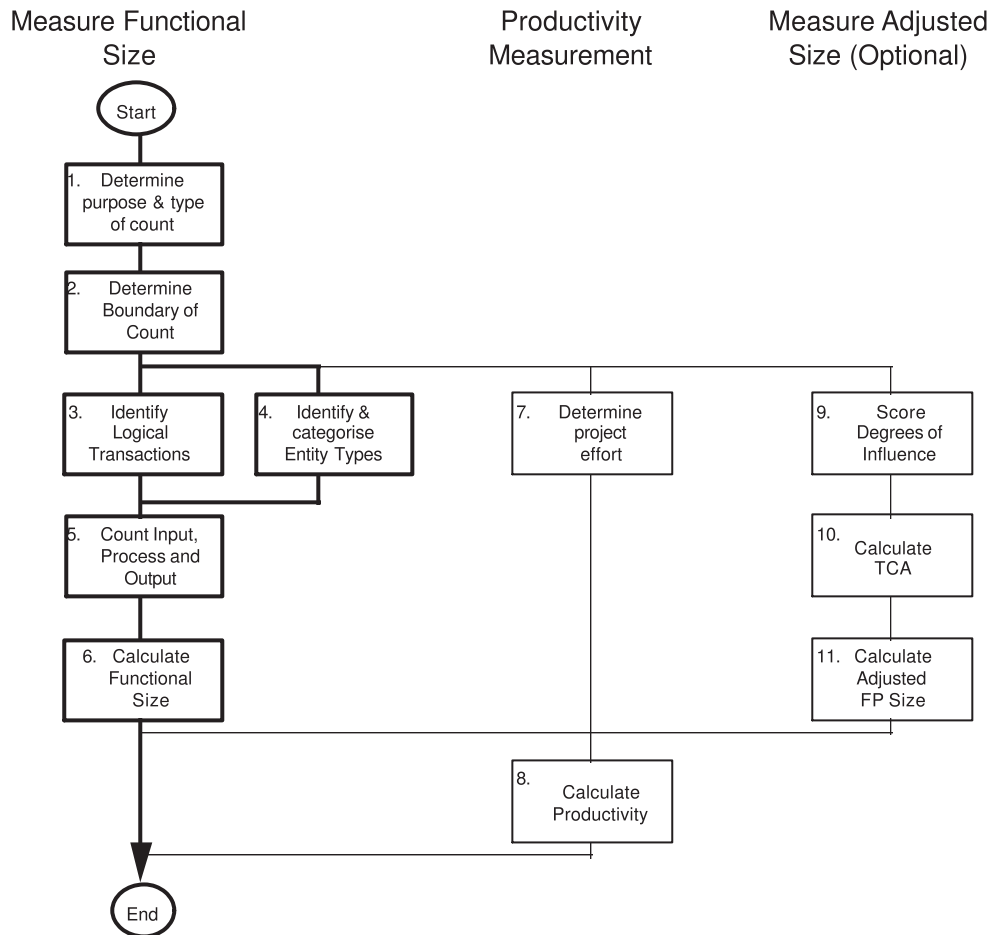
Any function that is provided, but which was not required by the user, is ignored for FPA purposes (Rule 1).

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



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 (N_i), the number of Data Entity Types Referenced (N_e), and the number of Output Data Element Types (N_o).

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 (N_i), the Data Entity Types Referenced (N_e), and the Output Data Element Types (N_o).

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

$$FS = W_i * \sum N_i + W_e * \sum N_e + W_o * \sum N_o,$$

where ' $\sum$ ' 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:

$$W_i = 0.58$$

$$W_e = 1.66$$

$$W_o = 0.26$$

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.

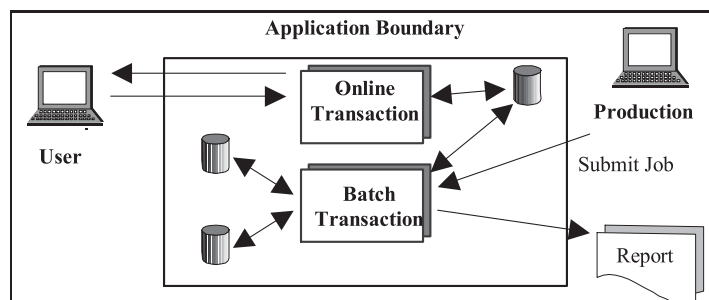


Figure 4.1 - Business user view of application boundary

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 (revali-dated)	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

unique	<p>which itself corresponds to an event in the external world.</p> <p>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.</p>
event of interest	<p>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.</p> <p>Events are:</p> <ul style="list-style-type: none"> • 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. <p>These are the ACID tests – if any candidate ‘event’ fails one or more of these criteria, reject it from the application’s logical model.</p> <p>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.</p>
external world	<p>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.</p>

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.

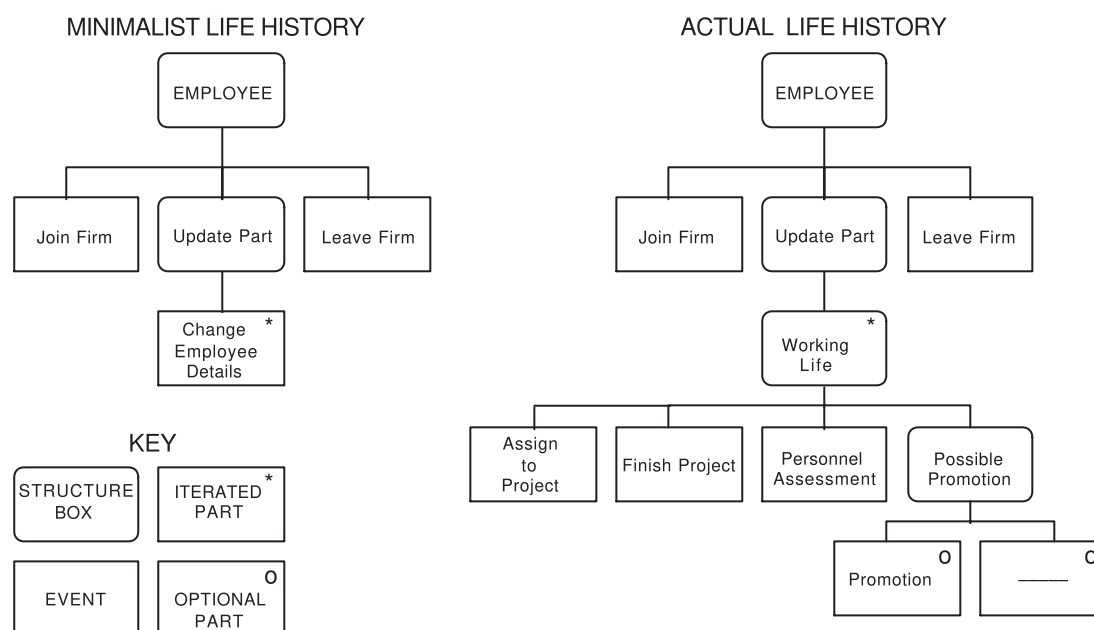


Figure 4.2: Minimalist & More Complex Entity Life Histories

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