The COSMIC Functional Size Measurement Method

Version 3.0.1

Measurement Manual

(The COSMIC Implementation Guide for ISO/IEC 19761: 2003)

May 2009

Acknowledgements

COSMIC Core Team Authors, Version 2.0[1] (alphabetical order)

Alain Abran, École de technologie supérieure – Université du Québec,

Jean-Marc Desharnais, Software Engineering Laboratory in Applied Metrics - SELAM,

Serge Oligny, Bell Canada,

Denis St-Pierre, DSA Consulting Inc.,

Charles Symons, Software Measurement Services Ltd.

Version 2.0 reviewers 1998/1999 (alphabetical order)		
Moritsugu Araki, JECS Systems Research, Japan	Thomas Fetcke, Germany	Patrice Nolin, Hydro Québec, Canada
Fred Bootsma, Nortel, Canada	Eric Foltin, University of Magdeburg, Germany	Marie O'Neill, Software Management Methods, Ireland
Denis Bourdeau, Bell Canada, Canada	Anna Franco, CRSSM, Canada	Jolijn Onvlee, The Netherlands *
Pierre Bourque, , ÉCole de Technologie supérieure, Canada	Paul Goodman, Software Measurement Services,United Kingdom	Laura Primera, UQAM, Canada
Gunter Guerhen, Bürhen & Partner, Germany	Nihal Kececi, University of Maryland, United States	Paul Radford, Charismatek, Australia
Sylvain Clermont, Hydro Québec, Canada	Robyn Lawrie, Australia	Eberhard Rudolph, Germany
David Déry, CGI, Canada	Ghislain Lévesque, UQAM, Canada	Grant Rule, Software Measurement Services, United Kingdom*
Gilles Desoblins, France	Roberto Meli, Data Processing Organization, Italy	Richard Stutzke, Science Applications Int'l Corporation, United States
Martin D'Souza, Total Metrics, Australia	Pam Morris, Total Metrics, Australia*	Ilionar Sylva, UQAM, Canada
Reiner Dumke, University of Magdeburg, Germany	Risto Nevalainen, Software Technology Transfer Finland, Finland *	Vinh T. Ho, UQAM, Vietnam
Peter Fagg, United Kingdom	Jin Ng, Hmaster, Australia	

* Founding members of the COSMIC Core Team, along with the COSMIC-FFP authors

[1] Version 2.0 was the first publicly available version of the COSMIC-FFP method, as it was first known

Version 3.0 Reviewers 2006/07 (alphabetical order)		
Alain Abran, École de Technologie Supérieure, Université du Québec, Canada	Jean-Marc Desharnais, Software Engineering Lab in Applied Metrics – SELAM, Canada	Arlan Lesterhuis*, Sogeti, The Netherlands
Bernard Londeix, Telmaco, United Kingdom	Roberto Meli, Data Processing Organization, Italy	Pam Morris, Total Metrics, Australia
Serge Oligny, Bell Canada	Marie O'Neill, Software Management Methods, Ireland	Tony Rollo, Software Measurement Services, United Kingdom
Grant Rule, Software Measurement Services, United Kingdom	Luca Santillo, Agile Metrics, Italy	Charles Symons*, United Kingdom
Hannu Toivonen, Nokia Siemens Networks, Finland	Frank Vogelezang, Sogeti, The Netherlands	

* Editors of versions 3.0 and 3.0.1 of the COSMIC method

A public domain version of the COSMIC Measurement Manual and other technical reports, including translations into other languages can be found on the Web at www.gelog.etsmtl.ca/cosmic-ffp

Version Control

The following table gives the history of the versions of this document

DATE	REVIEWER(S)	Modifications / Additions
99-03-31	Serge Oligny	First draft, issued for comments to reviewers
99-07-31	See Acknowledgements	Revised, including comments from reviewers
99-10-06	See Acknowledgements	Revised, including comments from IWSM '99 workshop2
99-10-29	COSMIC Core team	Revised, final comments before publishing "field trial" version 2.0.
01-05-01	COSMIC Core Team	Revised for conformity to ISO/IEC 14143-1: 1998 + clarifications on measurement rules to version 2.1.
03-01-31	COSMIC Measurement Practices Committee	Revised for conformity to ISO/IEC FDIS 19761: 2002 + further clarifications on measurement rules to version 2.2
07-09-01	COSMIC Measurement Practices Committee	Revised for further clarifications and additions to the measurement rules to version 3.0, particularly in the area of the Measurement Strategy phase. The method name was changed from the 'COSMIC-FFP method' to the 'COSMIC method'. In upgrading to v3.0 from v2.2, parts of the 'Measurement Manual' v2.2 were separated into other documents – see the Foreword below and Appendix D
09-05-01	COSMIC Measurement Practices Committee	Version 3.0 revised to v3.0.1 to make minor editorial improvements and clarifications, and to distinguish examples more clearly. This version also incorporates the changes proposed in Method Update Bulletins 3, 4 and 5. See Appendix D for details of these changes

[2] Proceedings of the International Workshop on Software Measurement IWSM '99, Lac Supérieur, Québec, Canada, September 8-10 1999. See http://www.gelog.etsmtl.ca/iwsm99/index2.html for details.

Foreword

The purpose of the COSMIC method is to provide a standardized method of measuring a functional size of software from the functional domains commonly referred to as 'business application' (or 'MIS') software and 'real-time' software.

The COSMIC method was accepted by ISO/IEC JTC1 SC7 in December 2002 as International Standard ISO/IEC 19761 'Software Engineering – COSMIC-FFP – A functional size measurement method' (hereafter referred to as 'ISO/IEC 19761').

For clarity, ISO/IEC 19761 contains the fundamental normative definitions and rules of the method. The purpose of the Measurement Manual is not only to provide these rules and definitions, but also to provide further explanation and many more examples in order to help measurers to fully understand and to apply the method. However, as more and more experience has been gained with the method, it has been found valuable to add to the rules and examples and even to refine the definitions of some of the underlying concepts. The Common Software Measurement International Consortium (COSMIC) envisages that these additions and refinements will be submitted to ISO for inclusion in ISO/IEC 19761 when it is due for revision in 2007/08.

This 'Measurement Manual' is one of four COSMIC documents that define version 3.0 of the method. The other three are:

- 'Documentation Overview and Glossary of Terms' (The Glossary defines all terms that are common to all COSMIC documents. This document also describes other available supporting documents such as case studies and domain-specific guidelines.)
- 'Method Overview'
- 'Advanced and Related Topics' (This document will deal in more detail with the task of ensuring the comparability of size measurements and will include chapters on early or rapid approximate sizing and on convertibility of measurements which previously appeared in version 2.2 of the Measurement Manual.)

Readers who are new to functional size measurement ('FSM') or who are familiar with another FSM method are strongly advised to read the 'Method Overview' document before reading this Measurement Manual.

Main changes for version 3.0 of the COSMIC Method

The change in designation of the version of the COSMIC method from 2.2 to 3.0 indicates that this version represents a significant advance over the previous version. In producing this version 3.0 of the COSMIC method from the previous version defined in the Measurement Manual version 2.2, the main changes are as follows.

- In order to make the COSMIC method documentation more 'user-friendly', version 3.0 of the method is now defined in four separate documents as listed above.
- The proposals from the two Method Update Bulletins that were published since the last version 2.2 of the Measurement Manual have been incorporated. These are MUB 1 'Proposed Improvements to the Definition and Characteristics of a software 'Layer' and MUB 2 'Proposed Improvement to the Definition of an 'object of interest'. (Version 3.0.1 incorporates three more MUB's – see Appendix D.)
- A 'Measurement Strategy' phase has been separately defined as the first phase of the measurement process. The strategy phase is also enhanced by introducing guidance on considering the concept of the 'level of granularity' of the Functional User Requirements of the software to be measured, to help ensure comparability of measurements across different pieces of software.

- Experience has shown that the concepts of the 'End User' and the 'Developer' Measurement Viewpoints, which were introduced into the Measurement Manual in version 2.2 can be replaced by the simpler concept of a 'functional user'. The latter can be loosely defined as a sender or intended recipient of data in the Functional User Requirements (FUR) of the software to be measured. All size measurements on a piece of software are then made of the functionality as provided to the software's functional users, as identified in its FUR.
- The name of the unit of measure of the method has been changed from 'COSMIC functional size unit' (abbreviated as 'Cfsu') to 'COSMIC Function Point' (abbreviated as 'CFP'). This change has been made for ease of reading and pronunciation, and to bring the method into conformity with other 'Function Point' methods. As a further simplification, the name of the method has been changed from 'COSMIC-FFP' to 'COSMIC'.
- The glossary has been updated and enhanced to improve readability and has been moved to the new 'Documentation Overview and Glossary of Terms' document.
- Some material, notably on data analysis concepts has been removed. It is now included in the 'Guideline for Sizing Business Application Software using COSMIC', since it is very specific to that one domain and is not specific to the COSMIC method.
- Many editorial improvements and additions have been made to improve consistency of terminology and to improve understanding. Amongst these, a more consistent distinction has been made between 'principles' and 'rules' of the method by adopting a convention from the world of accounting that 'rules exist to help apply definitions and principles'. Both principles and rules should be regarded as mandatory. Hence most examples have been removed from the statements of principles and rules to the body text.

All these changes are summarized in Appendix D.

Readers familiar with version 2.2 of the Measurement Manual will see the greatest number of changes in version 3.0 in the newly-separated 'Measurement Strategy' phase.

Consequences of the main changes on existing size measurements

The original principles of the COSMIC method have remained unchanged since they were first published in the first draft of the Measurement Manual in 1999, in spite of the refinements and additions needed to produce the International Standard and when producing versions 2.1, 2.2, 3.0 and this latest version 3.0.1 of the Measurement Manual.

Functional sizes measured according to the principles and rules of versions 3.0 and 3.0.1 of the Measurement Manual may differ from sizes measured using earlier versions only because the new rules intend to be more precise. Hence measurers have less discretion for personal interpretation of the rules than was possible with earlier versions. The changes in the area of the Measurement Strategy and the change to the name of the unit of measure result in trivial differences in rules for reporting measurement results compared with earlier versions.

Explanation for the main changes from version 2.2 to version 3.0 of the Measurement Manual

We must first emphasize that the COSMIC unit of measure, the 'Cfsu' (now renamed the 'CFP'), remains unchanged from when it was introduced in the first public version of the COSMIC-FFP Measurement Manual in the year 1999. This is the COSMIC equivalent of, for example, a standard unit of length such as the meter. But the functional size of a piece of software can be measured in many ways, and it is sometimes a problem, for any Functional Size Measurement (FSM) Method, to answer the question of 'what size should we measure?'

The first problem is that we know that any piece of software provides functionality to various types of 'users', where a user is defined in the terminology of the ISO/IEC 14143/1 standard ('Principles of FSM') essentially as 'anything that interacts with the software being measured'. It follows that the functional size of a piece of software depends on who or what is defined as its user(s).

Let us consider the example of the application software of a mobile phone. If we accept the ISO definition of 'user' literally, the users of a mobile phone application could include all of the following: a human who presses the buttons; the hardware devices (e.g. the screen, keys, etc) that interact with the application; the operating system that supports the application; separate peer pieces of software that interact with the application being measured. All four types of users require different functionality (hence the functional size differs depending on who or what is defined as the user). So how, if we are given a particular functional size, do we know who or what were the user(s), i.e. what functionality was measured?

It was this issue that initially led the COSMIC Measurement Practices Committee ('MPC') to introduce in version 2.2 of the Measurement Manual the concepts of the 'End User' and 'Developer' Measurement Viewpoints. However, experience has shown that these definitions, particularly of the Developer Measurement Viewpoint, were not general enough to help define all measurement needs. The MPC has concluded that the correct and most general approach is to define the concept of a 'functional user' and that the functional size changes depending on who or what is defined as the functional user. The identification of the functional users depends on the purpose of the measurement and the functional users should normally be identifiable in the Functional User Requirements (FUR) of the software to be measured. The idea of defining specific 'Measurement Viewpoints' is not then needed anymore.

The second problem is that we know that FUR evolve as a project progresses and, depending on how the size is measured, their measured size may appear to grow. The first version of the FUR for a new piece of software may be specified at a 'high level'. As the project progresses and requirements are elaborated in more detail, the FUR are specified in more detail, or at a 'lower level' and their size may appear to increase. We distinguish these different levels of detail as 'levels of granularity'.

So the problem that has to be addressed now is how can we be sure that two different measurements have been made at the same level of granularity? Versions 3.0 and 3.0.1 of the Measurement Manual provide recommendations on this topic, which is especially important when sizes are measured early in the life of a project when the FUR are still evolving. The topic becomes critical when sizes are used for performance measurements that must be compared from different sources such as in benchmarking exercises.

It is important to emphasize that these new concepts of 'functional user' and 'level of granularity' and the associated processes for determining them that have been introduced into the Measurement Strategy need not be unique to the COSMIC method. They are applicable to all Functional Size Measurement (FSM) methods. Because the COSMIC method is based on sound software engineering principles and is applicable to a wider range of software domains than '1st generation' FSM methods, the problem of properly defining 'what size should we measure?' has been recognized and a solution has been found.

Most measurers using COSMIC, where the purpose of the measurement is performance-related, (e.g. for estimating, benchmarking, etc), will not need to spend any time on the identification of functional users or on the level of granularity at which to measure, since they will be obvious. But for measurements where the choices may not be obvious, new material in the 'Measurement Strategy Phase' of the Measurement Manual and the chapter on ensuring the comparability of size measurements in the 'Advanced and Related Topics' document, discuss the factors to consider in greater depth.

The COSMIC Measurement Practices Committee

May 2009

Table of Contents

1 INTRODUCTION 10

1.1 Applicability of the COSMIC method 10

1.1.1 Applicable domains 10

1.1.2 Non-applicable domain 10

1.1.3 Limitations on the factors contributing to functional size 10

1.1.4 Limitations on measuring very small pieces of software 11

1.2 Functional User Requirements 11

1.2.1 Extracting the functional user requirements from software artifacts in practice 11

1.2.2 Deriving the functional user requirements from installed software 12

1.2.3 Extracting or deriving the functional user requirements from software artifacts 12

1.3 The COSMIC Software Context Model 13

1.4 The Generic Software Model 13

1.5 The COSMIC Measurement Process 14

2 THE MEASUREMENT STRATEGY PHASE 16

2.1 Defining the purpose of the measurement 17

2.1.1 The purpose of the measurement – an analogy 17

2.1.2 The importance of the purpose 18

2.2 Defining the scope of the measurement 18

2.2.1 Deriving the scope from the purpose of a measurement 19

2.2.2 Generic types of Scope 20

2.2.3 Levels of decomposition 20

2.2.4 Layers 21

2.2.5 Peer components 23

2.3 Identifying the functional users 24

2.3.1 Functional size varies with the functional user 24

2.3.2 Functional users 25

2.4 Identifying the level of granularity 26

2.4.1 The need for a standard level of granularity 26

2.4.2 Clarification of 'level of granularity' 27

2.4.3 The standard level of granularity 28

2.5 Concluding remarks on the measurement strategy phase 31

3 THE MAPPING PHASE 32

3.1 Applying the Generic Software Model 33

3.2 Identifying functional processes 34

3.2.1 Definitions 34

3.2.2 The approach of identifying functional processes 35

3.2.3 Triggering events and functional processes in the business applications domain 36

3.2.4 Triggering events and functional processes in the real-time applications domain 37

3.2.5 More on separate functional processes 38

3.2.6 The functional processes of peer components 38

3.3 Identifying objects of interest and data groups 38

3.3.1 Definitions and principles 38

3.3.2 About the materialization of a data group 39

3.3.3 About the identification of objects of interest and data groups 40

3.3.4 Data or groups of data that are not candidates for data movements 41

3.3.5 The functional user as object of interest 41

3.4 Identifying data attributes (optional) 41
3.4.1 Definition 41
3.4.2 About the association of data attributes and data groups 42

4 THE MEASUREMENT PHASE 43

4.1 Identifying the data movements 43
4.1.1 Definition of the data movement types 44
4.1.2 Identifying Entries (E) 45
4.1.3 Identifying Exits (X) 46
4.1.4 Identifying Reads (R) 47
4.1.5 Identifying Writes (W) 47
4.1.6 On the data manipulations associated with data movements 48
4.1.7 Data movement uniqueness and possible exceptions 49
4.1.8 When a functional process moves data to or from persistent storage 51
4.1.9 When a functional process requires data from a functional user 53
4.1.10 Control commands 55
4.2 Applying the measurement function 55
4.3 Aggregating measurement results 56
4.3.1 General rules of aggregation 56
4.3.2 More about functional size aggregation 57
4.4 More on measurement of the size of changes to software 58
4.4.1 Modifying functionality 58
4.4.2 Size of the functionally changed software 59
4.5 Extending the COSMIC measurement method 59
4.5.1 Introduction 59
4.5.2 Local extension with complex algorithms 60
4.5.3 Local extension with sub-units of measurement 60

5 MEASUREMENT REPORTING 61

5.1 Labeling 61
5.2 Archiving COSMIC measurement results 62

APPENDIX A - DOCUMENTING A COSMIC SIZE MEASUREMENT 63

APPENDIX B - SUMMARY OF COSMIC METHOD PRINCIPLES 64

APPENDIX C - SUMMARY OF COSMIC METHOD RULES 68

APPENDIX D - HISTORY OF COSMIC METHOD RELEASES 74
From version 2.2 to version 3.0 74
From version 3.0 to version 3.0.1 77

APPENDIX E - COSMIC CHANGE REQUEST AND COMMENT PROCEDURE 79

1

INTRODUCTION

1.1 Applicability of the COSMIC method

1.1.1 Applicable domains

The COSMIC functional size measurement method is designed to be applicable to the functionality of software from the following domains:

- Business application software which is typically needed in support of business administration, such as banking, insurance, accounting, personnel, purchasing, distribution or manufacturing. Such software is often characterized as 'data rich', as it is dominated largely by the need to manage large amounts of data about events in the real world.
- Real-time software, the task of which is to keep up with or control events happening in the real world. Examples would be software for telephone exchanges and message switching, software embedded in devices to control machines such as domestic appliances, elevators, car engines and aircraft, for process control and automatic data acquisition, and software within the operating system of computers.
- Hybrids of the above, as in real-time reservation systems for airlines or hotels for example.

1.1.2 Non-applicable domain

The COSMIC measurement method has not yet been designed to take into account the functionality of mathematically-intensive software, that is, software which is characterized by complex mathematical algorithms or other specialized and complex rules, such as in expert systems, simulation software, self-learning software, weather forecasting systems, etc., or which processes continuous variables such as audio sounds or video images, such as, for instance, in computer game software, musical instruments, etc.

For software with such functionality it is possible, however, to define local extensions to the COSMIC measurement method. The Measurement Manual explains in what contexts such local extensions should be used and provides examples of a local extension. When used, local extensions must be reported according to the conventions presented in the chapter on measurement reporting of this Measurement Manual.

1.1.3 Limitations on the factors contributing to functional size

Further, within its domain of applicability, the COSMIC method of measuring functional size does not attempt to measure all possible aspects of functionality that might be considered to contribute to software 'size'. For example, neither the influence of 'complexity' (however defined), nor the influence of the number of data attributes per data movement on software functional size are captured by this measurement method (for more on the latter, see the Mapping Phase chapter of the Measurement Manual). As described in section 1.1.2, if desired, such aspects of functional size may be accounted for by a local extension to the COSMIC measurement method.

1.1.4 Limitations on measuring very small pieces of software

All functional size measurement methods are based on the assumptions of a simplified model of software functionality that is intended to be reasonable 'on average' for its intended domain of applicability. Caution is therefore needed when measuring, comparing or using (e.g. for estimating) sizes of very small pieces of software, and especially of very small changes to a piece of software, where the 'average' assumption may break down. In the case of the COSMIC method 'very small' means 'a few data movements'.

1.2 Functional User Requirements

The COSMIC measurement method involves applying a set of models, principles, rules and processes to the **Functional User Requirements** (or **'FUR'**) of a given piece of software. The result is a numerical 'value of a quantity' (as defined by ISO) representing the functional size of the piece of software according to the COSMIC method.

Functional sizes measured by the COSMIC method are designed to be independent of any implementation decisions embedded in the operational artifacts of the software to be measured. 'Functionality' is concerned with 'the information processing that the software must perform for its users'.

More specifically, a statement of FUR describes 'what' the software must do for the functional users who are the senders and intended recipients of data to and from the software. A statement of FUR excludes any technical or quality requirements that say 'how' the software must perform. (For the formal definition of FUR, see section 2.2.) Only the FUR are taken into account when measuring a functional size.

1.2.1 Extracting the functional user requirements from software artifacts in practice

In the real world of software development it is rare to find artifacts for the software in which the FUR are clearly distinguished from other types of requirements and are expressed in a form suitable for direct measurement without any need for interpretation. This means that usually the measurer will have to extract the FUR as supplied in or implied in the actual artifacts of the software, before mapping them to the concepts of the COSMIC 'models of software'.

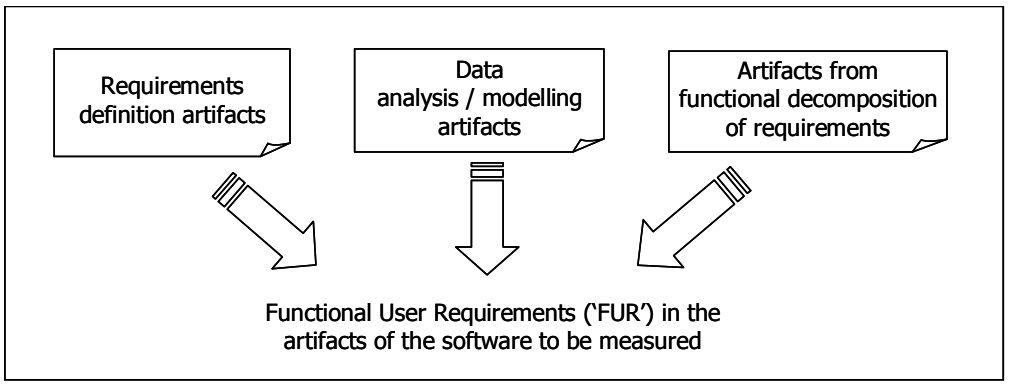


Figure 1.2.1 – COSMIC model of pre-implementation functional user requirements

As illustrated in figure 1.2.1 FUR can be derived from software engineering artifacts that are produced before the software exists. Thus, the functional size of software can be measured prior to its implementation in a computer system.

It is important to note that artifacts that are produced before software is implemented, e.g. during requirements gathering or analysis, may describe the software at different 'levels of granularity' as the requirements evolve – see further section 2.4 below

Note: Functional user requirements may be produced before they are allocated to hardware or software. Since the COSMIC method is aimed at sizing the FUR of a piece of software, only the FUR allocated to the software are measured. However, in principle COSMIC can be applied to FUR before they are allocated to software or to hardware, regardless of the eventual allocation decision. For example, it is straightforward to size the functionality of a pocket calculator using COSMIC without any knowledge of what hardware or software (if any) is involved. However, the assertion that the COSMIC method can be used to size FUR allocated to hardware needs more testing in practice before it can be considered as fully validated without the need for further rules.

1.2.2 Deriving the functional user requirements from installed software

In other circumstances, some existing piece of software may need to be measured without there being any, or with only a few, architecture or design artifacts available, and the FUR might not be documented (e.g. for legacy software). In such circumstances, it is still possible to derive the FUR from the artifacts installed on the computer system even after it has been implemented, as illustrated in figure 1.2.2

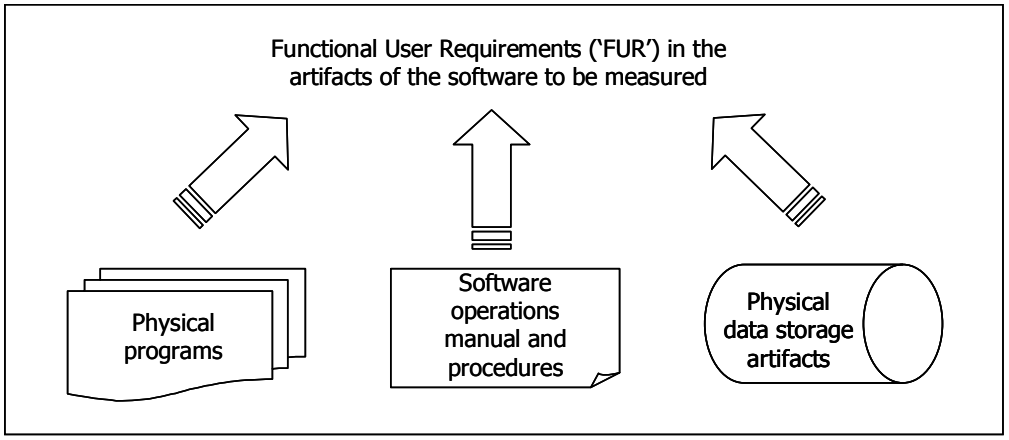


Figure 1.2.2 – COSMIC model of post-implementation functional user requirements

1.2.3 Extracting or deriving the functional user requirements from software artifacts

The process to be used and hence the effort required to extract the FUR from different types of software engineering artifacts or to derive them from installed software and to express them in the form required for measurement by the COSMIC method will obviously vary; these processes cannot be dealt with in the Measurement Manual. The assumption is made that the functional user requirements of the software to be measured either exist or that they can be extracted or derived from its artifacts, in light of the purpose of the measurement.

The Measurement Manual is therefore confined to describing and defining the concepts of the COSMIC software models, i.e. the targets of the extraction or derivation.[3] These concepts are embodied in a set of principles set out in two COSMIC software models – the 'Software Context Model' and the 'Generic Software Model'.

[3] The 'Guideline for Sizing Business Application Software using COSMIC' gives guidance on the mapping from various data analysis and requirements determination methods used in the business application domain to the concepts of COSMIC

These two models are described in the 'Method Overview' document. Readers requiring a general understanding of and justification for the models are referred to the 'Method Overview' document. The models are copied below for convenience and are elaborated in detail in Chapters 2 and 3 of this Measurement Manual respectively.

1.3 The COSMIC Software Context Model

A piece of software to be measured by the COSMIC method must be carefully defined (in the measurement scope) and this definition must take into account its context of any other software and/or hardware with which it interacts. This Software Context Model introduces the principles and concepts needed for this definition.

N.B. Terms that are given in bold when first used in the following sections 1.3 and 1.4 are used with meanings that may be specific to the COSMIC method. For the formal definitions, see the glossary in the 'Documentation Overview and Glossary of Terms' document. All these terms are explained in greater detail in the following chapters 2, 3 and 4

PRINCIPLES – The COSMIC Software Context Model
a) Software is bounded by hardware
b) Software is typically structured into **layers**
c) A layer may contain one or more separate '**peer'** pieces of software and any one piece of software may further consist of separate peer components
d) Any piece of software to be measured, shall be defined by its measurement **scope**, which shall be confined wholly within a single layer
e) The scope of a piece of software to be measured shall depend on the **purpose** of the measurement
f) The **functional users** of a piece of software shall be identified from the functional user requirements of the piece of software to be measured as the senders and/or intended recipients of data
g) A piece of software interacts with its functional users via **data movements** across a **boundary** and the piece of software may move data to and from **persistent storage** within the boundary
h) The FUR of software may be expressed at different **levels of granularity**
i) The level of granularity at which measurements should normally be made is that of the **functional processes**
j) If it is not possible to measure at the level of granularity of the functional processes, then the FUR of the software should be measured by an approximation approach and scaled to the level of granularity of the functional processes[4]

The concepts of the Software Context Model are elaborated in the 'Measurement Strategy' Chapter 2 of this Measurement Manual.

1.4 The Generic Software Model

Having interpreted the FUR of the software to be measured in terms of the Software Context Model, we now apply the Generic Software Model to the FUR to identify the components of the functionality that will be measured. This Generic Software Model assumes that the following general principles hold true for any software that can be measured with the method. (As noted in the glossary, any

[4] The subjects of 'approximate sizing' (i.e. of estimating a size when all the detail is not available to measure an accurate size) and of scaling between different levels of granularity is dealt with in the COSMIC method document 'Advanced and Related Topics'.

functional size measurement method aims to identify 'types' and not 'occurrences' of data or functions. In the text below, the suffix 'type' will therefore be omitted when mentioning COSMIC basic concepts unless it is essential to distinguish 'types' from 'occurrences'.)

PRINCIPLES – The COSMIC Generic Software Model
a) Software receives **input** data from its functional users and produces **output**, and/or another outcome, for the functional users
b) Functional user requirements of a piece of software to be measured can be mapped into unique functional processes
c) Each functional process consists of **sub-processes**
d) A sub-process may be either a data movement or a **data manipulation**
e) Each functional process is triggered by an **Entry** data movement from a functional user which informs the functional process that the functional user has identified an **event**
f) A data movement moves a single **data group**
g) A data group consists of a unique set of **data attributes** that describe a single object of interest
h) There are four types of data movement. An Entry moves a data group into the software from a functional user. An **Exit** moves a data group out of the software to a functional user. A **Write** moves a data group from the software to persistent storage. A **Read** moves a data group from persistent storage to the software
i) A functional process shall include at least one Entry data movement and either a Write or an Exit data movement, that is it shall include a minimum of two data movements
j) As an approximation for measurement purposes, data manipulation sub-processes are not separately measured; the functionality of any data manipulation is assumed to be accounted for by the data movement with which it is associated

The concepts of the Generic Software Model are elaborated in the 'Mapping Phase' Chapter 3 of this Measurement Manual.

When the FUR to be measured have been mapped onto the Generic Software Model, they can be measured using the process of the Measurement Phase (Chapter 4). Measurement results should be reported according to the conventions of 'Measurement Reporting' (Chapter 5).

1.5 The COSMIC Measurement Process

The general COSMIC measurement process consists of three phases:

- the Measurement Strategy, in which the Software Context Model is applied to the software to be measured (Chapter 2)
- the Mapping Phase in which the Generic Software Model is applied to the software to be measured (Chapter 3)
- the Measurement Phase, in which actual size measurements are obtained (Chapter 4)

The result of applying the measurement process to a piece of software is a functional size measure of the Functional User Requirements of the piece of software expressed in 'COSMIC Function Points' (or 'CFP')

The relationship of these three phases of the COSMIC method is shown in Fig. 1.5.

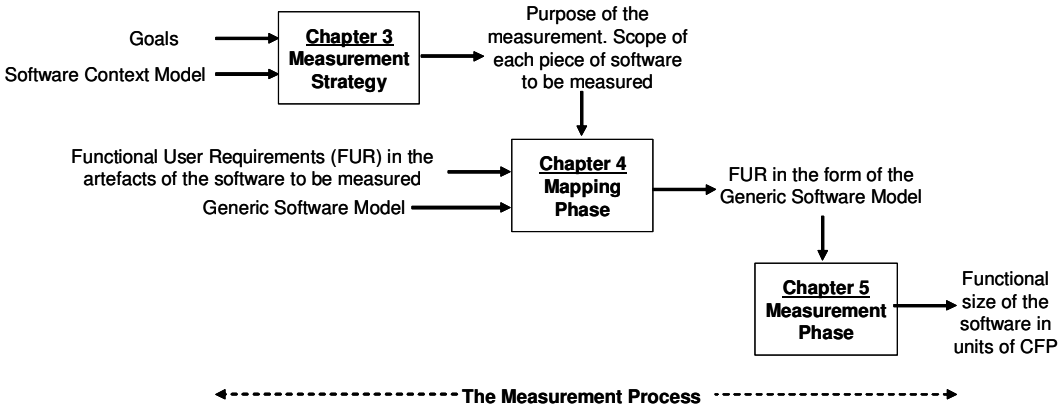


Figure 1.5 – Structure of the COSMIC method

2

THE MEASUREMENT STRATEGY PHASE

This chapter addresses the four key parameters of software functional size measurement that must be considered before actually starting to measure, namely the purpose and scope of the measurement, the identification of functional users and the level of granularity that should be measured. Determining these parameters helps to address the questions of 'which size should be measured?' or, for an existing measurement, 'how should we interpret this measurement?'

It is important to note that these four parameters and related concepts are not specific to the COSMIC functional size measurement (FSM) method, but should be common to all FSM Methods. Other FSM Methods may not distinguish different types of functional users and may not discuss different levels of granularity, etc. It is only the broader applicability and flexibility of the COSMIC method that requires these parameters to be considered more carefully than with other FSM Methods.

It is very important to record the data arising from this Measurement Strategy phase (as listed in section 5.2) when recording the result of any measurement. Failure to define and to record these parameters consistently will lead to measurements that cannot be interpreted reliably and compared, or be used as input for processes such as estimating project effort.

The four sections of this chapter give the formal definitions, principles and rules and some examples for each of the key parameters to help the measurer through the process of determining a Measurement Strategy, as shown in figure 2.0 below.

Each section gives some background explanation of why the key parameter is important, using analogies to show why the parameter is taken for granted in other fields of measurement, and hence must also be considered in the field of software functional size measurement.

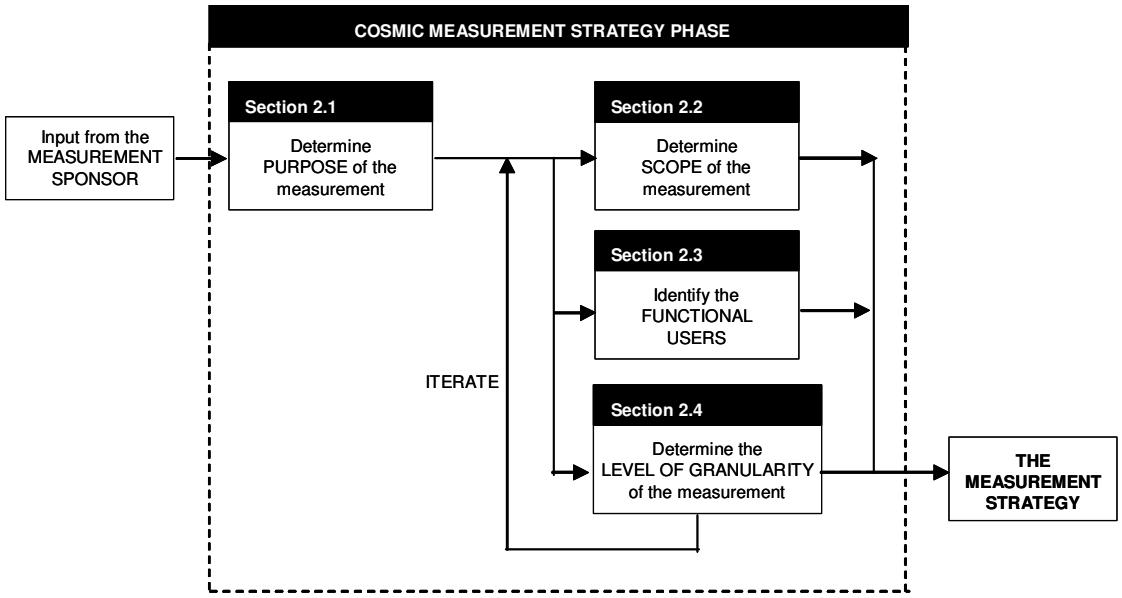


Figure 2.0 - The process of determining a Measurement Strategy

2.1 Defining the purpose of the measurement

The term 'purpose' is used in its normal English meaning.

DEFINITION – Purpose of a measurement
A statement that defines why a measurement is required, and what the result will be used for.

2.1.1 The purpose of the measurement – an analogy

There are many reasons to measure the functional size of software, just as there are many reasons to measure, say, the surface areas of a house. When the purpose is to estimate the cost of a new software development, it might be necessary to measure the functional size of software prior to its development, just as it might be necessary to measure the surface areas of a house prior to its construction. In a different context, e.g. when needing to compare actual with estimated costs, it might be necessary to measure the functional size of software once it has been put into production, just as it might be useful to measure the surface areas of a house after it had been built to check that the actual dimensions conform to the agreed plans. The reason why a measure is taken has an impact, albeit at times subtle, on *what* is being measured, *without affecting* the unit of measure or the measurement principles.

In the house example above, measuring its surface areas prior to construction is based, obviously, on the building plans. The required dimensions (length and width) are extracted from the plans using appropriate scaling conventions and the surface areas are calculated according to well-established conventions.

Likewise, measuring the functional size of software prior to its development is based on the software functional user requirements which are derived from its 'plans'; i.e. the software artifacts produced prior to development. The functional user requirements are derived from these artifacts using appropriate conventions and the required dimensions (the number of data movements) are identified so that the size can be calculated.

To further pursue the house analogy, measuring its surface areas after it has been constructed entails a somewhat different measurement process. Now, the required dimensions (length and width) are extracted from the building itself using a different tool (measuring tape). However, although the physical object being measured differs (the house rather than its plans), the dimensions, the unit of measure (including any scaling convention) and the measurement principles all remain the same.

Likewise, measuring the functional size of software after it has been put into production entails a somewhat different measurement process when the required dimensions are extracted from the various artifacts of the software itself. Although the nature of these artifacts differs, the dimensions, the unit of measure and the measurement principles remain the same.

It is up to the measurer, based on the measurement purpose, to determine if the object to be measured is the house as verbally described by its owner, the house as described through the plans, or the house as it has been built, and to select the most appropriate artifacts for the measurement. Clearly, the three sizes could be different. The same reasoning applies to the measurement of software.

EXAMPLES: The following are typical measurement purposes

- To measure the size of the FUR as they evolve, as input to a process to estimate development effort
- To measure the size of changes to the FUR after they have been initially agreed, in order to manage project 'scope creep'

- To measure the size of the FUR of the delivered software as input to the measurement of the developer's performance
- To measure the size of the FUR of the total software delivered, and also the size of the FUR of the software which was developed, in order to obtain a measure of functional re-use
- To measure the size of the FUR of the existing software as input to the measurement of the performance of those responsible for maintaining and supporting the software
- To measure the size of some changes to (the FUR of) an existing software system as a measure of the size of the work-output of an enhancement project team
- To measure the size of functionality of the existing software provided to human functional users

2.1.2 The importance of the purpose

The purpose helps the measurer to determine:

- the scope to be measured and hence the artifacts which will be needed for the measurement
- the functional users (as will be shown in section 2.3, the functional size changes depending on who or what is defined as the functional user)
- the point in time in the project life-cycle when the measurement will take place
- the required accuracy of the measurement, and hence whether the COSMIC measurement method should be used, or whether a locally-derived approximation version of the COSMIC method should be used (e.g. early in a project's life-cycle, before the FUR are fully elaborated). Both of these latter two points will determine the level of granularity at which the FUR will be measured.

2.2 Defining the scope of the measurement

DEFINITION – Scope of a measurement

The set of Functional User Requirements to be included in a specific functional size measurement exercise.

NOTE: A distinction should be made between the 'overall scope', i.e. all the software that should be measured according to the purpose, and the 'scope' of any individual piece of software within the overall scope, whose size should be measured separately. In this Measurement Manual, the term 'scope' (or the expression 'measurement scope') will relate to an individual piece of software whose size must be measured separately.

Functional User Requirements are defined by ISO as follows

DEFINITION – Functional User Requirements (FUR)

A sub-set of the User Requirements. Requirements that describe what the software shall do, in terms of tasks and services.

NOTE: Functional User Requirements relate to but are not limited to:

- data transfer (for example Input customer data; Send control signal)
- data transformation (for example Calculate bank interest; Derive average temperature)
- data storage (for example Store customer order; Record ambient temperature over time)
- data retrieval (for example List current employees; Retrieve latest aircraft position)

Examples of User Requirements that are not Functional User Requirements include but are not limited to:

- quality constraints (for example usability, reliability, efficiency and portability)
- organizational constraints (for example locations for operation, target hardware and compliance to standards)
- environmental constraints (for example interoperability, security, privacy and safety)
- implementation constraints (for example development language, delivery schedule)

RULES – Scope

a) The scope of a Functional Size Measurement (FSM) shall be derived from the purpose of the measurement.

b) The scope of any one measurement shall not extend over more than one layer of the software to be measured

2.2.1 Deriving the scope from the purpose of a measurement

It is important to define the scope of a measurement, before commencing a particular measurement exercise.

Continuing with the analogy of house construction, it may be necessary if the purpose is cost estimating to measure the size of different parts of the house separately, e.g. the foundations, walls and roof, because they use different construction methods. The same is true for estimating software development costs.

If a software system to be developed happens to consist of pieces that will reside in different layers of the system architecture, then the size of the software in each layer must be measured separately, that is, each piece will have a separate scope defined for size measurement purposes. This follows from principle (d) of the Software Context Model. (For more on layers, see section 2.2.4 below.)

Similarly, if the software must be developed as a set of peer components within a single layer, each using different technologies, then it will be necessary to define a separate measurement scope for each component before measuring their sizes.

EXAMPLE 1: If each component of the software uses a different technology and the measurements are to be used for estimating development effort, then a separate measurement scope should be defined for each component because each size measurement will be associated with a different development productivity figure. (For more on peer components, see section 2.2.5 below.)

The purpose will also help decide what software should be included and excluded from the measurement scope

EXAMPLE 2: If the purpose is to measure the functional size of all of the software delivered by a particular project team, it will first be necessary to define the functional user requirements of all the various pieces and components to be delivered by the team. These might include the FUR of a piece of software which was used only once to convert data from software which is being replaced.

EXAMPLE 3: If the purpose is to measure the size of the new software that is available for operational use, this would be smaller than for Example 2, as the FUR of the software used for conversion would not be included in the scope of the measured size.

In summary, the purpose of the measurement must always be used to determine (a) what software is included or excluded from the overall scope and (b) the way the included software may need to be divided up into separate pieces, each with its own scope, to be measured separately.

2.2.2 Generic types of Scope

EXAMPLES

- An enterprise portfolio
- A contractually-agreed statement of requirements
- A project team's *delivered* work-output (i.e. including that obtained by exploiting existing software parameters, bought-in packages and re-usable code, any software used for data conversion and subsequently discarded, and utilities and testing software developed specifically for this project)
- A project team's *developed* work-output (i.e. including any software developed by the team and used for data conversion but subsequently discarded, and any utilities and testing software developed specifically for this project, but *excluding* all functionality obtained by changing parameters and exploiting re-usable code or by bought-in packages)
- All the software in a layer
- A software package
- An application
- A major ('peer') component of an application
- A re-usable object-class
- All the changes required for a new release of a piece of existing software

In practice a scope statement needs to be explicit rather than generic, e.g. the developed work-output of project team 'A', or application 'B', or the portfolio of enterprise 'C'. The scope statement may also, for clarity, need to state what is excluded.

2.2.3 Levels of decomposition

Note that some of the 'generic types of scope' listed above correspond to different 'levels of decomposition' of software, defined as follows:

DEFINITION – Level of decomposition
Any level resulting from dividing a piece of software into components (named 'Level 1', for example), then from dividing components into sub-components ('Level 2'), then from dividing sub-components into sub-sub components (Level 3'), etc. NOTE 1: Not to be confused with 'level of granularity'. NOTE 2: Size measurements of the components of a piece of software may only be directly comparable for peer components, i.e. components at the same level of decomposition.

EXAMPLE: Different levels of decomposition, corresponding to different 'generic types of scope' would be when an 'application portfolio' consists of multiple 'applications', each of which may consist of 'major (peer) components', each of which may consist of 're-usable object classes'.

Determining a measurement scope may therefore be more than just a question of simply deciding what functionality should be included in the measurement. The decision may also involve consideration of the level of decomposition of software at which the measurement(s) will be made. This is an important Measurement Strategy decision, dependent on the purpose of the measurement, because measurements at different levels of decomposition may not be easily compared. This arises because, as will be seen in section 4.3.1 rule g), the size of any piece of software cannot be obtained by simply adding up the sizes of its components.

2.2.4 Layers

Since the scope of a piece of software to be measured must be confined to a single software layer, the process of defining the scope may require that the measurer first has to decide what are the layers of the software architecture. In this section we will therefore define and discuss 'layers' of software as these terms are used in the COSMIC method.

The reasons why we need these definitions and rules are as follows

- The measurer may be faced with measuring some software in a 'legacy' environment of software that evolved over many years without ever having been designed according to an underlying architecture (a so-called 'spaghetti architecture'). The measurer may therefore need guidance on how to distinguish layers according to the COSMIC terminology
- The expressions 'layer', 'layered architecture' and 'peer component' are not used consistently in the software industry. If the measurer must measure some software that is described as being in a 'layered architecture', it is advisable to check that 'layers' in this architecture are defined in a way that is compatible with the COSMIC method. To do this, the measurer should establish the equivalence between specific architectural objects in the 'layered architecture' paradigm and the concept of layers as defined in this manual

Layers may be identified according to the following definitions and principles.

DEFINITION – Layer

A layer is a partition resulting from the functional division of a software system which together with hardware forms a whole computer system where:

- layers are organized in a hierarchy
- there is only one layer at each level in the hierarchy
- there is a 'superior/subordinate' hierarchical dependency between the functional services provided by software in any two layers in the software architecture that exchange data directly
- the software in any two layers in the software architecture that exchange data interpret only part of that data identically

Layer identification is an iterative process. The exact layers will be refined as the mapping process progresses. Once identified, each candidate layer must comply with the following principles and rules:

PRINCIPLES – Layer

a) Software in one layer exchanges data with software in another layer via their respective functional processes.

b) The 'hierarchical dependency' between layers is such that software in any layer may use the functional services of any software in any layer beneath it in the hierarchy. Where there are such usage relationships, we designate the using software layer as the 'superior' and any layer containing the used software as its 'subordinate'. The software in the superior layer relies on the services of software in these subordinate layers to perform properly; the latter rely in turn on software in their subordinate layers to perform properly, and so on, down the hierarchy. Conversely, software in a subordinate layer, together with software in any subordinate layers on which it depends, can perform without needing the services of software in any superior layer in the hierarchy.

c) Software in one layer does not necessarily use all the functional services supplied by software in a subordinate layer.

d) The data that is exchanged between software in any two layers is defined and interpreted differently in the respective FUR of the two pieces of software, that is, the two pieces of software recognize different data attributes and/or sub-groupings

of the data that they exchange. However, there must also exist one or more commonly defined data attributes or sub-groups to enable the software in the receiving layer to interpret data that has been passed by the software in the sending layer, according to the receiving software's needs.

RULES – Layer

a) If software is conceived using an established architecture of layers according to the COSMIC model, then that architecture should be used to identify the layers for measurement purposes

b) In the domain of MIS or business software, the 'top' layer, i.e. the layer that is not a subordinate to any other layer, is normally referred to as the 'application' layer. (Application) software in this layer ultimately relies on the services of software in all the other layers for it to perform properly. In the domain of real-time software, software in the 'top layer' is commonly referred to as a 'system', for example as in 'process control system software', 'flight control system software'..

c) Do not assume that any software that has evolved without any consideration of architectural design or structuring can be partitioned into layers according to the COSMIC model.

Functional service software packages such as database management systems, operating systems or device drivers, should normally be considered to be located in separate layers.

Once identified, each layer can be registered as a separate 'component' in the Generic Software Model matrix (appendix A), with the corresponding label.

EXAMPLE 1: The physical structure of a typical layered software architecture (using the term 'layer' as defined here) supporting business applications software is given in Figure 2.2.4.1:

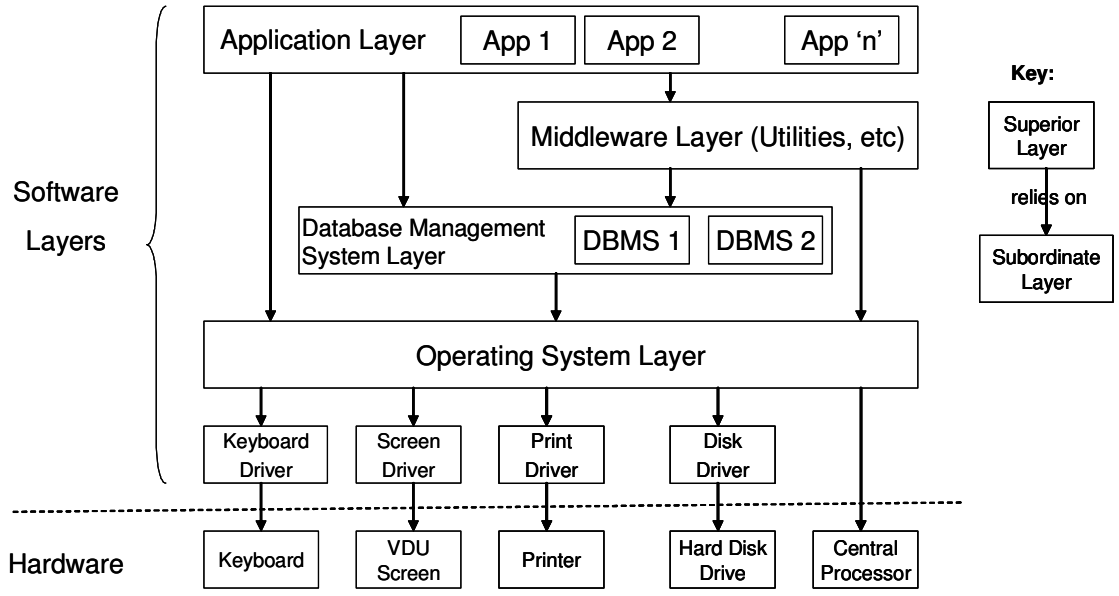


Figure 2.2.4.1 - Typical layered software architecture for a Business/MIS computer system

EXAMPLE 2: The physical structure of a typical layered software architecture (again using the term 'layer' as defined here) supporting a piece of embedded real-time software is given in Figure 2.2.4.2:

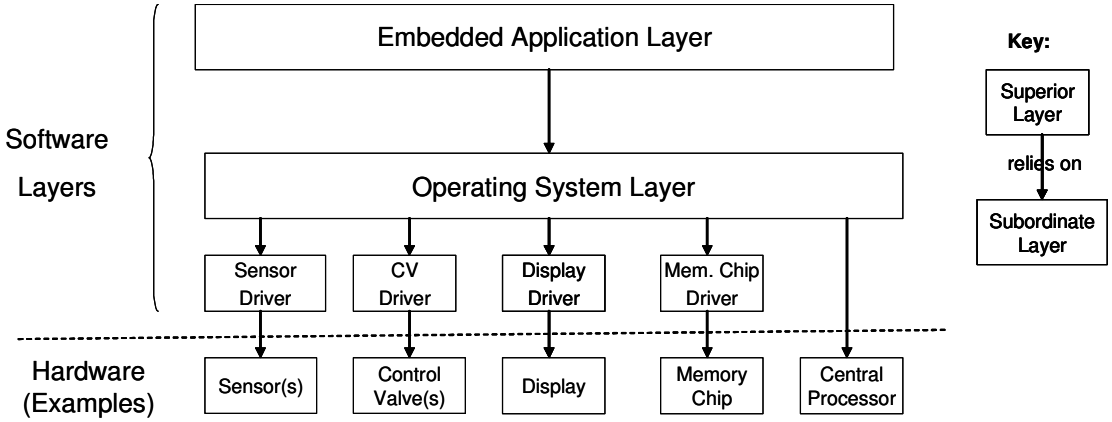


Figure 2.2.4.2 - Typical layered architecture for a real-time embedded-software computer system

2.2.5 Peer components

Given the concept of layers, 'peer' is defined as follows:

DEFINITION – Peer
Two pieces of software are peers of each other if they reside in the same layer. NOTE. Two peer pieces of software do not have to be at the same level of decomposition.

Peer components may be identified according to the following definition and principles.

DEFINITION – Peer component
One component of a set of co-operating components, all at the same level of decomposition, that results from dividing up a piece of software within one layer, where each component fulfills a portion of the Functional User Requirements of that piece of software. NOTE: The division of a piece of software into peer components may be in response to functional and/or non-functional user requirements.

Once identified, each candidate peer component must comply with the following principles:

PRINCIPLES – Peer component
a) In a set of peer components of a piece of software in one layer there is no hierarchical dependency between the peer components as there is between layers. The FUR of all peer components of a piece of software in any one layer are at the same 'level' in the hierarchy of layers. b) All peer components of a piece of software must co-operate in order that the piece of software can perform successfully. c) A data group may be exchanged directly between two peer components of a piece of software by a functional process of a first component issuing an Exit which is received as an Entry by a functional process of the second component. Alternatively, the exchange may take place indirectly by a functional process of a first component making a data group persistent via a Write that can be subsequently retrieved via a Read of a functional process of the second component.

Once identified, each peer component can be registered as an individual component in the Generic Software Model matrix (appendix A), with the corresponding label.

EXAMPLE: When a business application is developed as three main components, namely a 'user interface' (or 'front-end') component, a 'business rules' component and a 'data services' component the three components are peer components.

Note: Two separate, but peer, pieces of software that exist in the same layer and that exchange data with each other may do so by either of the alternative sequences that are defined in Principle c) for two peer components.

The following diagram shows a situation that illustrates the Example All the software shown resides in the same application layer. The exchanges between the two peer components of Application X and between the two peer pieces of software (Component 2 of Application X and Application Y) may take place via either of the alternative sequences that are defined in Principle c) for two peer components.

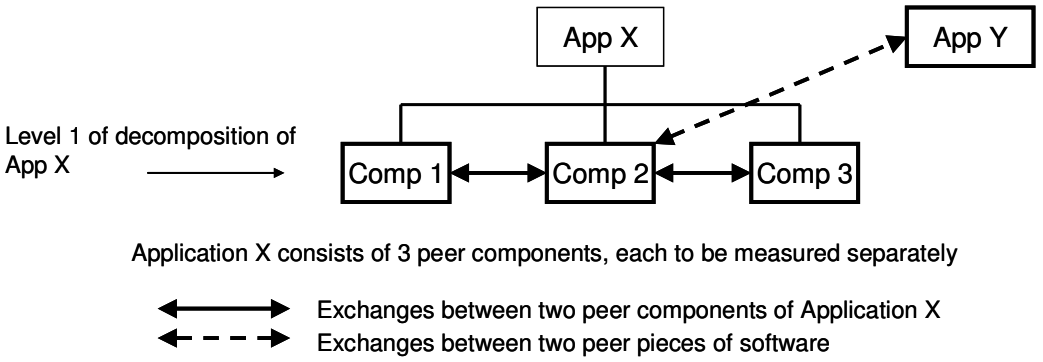


Figure 2.2.5.1 – Relation between the concepts of 'peer', 'component' and 'peer component'

2.3 Identifying the functional users

2.3.1 Functional size varies with the functional user

Starting with an analogy, measurement of the floor areas of an office can be undertaken according to four different conventions, as follows. (N.B. the scope – the particular office – is the same for all four conventions.)

- The building owner has to pay taxes on the office. For the building owner, the surface area is the 'gross sq. m.', determined from the external dimensions and therefore including all public hallways, space occupied by walls, etc.
- The building heating manager is interested in the 'net sq. m.', i.e. the internal areas, including public areas and space taken by elevators, but excluding the thickness of walls.
- The cleaning services contractor employed by the office Lessee is interested in the 'net-net plus sq. m.', which excludes the public areas, but includes passageways used by the lessee.
- The office planning manager is interested in the 'net-net sq. m.', i.e. only the space usable for offices.

The lesson from this analogy is that different types of users of a 'thing' may 'see' different functionality and hence may measure different sizes of the 'thing'. In the case of software, different (types of) functional users may require (via their FUR) and use different functionality and therefore functional sizes will vary with the choice of functional users

2.3.2 Functional users

A 'user' is defined, in effect[5], as 'any thing that interacts with the software being measured'. This definition is too broad for the needs of the COSMIC method. For the COSMIC method, the choice of user (or users) is determined by the functional 'user' requirements that must be measured. This (type of) user, known as the 'functional user', is defined as follows.

DEFINITION – Functional user
A (type of) user that is a sender and/or an intended recipient of data in the Functional User Requirements of a piece of software.

In the COSMIC method it is essential to distinguish the functional users of a piece of software that must be measured from all of its possible users.

EXAMPLE 1: Consider a business application; ts functional users would normally include humans and other peer applications with which the application interfaces. For a real-time application, the functional users would normally be engineered hardware devices or other interfacing peer software. The functional user requirements (FUR) of such software are normally expressed so that the functional users are the senders of data and/or the intended recipients of data to and from the software, respectively.

However, the total set of 'users', i.e. including 'any thing that interacts with the software', must include the operating system. But the FUR of any application would never include the operating system as a user. Any constraints that the operating system may impose on an application will be common to all applications, will normally be handled by the compiler or interpreter, and are invisible to the real functional users of the application. In practical functional size measurement, an operating system would never be considered as a functional user of an application[6].

But it is not always the case that the functional users are obvious.

EXAMPLE 2: Consider the application software of a mobile phone (first mentioned in the Introduction). Although we have eliminated the operating system of the mobile phone as a possible functional user of the application, the 'users' could still be either (a) humans who press the keys, or (b) hardware devices (e.g. the screen, keys, etc) and peer applications that interact directly with the phone application. The human user, for example, will see only a sub-set of all the functionality that must be provided to enable the mobile phone application to work. So these two types of users will 'see' different functionality; the size of the FUR for the human user will be smaller than the size of the FUR that must be developed to make the phone application work.[7]

RULES – Functional users
a) The functional users of a piece of software to be measured shall be derived from the purpose of the measurement
b) When the purpose of a measurement of a piece of software is related to the effort to develop or modify the piece of software, then the functional users should be those for whom the new or modified functionality must be provided.

[5] See the glossary for the definition, taken from ISO/IEC 14143/1:2007

[6] In reality, of course, the reverse is true. Applications are functional users of an operating system. The FUR of an operating system are defined taking into account the applications that it must support as its functional users.

[7] Toivonen, for example, compared the functionality of mobile phones available to human users in "Defining measures for memory efficiency of the software in mobile terminals", International Workshop on Software Measurement, Magdeburg, Germany, October 2002

Having identified the functional users, it is then straightforward to identify the boundary, as it simply lies between the piece of software being measured and its functional users. We ignore any other hardware or software in that intervening space[8].

DEFINITION – Boundary
The boundary is defined as a conceptual interface between the software being measured and its functional users. NOTE: The boundary of a piece of software is the conceptual frontier between this piece and the environment in which it operates, as it is perceived externally from the perspective of its functional users. The boundary allows the measurer to distinguish, without ambiguity, what is included inside the measured software from what is part of the measured software's operating environment.

N.B. This definition of 'Boundary' is taken from ISO/IEC 14143/1:2007, modified by the addition of 'functional' to qualify 'user'. To avoid ambiguity, note that the boundary should not be confused with any line that might be drawn around some software to be measured to define the measurement scope. The boundary is not used to define the scope of a measurement.

The following rules might be useful to confirm the status of a candidate boundary:

RULES – Boundary
a) Identify the functional user(s) that interact with the software being measured. The boundary lies between the functional users and this software. b) By definition, there is a boundary between each identified pair of layers where the software in one layer is the functional user of software in another, and the latter is to be measured. c) There is a boundary between any two pieces of software, including any two components that are peers of each other; in this case each piece of software and/or each component can be a functional user of its peer.

The boundary allows a clear distinction to be made between anything that is part of the piece of software being measured (*i.e. that is on the software side of the boundary*) and anything that is part of the functional users' environment (*i.e. that is on the functional users' side the boundary*). Persistent storage is not considered as a user of the software and is therefore on the software side of the boundary.

2.4 Identifying the level of granularity

2.4.1 The need for a standard level of granularity

When a project is started to design and construct a new house, the first plans drawn by an architect are at a 'high level', that is, they show an outline and little detail. As the project progresses towards the construction phase, more detailed ('low level') plans are needed.

The same is true for software. In the initial stages of a software development project, Functional User Requirements (FUR) are specified 'at a high level', that is, in outline, or in little detail. As the project progresses, the FUR are refined, (e.g. through versions 1, 2, 3 etc.), revealing more detail 'at a lower level'. These different degrees of detail of the FUR are known as different 'levels of granularity'. (See

[8] In fact, if the measurer has had to examine the FUR in order to identify the senders and intended recipients of data, the boundary will have already been identified.

also section 2.4.3 for other terms that may be confused with the concept of 'level of granularity' as defined here.)

DEFINITION – Level of granularity
Any level of expansion of the description of a single piece of software (e.g. a statement of its requirements, or a description of the structure of the piece of software) such that at each increased level of expansion, the description of the functionality of the piece of software is at an increased and uniform level of detail. NOTE: Measurers should be aware that when requirements are evolving early in the life of a software project, at any moment different parts of the required software functionality will typically have been documented at different levels of granularity

House construction plans are drawn to standard scales, and it is easy to translate dimensions measured on one drawing to those on another drawing with a different scale. In contrast there are no standard scales for the various levels of granularity at which software may be specified, so it may be difficult to be sure that two statements of FUR are at the same level of granularity. Without agreement on some standard level of granularity at which to measure (or to which measurements should be scaled) it is impossible to know for sure that two functional size measurements may be compared. And measurers may have to develop their own method of scaling measurements at one level of granularity to another.

To illustrate the problems further, consider another analogy. A set of road maps reveals the details of a national road network at three levels of granularity.

- Map A shows only motorways and main highways
- Map B shows all motorways, main and secondary roads (as in an atlas for motorists),
- Map C shows all roads with their names (as in a set of local district maps),

If we did not recognize the phenomenon of different levels of granularity, it would appear that these three maps revealed different sizes of the nation's road network. Of course, with road maps, everyone understands the different levels of detail shown and there are standard scales to interpret the size of the network revealed at any level. The abstract concept of 'level of granularity' lies behind the scales of these different maps.

For software measurement, there is only one standard level of granularity that it is possible to define unambiguously. That is the level of granularity at which individual functional processes have been identified and their data movements defined. Measurements should be made at this level or scaled to this level whenever possible[9].

2.4.2 Clarification of 'level of granularity'

Before proceeding further, it is important to ensure there is no misunderstanding about the meaning of 'level of granularity' in the COSMIC method. As defined here, expanding the *description* of some software from a 'higher' to a 'lower' level of granularity involves 'zooming in' and revealing more detail, *without changing its scope*. This process should NOT be confused with any of the following.

- Zooming-in on an artifact describing some software in order to reveal different sub-sets of the functionality delivered to different users, hence probably limiting the functionality to be measured
- Zooming in on some software in order to reveal its components, sub-components, etc (at different 'levels of decomposition' – see section 2.2.2 above). Such zooming in may be required if the

[9] The topic of scaling measurements from one level of granularity to another was already introduced in v2.2 of the Measurement Manual in chapter 7 on 'Early approximate sizing using COSMIC'. (This topic will now be found in the document 'Advanced and Related Topics').

measurement purpose requires the overall measurement scope to be sub-divided following the physical structure of the software

- Evolving the description of some software as it progresses through its development cycle, e.g. from requirements to logical design, to physical design, etc. Whatever the stage in the development of some software, we are only interested in the FUR for measurement purposes

The concept of 'level of granularity' is therefore intended to be interpreted as applying only to the Functional User Requirements of software. Other ways of 'zooming in' or 'breaking down' software or its various descriptions can also be very important when using or comparing functional size measurements. These will be dealt with in the document 'Advanced and Related Topics, v3.0', in the chapter on 'Ensuring the Comparability of Size Measurements'.

2.4.3 The standard level of granularity

The standard level of granularity for measurement is the 'functional process level' defined as follows[10].

DEFINITION - Functional process level of granularity
A level of granularity of the description of a piece of software at which the functional users • are individual humans or engineered devices or pieces of software (and not any groups of these) AND • detect single occurrences of events that the piece of software must respond to (and not any level at which groups of events are defined) NOTE 1: In practice, software documentation containing functional user requirements often describes functionality at varying levels of granularity, especially when the documentation is still evolving. NOTE 2: 'Groups of these' (functional users) might be, for example, a 'department' whose members handle many types of functional processes; or a 'control panel' that has many types of instruments; or 'central systems'. NOTE 3: 'Groups of events' might, for example, be indicated in a statement of FUR at a high level of granularity by an input stream to an accounting software system labeled 'sales transactions'; or by an input stream to an avionics software system labeled 'pilot commands'

With this definition, we can now define the following rules and a recommendation.

RULES - Functional process level of granularity
a) Functional size measurement should be made at the functional process level of granularity b) Where a functional size measurement is needed of some FUR that have not yet evolved to the level where all the functional processes have been identified and all the details of their data movements have been defined, measurements should be made of the functionality that has been defined, and then scaled to the level of granularity of functional processes. (See the 'Advanced and Related Topics' document for methods of 'approximate sizing, i.e. of estimating a functional size early in the process of establishing FUR.)

[10] The reason for the name 'functional process level of granularity' is that this is the level at which functional processes and their data movements are identified – see section 3.2 for a more detailed discussion of functional processes.

In addition to the rules, COSMIC *recommends*[11] that the functional process level of granularity should be the standard at which functional size measurements are required and used by providers of benchmarking services and of software tools designed to support or use functional size measurements, e.g. for estimating project effort.

EXAMPLE: Functionality at different levels of granularity

The example, from the domain of business application software, will be of a well-known system for ordering goods over the Internet, which we will call the 'Everest' system. (The 'Advanced and Related Topics' document contains an example from the domain of real-time software.). The description below is highly-simplified for the purposes of this illustration of levels of granularity. The description also covers only the functionality available to Everest's customer users. It thus excludes functionality that must be present so that the system can complete the supply of goods to a customer, such as functionality available to Everest staff, product suppliers, advertisers, payment service suppliers, etc.

The scope of the measurement is therefore defined as 'the parts of the Everest application system accessible to customers over the Internet'. We assume the purpose of the measurement is to determine the functional size of the application available to the human customer users (as 'functional users'). At the highest 'level 1' of the whole application a statement of the Functional User Requirements (FUR) of the Everest Ordering System would be a simple summary statement such as the following.

"The Everest system must enable customers to enquire upon, select, order, pay for and obtain delivery of any item of Everest's product range, including products available from third party suppliers."

Zooming-in on this highest-level statement we find that at the next lower level 2 the system consists of four sub-systems, as shown on Fig. 2.4.3.1.

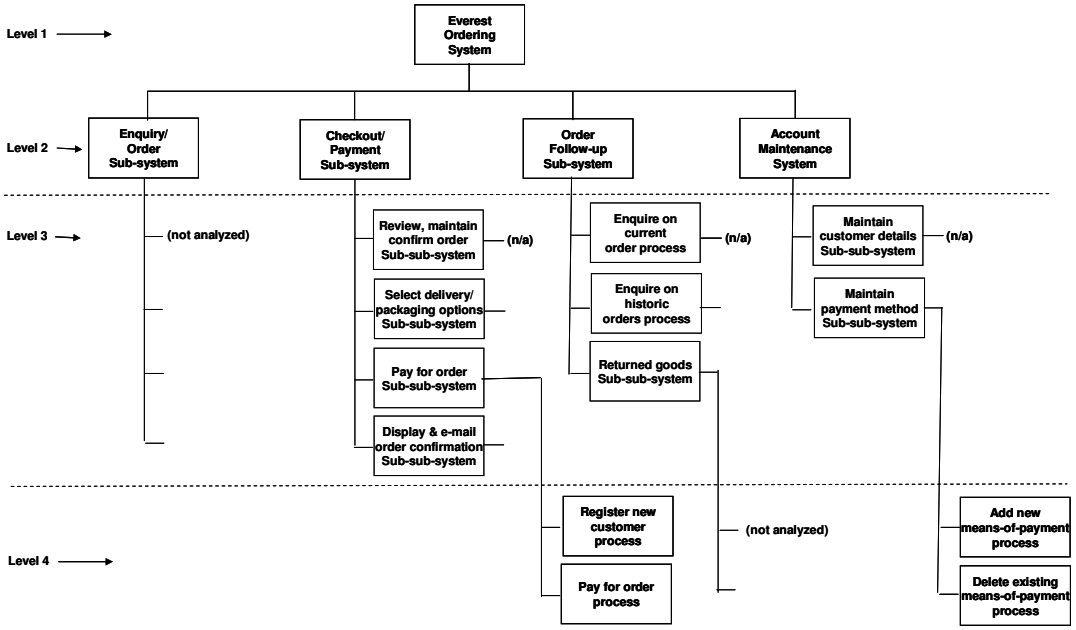


Figure 2.4.3.1 - Partial analysis of the Everest Ordering System showing four levels of analysis

The four sub-systems are:

- The Enquiry/Order sub-system which enables a customer to find any product in the Everest database, as well as its price and availability and to add any selected product to a 'basket' for purchase. This sub-system also promotes sales

[11] The reason that use of the functional process level of granularity is 'recommended' rather than given as a rule is that this recommendation applies not just to individual users of the COSMIC method but to their networks of suppliers of services and tools that use the size measurements. COSMIC can only make recommendations to this wider community.

by suggesting special offers, offering reviews of selected items and enabling general enquiries such as on delivery terms, etc. It is a very complex sub-system and is not analyzed in any further detail below level 2 for the purposes of this example.

- The Checkout/Payment sub-system which enables a customer to commit to order and pay for the goods in the basket
- The Order Follow-up sub-system which enables a customer to enquire how far an existing order has progressed in the delivery process, to maintain their order (e.g. change delivery address) and to return unsatisfactory goods
- The Account Maintenance sub-system which enables an existing customer to maintain various details of his/her account such as home address, means of payment, etc.

Figure 2.4.3.1 shows some detail that is revealed when we zoom-in, down two further levels on the Checkout/Payment sub-system, the Order Follow-up sub-system and the Account Maintenance sub-system. In this zooming-in process it is important to note that

- we have not changed the scope of the application system to be measured, and
- all levels of the description of the Everest software system show the functionality available to the customers (as functional users). A customer can 'see' the functionality of the system at all these levels of the analysis

Figure 2.4.3.1 also reveals that when we zoom-in to lower levels of this particular analysis of the three sub-systems, we find individual functional processes at level 3 (for two enquiries of the Order follow-up sub-system) and at level 4 for the other two sub-systems. This example demonstrates, therefore, that when some functionality is analyzed *in a top-down approach*, it cannot be assumed that the functionality shown at a particular 'level' on a diagram will always correspond to the same 'level of granularity' as this concept is defined in the COSMIC method. (This definition requires that at any one level of granularity the functionality is 'at a comparable level of detail'.)

Furthermore, other analysts might well draw the diagram differently, showing other groupings of functionality at each level of the diagram. There is not one 'correct' way of analyzing the functionality of such a complex system.[12]

Given these variations that inevitably occur in practice, a measurer must carefully examine the various levels of an analysis diagram to find the functional processes that must be measured. Where in practice this is not possible, for example because the analysis has not yet reached the level where all functional processes have been revealed, rule (b) above must be applied. To illustrate this, let us examine the case of the 'Maintain customer details sub-sub-system' (see figure 2.4.3.1 above), in the branch of the Account Maintenance sub-system.

To an experienced measurer, the word 'maintain' almost invariably suggests a group of events and thus a group of functional processes. We can therefore safely assume that this 'Maintain' sub-sub-system must comprise three functional processes, namely an 'enquire on customer details',' update customer details' and 'delete customer'. (The 'create customer details' process must also obviously exist, but this occurs in another branch of the system, when a customer orders goods for the first time. It is outside the scope of this simplified example.)

An experienced measurer should be able to 'guestimate' a size of this sub-sub-system in units of CFP by taking the assumed number of functional processes (three in this case) and multiplying this number by the average size of a functional process. This average size would be obtained by calibration in other parts of this system or in other comparable systems. Examples of this calibration process are given in the document 'Advanced and Related Topics' on approximate sizing which also contains other examples of other approaches to approximate sizing.

Clearly, such approximation methods have their limitations. If we apply such an approach to the level 1 statement of FUR as given above ("*The Everest system must enable customers to enquire upon, select, order, pay for and obtain delivery of any item of Everest's product range ….*), we could identify a few functional processes. But more detailed analysis would reveal that the real number of functional processes in this complex system must be much greater. That is why functional sizes usually appear to increase as more details of the requirements are established, even without changes in scope. These approximation methods must therefore be used with great care at high levels of granularity, when very little detail is available.

The document 'Advanced and Related Topics' gives a different type of example of sizing at various levels of granularity, in this case of some software that is part of a telecommunications software architecture. This example is more complex than the 'Everest' example since the functional users of

[12] Figure 2.4.3.1 may not even be an example of best practice, but it is typical of how such diagrams may be drawn.

the part of the architecture being measured are all other pieces of software of the same architecture. This example therefore also deals with varying levels of decomposition of the software being measured and of its functional users.

2.5 Concluding remarks on the measurement strategy phase

Carefully considering the four elements of the measurement strategy process before starting a measurement should ensure that the resulting size can be properly interpreted. The four elements are:

a) establish the *purpose* of the measurement
b) define the overall *scope* of the piece of software to be measured and the scope of pieces to be separately measured, considering the layers and peer components of the software architecture
c) establish the *functional users* of the piece of software that is to be measured
d) establish the level of granularity of the software artifacts to be measured and how, if necessary, to scale to sizes at the standard *functional process level of granularity*

Some iteration may be needed around steps (b), (c) and (d) when requirements are evolving and new details indicate the need to refine the definition of the scope.

The great majority of functional size measurements are carried out for a purpose that is related to development effort in some way, e.g. for developer performance measurement, or for project estimating. In these situations, defining the measurement strategy should be very straightforward. The purpose and scope are usually easy to define, the functional users are the users for whom the developer must provide the functionality, and the level of granularity at which the measurements are required is that at which the functional users detect single events.

However, not all measurements fit this common pattern, so the measurement strategy should be defined in each situation.

3

THE MAPPING PHASE

This chapter presents the rules and method for the mapping process. The general method for mapping software to the COSMIC Generic Software Model is summarized in Figure 3.0.

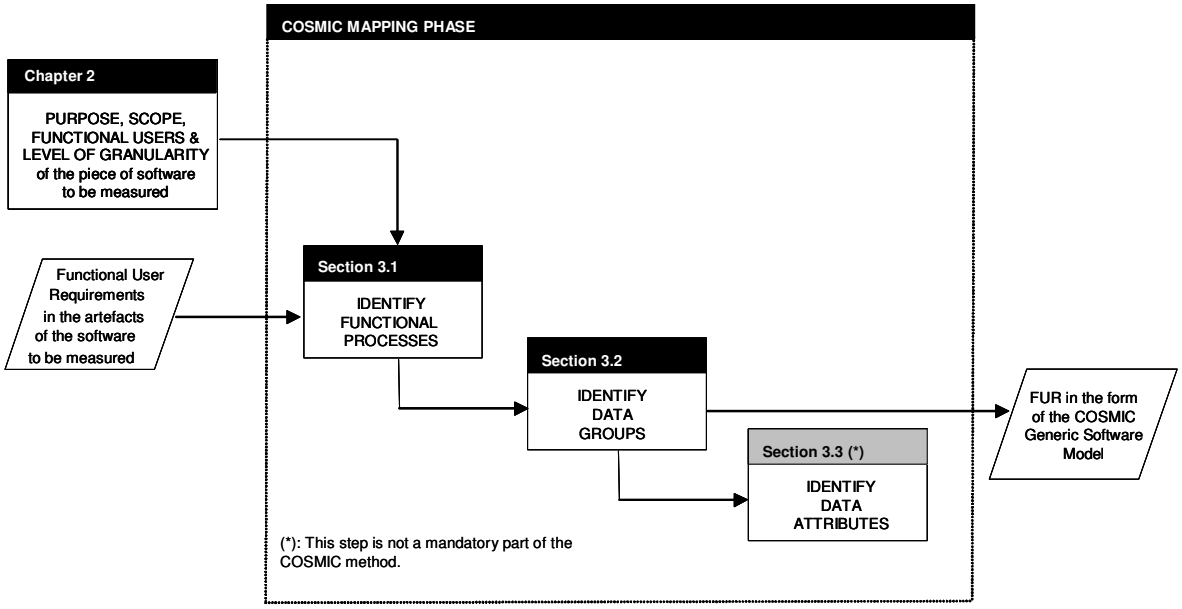


Figure 3.0 – General method of the COSMIC mapping process

Each step in this method is the subject of a specific section (indicated in the step's title bar in figure 3) where the definitions and rules are presented, along with some guiding principles and examples.

The general method outlined in figure 3 above is designed to be applicable to a very broad range of software artifacts. A more systematic and detailed process would provide precise mapping rules for a larger collection of highly specific artifacts, thus diminishing the level of ambiguity when generating the COSMIC Generic Software Model. Such a process would, by definition, be highly dependent on the nature of the artifacts, which, in turn, depends on the software engineering methodology in use in each organization.

The document 'Guideline for Sizing Business Application Software using COSMIC' gives guidance on the mapping from various data analysis and requirements determination methods used in the business application domain to the concepts of the COSMIC method.

3.1 Applying the Generic Software Model

PRINCIPLE – Applying the COSMIC Generic Software Model
The COSMIC Generic Software Model shall be applied to the functional user requirements of each separate piece of software for which a separate measurement scope has been defined. 'Applying the COSMIC Generic Software Model' means identifying the set of triggering events sensed by each of the functional user (types) identified in the FUR, and then identifying the corresponding functional processes, objects-of-interest, data groups, and data movements that must be provided to respond to those events.

Figures 3.1.1 and 3.1.2 below illustrate the application of principle (g) of the Software Context Model and the principles of the Generic Software Model to a piece of business application software and to a typical piece of real-time embedded software respectively.

Whereas Figures 2.2.3.1 and 2.2.3.2 above showed actual physical views of the layered software architectures, respectively, these Figures 3.1.1 and 3.1.2 show a logical view of the relationships of the various concepts defined in the COSMIC Models. In this logical view the functional users interact with the software to be measured across a boundary via Entry and Exit data movements. The software moves data to and from persistent storage via Write and Read data movements respectively. In these logical views all hardware and software that is needed to enable these exchanges to take place between the software being measured, its functional users and persistent storage is ignored.

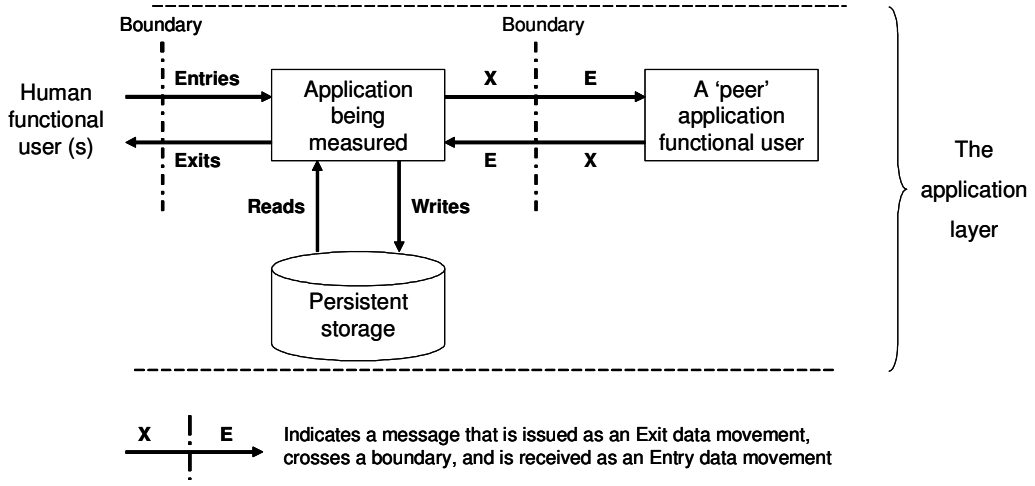


Figure 3.1.1 - A business application with both humans and another 'peer' application as its functional users (logical view)

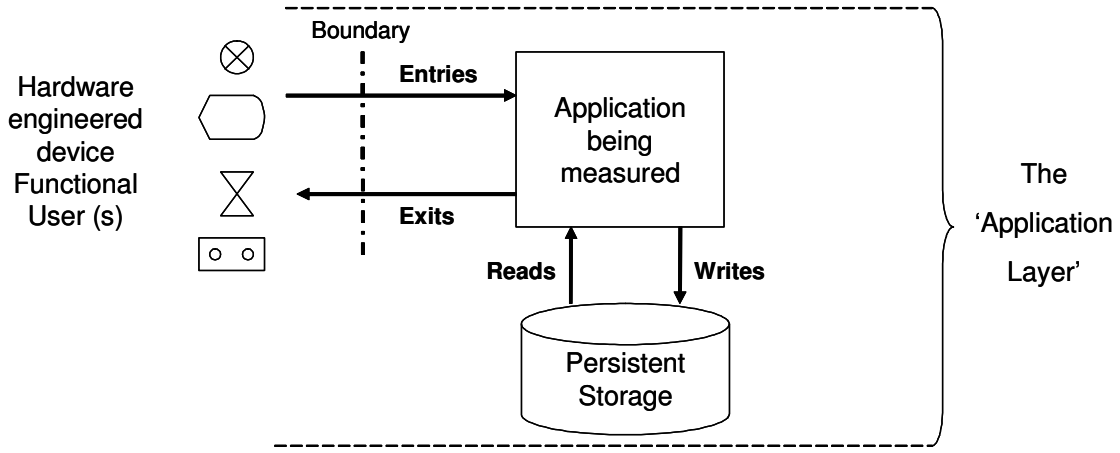


Figure 3.1.2 - A real-time embedded software application with various hardware engineered devices as its functional users (logical view)

3.2 Identifying functional processes

This step consists in identifying the set of functional processes of the piece of software to be measured, from its Functional User Requirements.

3.2.1 Definitions

DEFINITION – Functional process
A functional process is an elementary component of a set of Functional User Requirements comprising a unique, cohesive and independently executable set of data movements. It is triggered by a data movement (an Entry) from a functional user that informs the piece of software that the functional user has identified a triggering event. It is complete when it has executed all that is required to be done in response to the triggering event. NOTE: In addition to informing the piece of software that the event has occurred, the 'triggering Entry' may include data about the object of interest associated with the event.

DEFINITION – Triggering event
An event (something that happens) that causes a functional user of the piece of software to initiate ('trigger') one or more functional processes. In a set of Functional User Requirements, each event which causes a functional user to trigger a functional process • cannot be sub-divided for that set of FUR, and • has either happened or it has not happened.. NOTE: Clock and timing events can be triggering events.

The relationship between a triggering event, the functional user and the Entry data movement that triggers a functional process is depicted by figure 3.2.1 below. The interpretation of this diagram is: *an event is sensed by a functional user, and the functional user triggers a functional process*.

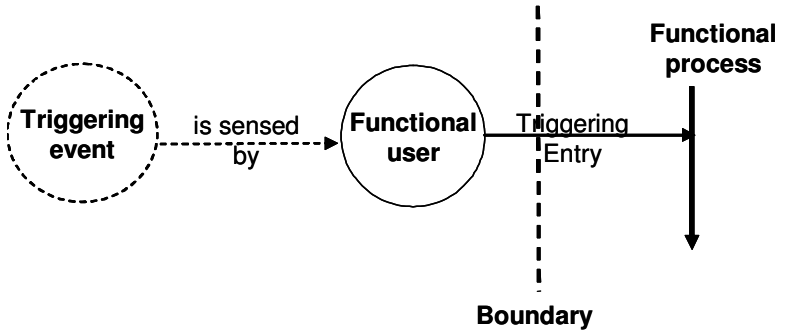


Figure 3.2.1 – Relation between triggering event, functional user and functional process

The triggering Entry is normally a positive, unambiguous message that informs the software that the functional user has identified a triggering event. The triggering Entry often also includes data about an object of interest associated with the event. After the triggering Entry has been received, a functional process may be required to receive and process other Entries describing other objects of interest.

If a functional user sends incorrect data, e.g. because a sensor-user is mal-functioning or an order entered by a human has errors, it is usually the task of the functional process to determine if the event really occurred and/or if the entered data are really valid, and how to respond.

3.2.2 The approach of identifying functional processes

The approach to identifying functional processes depends on the software artifacts that are available to the measurer, which in turn depend on the point in the software life-cycle when the measurement is required and on the software analysis, design and development methods in use. Since the latter vary enormously, even within a given software domain, it is beyond the scope of this Measurement Manual to provide one or more general processes for identifying functional processes.

The 'Guideline for sizing business application software using COSMIC', section 4.4 gives more rules and examples on how to identify and distinguish functional processes in the business application software domain.

The most important general advice is that it is almost always useful to try to identify the separate events in the world of the functional users that the software must respond to, since each such event (-type) gives rise to usually one (but sometimes more than one) functional process (-type). Events can be identified in state diagrams and in entity life-cycle diagrams, since each transition that the software must deal with corresponds to an event.

Use the following rules to check that candidate functional processes have been properly identified.

RULES – Functional process
a) A functional process shall be derived from at least one identifiable Functional User Requirement within the agreed scope.
b) A functional process (-type) shall be performed when an identifiable triggering event (-type) occurs.
c) A specific event (-type) may trigger one or more functional process (-types) that execute in parallel. A specific functional process (-type) may be triggered by more than one event (-type).

d) A functional process shall comprise at least two data movements, an Entry plus either an Exit or a Write.

e) A functional process shall belong entirely to the measurement scope of one piece of software in one, and only one, layer.

f) In the context of real-time software, a functional process shall be considered terminated when it enters a self-induced wait state (i.e. the functional process has done all that is required to be done in response to the triggering event and waits until it receives the next triggering Entry).

g) One functional process (-type) shall be identified even if its FUR allow that the one functional process can occur with different sub-sets of its maximum number of input data attributes, and even though such variations and/or differing input data values may give rise to different processing paths through the functional process.

h) Separate event (-types) and therefore separate functional process (-types) should be distinguished in the following cases:

- When decisions result in separate events that are disengaged in time (e.g. entering order data today and later confirming acceptance of the order, requiring a separate decision, should be considered as indicating separate functional processes),
- When the responsibilities for activities are separated (e.g. in a personnel system where the responsibility for maintaining basic personal data is separated from the responsibility for maintaining payroll data, indicating separate functional processes; or for an implemented software package where there is functionality available to a system administrator to maintain the package parameters, which is separate from the functionality available to the 'regular' functional user.)

3.2.3 Triggering events and functional processes in the business applications domain

a) Triggering events of an on-line business application usually occur in the real world of the human functional users of the application. The human user conveys the occurrence of the event to a functional process by entering data about the event.

EXAMPLE 1: In a company, an order is received (triggering event), causing an employee (functional user) to enter the order data (triggering Entry conveying data about an object of interest 'order'), as the first data movement of the 'order entry' functional process.

b) Sometimes, for an application A there may be a peer application B that needs to send data to, or obtain data from, application A. In this case, if application B triggers a functional process when it needs to send data to or obtain data from application A, then application B is a functional user of the application A.

EXAMPLE 2: Suppose that on receipt of an order in Example 1), the order application is required to send the client details to a central client-registration application, which is being measured. Now the order application has become a functional user of the central application. The order application senses the event of receipt of the client data and triggers a functional process in the central client-registration application to store these data, by sending data about the object of interest 'client' as a triggering Entry to the central application.

c) There is no difference in principle to the analysis of a functional process whether it is required to be processed on-line or in batch mode. The overnight processing is a technical implementation decision, and 'all that is required to be done' hasn't happened until the overnight process is completed. A batch-processed stream consists of one or more functional process (-types), each of which should be analyzed 'end-to-end', independently of any other functional processes in the same stream. Each functional process has its own triggering Entry which must be measured.

EXAMPLE 3: Suppose the orders in the Example 1 above are entered on-line, but are stored for automatic overnight batch processing. The functional user is still the human who entered the orders on-line; the triggering Entry is still the order data. There is one functional process for entry and processing of the orders.

d) Periodic signals from a clock ('clock ticks') can physically trigger a functional process.

EXAMPLE 4: Suppose a FUR for an end-of year batch process to report the outcome of business for the year, and to reset positions for the start of the next year. Physically, an end-of-year clock tick generated by the operating system causes the batch stream to start, consisting of one or more functional processes. Those functional processes in the stream that use input data in the stream should be analyzed in the normal way (e.g. the input data for any one functional process comprises one or more Entries, the first of which is the triggering Entry for that process).

However, assume there is a particular functional process in the stream that does not require any input data to produce its set of reports. Physically, the (human) functional user has delegated to the operating system the task of triggering this functional process. Since every functional process must have a triggering Entry, we may consider the end-of-year clock tick that started the batch stream as filling this role for this process. This functional process may then need several Reads and many Exits to produce its reports. Logically, the analysis of this example is no different if the functional user initiates the production of one or more reports via a mouse click on an on-line menu item, rather than delegating the report production to the operating system via a batch stream.

e) A single event may trigger one or more functional processes that execute independently.

EXAMPLE 5: An end-of-week clock tick causes the start of production of reports, and of the process to review expiry of time-limits in a workflow system.

f) A single functional process may be triggered by more than one type of triggering event.

EXAMPLE 6: In a banking system, a full statement might be triggered by an end-of-month batch process but also by a specific customer request.

For several examples on distinguishing triggering events and functional processes in batch streams, see the 'Guideline for Sizing Business Application Software using COSMIC', section 4.6.3.

3.2.4 Triggering events and functional processes in the real-time applications domain

a) A triggering event is typically detected by a sensor.

EXAMPLE 1: When the temperature reaches a certain value (triggering event), a sensor (functional user) is caused to send a signal (triggering Entry data movement) to switch on a warning light (functional process).

EXAMPLE 2: A military aircraft has a sensor that detects the event 'missile approaching'. The sensor is a functional user of the embedded software that must respond to the threat. For this software, an event occurs only when the sensor detects something, and it is the sensor (the functional user) that triggers the software, by sending it a message (triggering Entry) saying, e.g. 'sensor 2 has detected a missile', plus maybe a stream of data about how fast the missile is approaching and its co-ordinates. The missile is the object of interest.

b) Periodic signals from a clock ('clock ticks') can trigger a functional process.

EXAMPLE 3: In some real-time process control software, a tick (triggering event) of a clock (functional user) causes the clock to send a signal (triggering Entry), a one-bit message that tells a functional process to repeat its normal control cycle. The functional process then reads various sensors, receiving data about objects of interest, and takes whatever action is needed. There is no other data accompanying the clock-tick.

c) A single event may trigger one or more functional processes that execute independently and in parallel.

EXAMPLE 4: An emergency condition detected in a nuclear power plant may trigger independent functional processes in different parts of the plant to lower the control rods, start emergency cooling, close valves, sound alarms to warn the operators, etc.

d) A single functional process may be triggered by more than one type of triggering event.

EXAMPLE 5: Retraction of an aircraft's wheels may be triggered by the 'weight-off-ground' detector, or by pilot command.

3.2.5 More on separate functional processes

Software distinguishes events and provides the corresponding functional processes depending only on its FUR. When sizing software, it can sometimes be difficult to decide what the separate events are that the software is required to recognize. This is especially the case where the original FUR are no longer available and where for example the developer may have found it economical to combine several requirements. It may help with the analysis to examine the organization of data input (see below) or to examine the menus for some installed software to help distinguish the separate events that the software must respond to and the corresponding functional processes.

EXAMPLE 1: When there is a functional user requirement for tax credits for an additional child and also for 'working tax credits' for those on low income these are requirements for the software to respond to two events that are separate in the world of the human functional users. Hence there should be two functional processes, even though a single tax form may have been used to capture data for both cases.

EXAMPLE 2: If the income in one instance (for one person) exceeds the limit for working tax credit, and in another instance (for a different person) it does not, this difference does not give rise to two separate functional processes, but indicates two separate conditions catered for in the single functional process.

EXAMPLE 3: As an example of rule (g), if an occurrence of a specific event triggers the Entry of a data group comprising data attributes A, B and C, and the FUR allows that another occurrence of the same event triggers an Entry of a data group which has values for attributes A and B only, this does not result in identifying two functional process (-types). Only one Entry and one functional process (-type) are identified, moving and manipulating data attributes A, B and C.

Once identified, each functional process can be registered on an individual line, under the appropriate layer or peer component, in the Generic Software Model matrix (appendix A), under the corresponding label.

3.2.6 The functional processes of peer components

When the purpose of a measurement is to measure separately the size of each peer component, a separate measurement scope must be defined for each peer component. In such a case the sizing of the functional processes of each component follows all the normal rules as already described.

From the process for each measurement (… define the scope, then the functional users and boundary, etc…) it follows that if a piece of software consists of two or more peer components, there cannot be any overlap between the measurement scope of each component. The measurement scope for each component must define a set of complete functional processes. For example, there cannot be a functional process with part in one scope and part in another. Likewise, the functional processes within the measurement scope for one component have no knowledge of the functional processes within the scope for another component, even though the two components exchange messages.

The functional user(s) of each component are determined by examining where the events occur that trigger functional processes in the component being examined. (Triggering events can only occur in the world of a functional user.)

Figure 4.1.8.2 illustrates the functional processes of two peer components and the data movements that they exchange.

3.3 Identifying objects of interest and data groups

3.3.1 Definitions and principles

This step consists in identifying the data groups referenced by the piece of software to be measured. To identify the data groups, especially in the business application software domain, it is usually helpful to identify the objects of interest and probably also their attributes. Data groups are moved in 'data movements' which are identified in the next chapter.

DEFINITION – Object of interest
Any 'thing' that is identified from the point of view of the Functional User Requirements. It may be any physical thing, as well as any conceptual object or part of a conceptual object in the world of the functional user about which the software is required to process and/or store data. NOTE: In the COSMIC method, the term 'object of interest' is used in order to avoid terms related to specific software engineering methods. The term does not imply 'objects' in the sense used in Object Oriented methods.

DEFINITION – Data group
A data group is a distinct, non empty, non ordered and non redundant set of data attributes where each included data attribute describes a complementary aspect of the same object of interest.

DEFINITION – Persistent storage
Persistent storage is storage which enables a functional process to store a data group beyond the life of the functional process and/or from which a functional process can retrieve a data group stored by another functional process, or stored by an earlier occurrence of the same functional process, or stored by some other process. NOTE 1: In the COSMIC model, because persistent storage is on the software side of the boundary, it is not considered to be a user of the software. NOTE 2: An example of 'some other process' would be the manufacture of read-only memory.

Once identified, each candidate data group must comply with the following principles:

PRINCIPLES – Data group
a) Each identified data group shall be unique and distinguishable through its unique collection of data attributes. b) Each data group shall be directly related to one object of interest in the software's Functional User Requirements c) A data group shall be materialized within the computer system supporting the software.

Once identified, each data group can be registered in an individual column in the Generic Software Model matrix (appendix A), under the corresponding label.

3.3.2 About the materialization of a data group

In practice, the materialization of a data group can take many forms, e.g.:

a) As a physical record structure on a persistent storage device (file, database table, ROM memory, etc.).

b) As a physical structure within the computer's volatile memory (data structure allocated dynamically or through a pre-allocated block of memory space).

c) As the clustered presentation of functionally related data attributes on an input/output device (display screen, printed report, control panel display, etc.).

d) As a message in transmission between a device and a computer, or over a network, etc

3.3.3 About the identification of objects of interest and data groups

The definition and principles of objects of interest and of data groups are intentionally broad in order to be applicable to the widest possible range of software. This quality sometimes results in it being difficult to apply the definition and principles when measuring a specific piece of software. Therefore, the following examples are designed to assist in the application of the principles to specific cases.

When faced with a need to analyze a group of data attributes that is moved in or out of a functional process or is moved by a functional process to or from persistent storage, *it is critically important* to decide if the attributes all convey data about a single 'object of interest', since it is the latter that determine the number of separate 'data groups' as defined by the COSMIC method. For instance, if the data attributes to be input to a functional process are attributes of three separate objects of interest, then we need to identify three separate 'Entry' data movements.

Objects of interest and data groups in the business applications domain

EXAMPLE 1: In the domain of business application software, an object of interest could be 'employee' (physical) or 'order' (conceptual), assuming the software is required to store data about employees or orders. In the case of 'order', it commonly follows from the FUR of multi-line orders that two objects of interest are identified: 'order' and 'order-line'. The corresponding data groups could be named 'order data', and 'order line data'.

Data groups are formed whenever there is an ad hoc enquiry which asks for data about some 'thing' about which data is not held on persistent storage, but which can be derived from data held on persistent storage. The Entry data movement in an ad hoc enquiry (the selection parameters to derive the required data) and the Exit data movement (containing the desired attributes) both move data groups about such a 'thing'. These are transient data groups that do not survive the execution of the functional process. They are valid data groups because they cross the boundary between the software and its user(s).

EXAMPLE 2: Given an ad hoc enquiry against a personnel database to extract a list of names of all employees aged over 35. The Entry moves a data group containing the selection parameters. The Exit moves a data group containing the single attribute 'name'; the 'object of interest (or 'thing') is 'all employees aged over 35'. It is important when recording the functional process to clearly name a transient data group in relation to its object of interest, rather than relating it to the object(s) of interest from which the result of the ad hoc enquiry is derived.

For a detailed discussion on methods of analyzing data to determine objects of interest and separate data groups, the reader is referred to the 'Guideline for Sizing Business Application Software using COSMIC'.

Objects of interest and data groups in the real-time software domain

EXAMPLE 3: Data movements which are Entries from physical devices typically contain data about the state of a single object of interest, such as whether a valve is open or closed, or indicate a time at which data in short-term, volatile storage is valid or invalid, or contain data that indicates a critical event has occurred and which causes an interrupt. Similarly an Exit command to switch a warning lamp on or off conveys data about a single object of interest.

EXAMPLE 4: A message-switch may receive a message data group as an Entry and route it forward unchanged as an Exit, as per the FUR of the particular piece of software. The attributes of the message data group could be, for example, 'sender, recipient, route_code and message_content'; the object of interest of the message is 'message'.

EXAMPLE 5: A common data structure, representing objects of interest that are mentioned in the Functional User Requirements, which can be maintained by functional processes, and which is accessible to most of the functional processes found in the measured software.

EXAMPLE 6: A reference data structure, representing graphs or tables of values found in the Functional User Requirements, which are held in permanent memory (ROM memory, for instance) and accessible to most of the functional processes found in the measured software.

EXAMPLE 7: Files, commonly designated as 'flat files', representing objects of interest mentioned in the Functional User Requirements, which are held on a persistent storage device.

3.3.4 Data or groups of data that are not candidates for data movements

Any data appearing on input or output screens or reports that are not related to an object of interest to a functional user should not be identified as indicating a data movement, so should not be measured.

EXAMPLE 1: Application-general data such as headers and footers (company name, application name, system date, etc) appearing on all screens

EXAMPLE 2: Control commands (a concept defined only in the business application domain) that enables a functional user to control their use of the software, rather than to move data, e.g. page up/down commands, clicking 'OK' to acknowledge an error message, etc – see further section 4.1.10.

The COSMIC Generic Software Model assumes that all manipulation of data within a functional process is associated with one of the four data movement types. – see section 4.1.6. Hence no movements or manipulation of data within a functional process may be identified as candidates for a data movement in addition to the Entries, Exits, Reads and Writes. (For examples of manipulation and movements of data that might be mis-interpreted as data movements, see section 4.1.4, principle c) for a Read, and section 4.1.5 principle d) for a Write).

3.3.5 The functional user as object of interest

In many simple real-time software systems, such as described in Example 3 in section 3.3.3 above, the physical device – a functional user – is indistinguishable from the object of interest of the data movement that it generates or receives. In such cases it adds little value to document an object of interest as if it were something separate from the functional user. The important point is to use these concepts, when helpful, to distinguish the separate data groups and hence separate data movements.

EXAMPLE: Suppose a temperature sensor 'A' that, when interrogated by a functional process, sends the current temperature to the process. The functional user is the sensor A; the Entry message name might be 'current temperature at A'; the object of interest of this message could also be regarded as 'sensor A'. Theoretically, the object of interest is not 'sensor A', but 'the material or object whose temperature is being sensed by sensor A'. In practice however it adds little value to document these fine distinctions and it may not be worthwhile to identify the object of interest separately.

3.4 Identifying data attributes (optional)

This section discusses the identification of data attributes referenced by the piece of software to be measured. In this version of the measurement method, it is not mandatory to identify the data attributes. However, it may be helpful to analyse and identify data attributes in the process of distinguishing data groups and objects of interest. Data attributes might also be identified if a sub-unit of the size measure is required, as presented in section 4.5 'Extending the COSMIC measurement method'.

3.4.1 Definition

DEFINITION – Data attribute
A data attribute is the smallest parcel of information, within an identified data group, carrying a meaning from the perspective of the software's Functional User

Requirements.

EXAMPLE 1: Data attributes in the context of the business applications domain are for example data elements registered in the data dictionary and data attributes appearing in a conceptual or logical data model.

EXAMPLE 2: Data attributes in the context of real-time application software are for example data attributes of a signal received from a sensor and data attributes of a message in transmission.

3.4.2 About the association of data attributes and data groups

In theory, a data group might contain only one data attribute if this is sufficient, from the perspective of the Functional User Requirements, to describe the object of interest. In practice, such cases occur commonly in real-time application software (e.g. the Entry conveying the tick of a real-time clock); they are less common in business application software.

4

THE MEASUREMENT PHASE

This chapter presents the rules and method for the measurement phase of the COSMIC measurement process. The general method for measuring a piece of software when its Functional User Requirements have been expressed in terms of the COSMIC Generic Software Model is summarized in Figure 4.0 below.

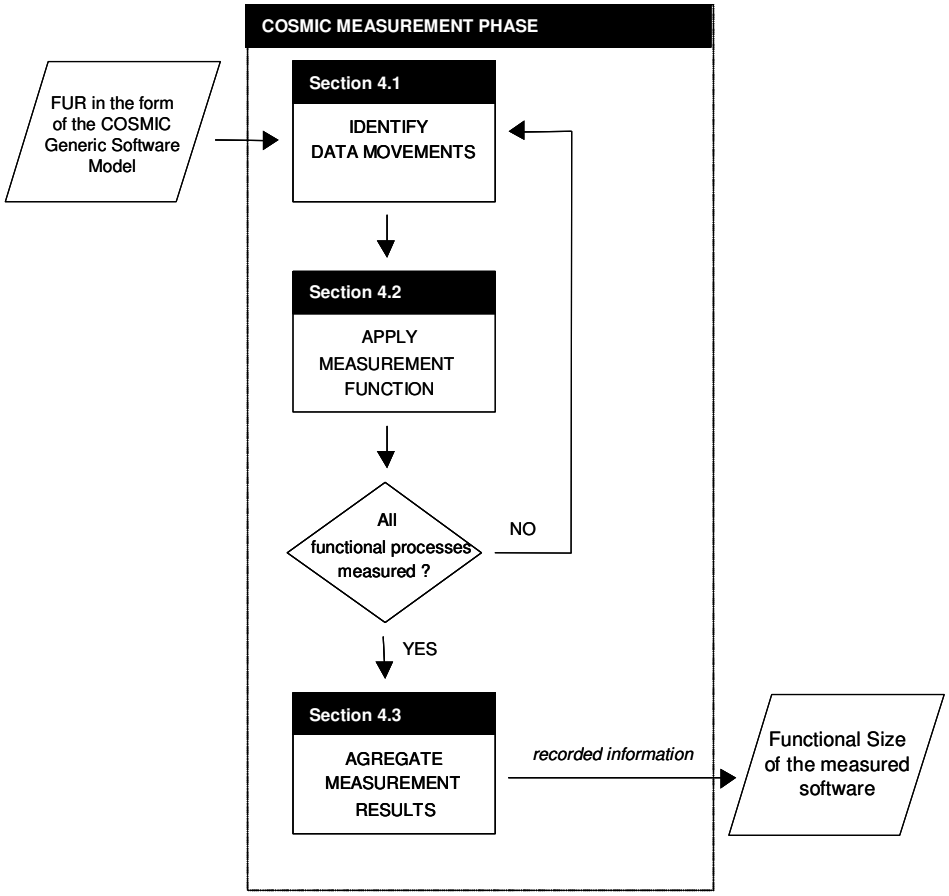


Figure 4.0 – General process for the COSMIC Measurement Phase

Each step in this method is the subject of a specific section of this chapter where the definitions and principles to apply are presented, along with some rules and examples.

4.1 Identifying the data movements

This step consists in identifying the data movements (Entry, Exit, Read and Write) of each functional process.

4.1.1 Definition of the data movement types

DEFINITION – Data movement
A base functional component which moves a single data group type. NOTE 1: There are four sub-types of data movement types: Entry, Exit, Read and Write (-types). NOTE 2: For measurement purposes, each data movement sub-type is considered to include certain associated data manipulations – see section 4.1.6 for details. NOTE 3: More precisely, it is an *occurrence* of a data movement, not a data movement *type*, that actually *moves* the data group *occurrences* (not *types*). This comment also applies to the definitions of Entry, Exit, Read and Write.

DEFINITION – Entry (E)
An Entry (E) is a data movement that moves a data group from a functional user across the boundary into the functional process where it is required. NOTE: An Entry is considered to include certain associated data manipulations (see section 4.1.6).

DEFINITION – Exit (X)
An Exit (X) is a data movement that moves a data group from a functional process across the boundary to the functional user that requires it. NOTE: An Exit is considered to include certain associated data manipulations (see section 4.1.6).

DEFINITION – Read (R)
A data movement that moves a data group from persistent storage within reach of the functional process which requires it. NOTE: A Read type is considered to include certain associated data manipulation (see section 4.1.6).

DEFINITION – Write (W)
A data movement that moves a data group lying inside a functional process to persistent storage. NOTE A Write type is considered to include certain associated data manipulation (see section 4.1.6).

Figure 4.1.1, below, illustrates the overall relationship between the four types of data movement, the functional process to which they belong and the boundary of the measured software.

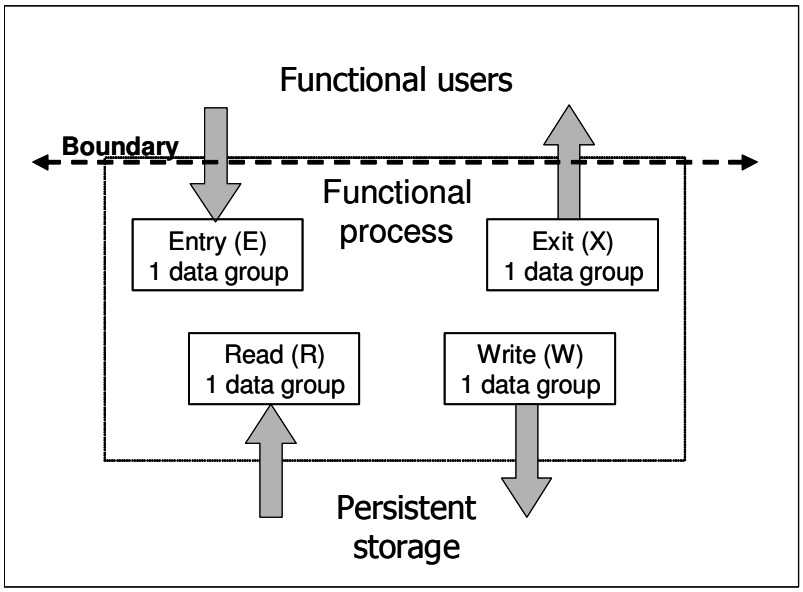


Figure 4.1.1 – The four types of data movement and their relationship with the functional process and data groups

4.1.2 Identifying Entries (E)

Once identified, a candidate Entry data movement must comply with the following principles:

PRINCIPLES – Entry (E)
a) An Entry shall move a single data group describing a single object of interest from a functional user across the boundary and into the functional process of which the Entry forms part. If the input to a functional process comprises more than one data group, identify one Entry for each unique data group in the input. (See also section 4.1.7 on 'Data movement uniqueness'.)
b) An Entry shall not exit data across the boundary, or read or write data.
c) Where a functional process needs to obtain data from a functional user but the latter does not need to be told what data to send, or the functional user is incapable of reacting to any incoming message, identify one Entry to the functional process for obtaining the data. Any message from the functional process to the functional user seeking to retrieve the data shall not be counted as an Exit in these cases. However, where a functional process must obtain some data from a functional user and the functional process must provide the functional user with data which the latter requires to fulfill the request, count one Exit for the request and one Entry for the return of the requested data (see further section 4.1.9).

The following rules help to confirm the status of a candidate Entry data movement:

RULES – Entry (E)
a) The data group of a triggering Entry may consist of only one data attribute which simply informs the software that 'an event Y has occurred'. Very often, especially in business application software, the data group of the triggering Entry has several data attributes which inform the software that 'an event Y has occurred and here is the data about that particular event'.

b) Clock-ticks that are triggering events shall always be external to the software being measured. Therefore, for example, a clock-tick event occurring every 3 seconds shall be associated with an Entry moving a data group of one data attribute. Note that it makes no difference whether the triggering event is generated periodically by hardware or by another piece of software outside of the boundary of the software being measured.

c) Unless a specific functional process is necessary, obtaining the time from the system's clock shall not be considered to cause an Entry.

d) If an occurrence of a specific event triggers the Entry of a data group comprising up to 'n' data attributes of a particular object of interest and the FUR allows that other occurrences of the same event can trigger an Entry of a data group which has values for attributes of only a sub-set of the 'n' attributes of the object of interest, then one Entry shall be identified, comprising all 'n' data attributes.

EXAMPLE illustrating rule c): when a functional process writes a time stamp, no Entry is identified for obtaining the system's clock value.

Once identified, each Entry data movement can be registered by marking the corresponding cell of the Generic Software Model matrix (appendix A) with an 'E'.

4.1.3 Identifying Exits (X)

Once identified, a candidate Exit data movement must comply with the following principles:

PRINCIPLES – Exit (X)

a) An Exit shall move a single data group describing a single object of interest from the functional process of which the Exit forms part across the boundary to a functional user. If the output of a functional process comprises more than one data group, identify one Exit for each unique data group in the output. (See also section 4.1.7 on 'Data movement uniqueness'.)

b) An Exit shall not enter data across the boundary, or read or write data

The following rules might be useful to confirm the status of a candidate Exit data movement:

RULES – Exit (X)

a) All messages generated and output by software without user data (e.g. error messages) shall be considered to be values of one attribute of one object of interest (which could be named 'error indication'). Therefore, a single Exit shall be identified to represent all these message occurrences within each functional process where they are required by the FUR.

b) If an Exit of a functional process moves a data group comprising up to 'n' data attributes of a particular object of interest and the FUR allows that the functional process may have an occurrence of an Exit that moves a data group which has values for attributes of only a sub-set of the 'n' attributes of the object of interest, then one Exit shall be identified, comprising all 'n' data attributes.

Examples of rule a) above are as follows:

EXAMPLE 1: In a human-computer dialogue, examples of error messages occurring during validation of data being entered could be 'format error', 'customer not found', 'error: check box indicating terms and conditions have been read', 'credit limit exceeded', etc. All such error messages should be considered as occurrences of one Exit in each functional process where such messages occur (which could be named 'error messages').

EXAMPLE 2: Error messages output to the human users but not generated by the application software should be completely ignored in the measurement of the application. An example of such a message passed on from the operating system could be 'printer X is not responding'.

EXAMPLE 3: In a human-computer dialogue, if a message is output in error situations but it contains functional user data, then it should be counted as an Exit in the functional process where it occurs. An example of such a message could be 'Warning: the amount you wish to withdraw exceeds your overdraft limit by $100' (where the $100 is a calculated variable). In this example, the Exit contains a data group about the customer's bank account.

EXAMPLE 4: In a real-time system, a functional process that periodically checks the correct functioning of all hardware devices might issue a message that reports 'Sensor X has failed', where 'X' is a variable. This message should be identified as one Exit in that functional process (regardless of the value of 'X').

EXAMPLE 5: Consider functional processes A and B. 'A' can potentially issue 2 distinct confirmation messages and 5 error messages to its functional users and 'B' can potentially issue 8 error messages to its functional users. In this example, one Exit shall be identified within functional process 'A' (handling 5+2=7 messages) and a separate Exit shall be identified within functional process 'B' (handling 8 messages).

Once identified, each Exit data movement can be registered by marking the corresponding cell of the Generic Software Model matrix (appendix A) with an 'X'.

4.1.4 Identifying Reads (R)

Once identified, a candidate Read data movement must comply with the following principles:

PRINCIPLES –Read (R)
a) A Read shall move a single data group describing a single object of interest from persistent storage to a functional process of which the Read forms part. If the functional process must retrieve more than one data group from persistent storage, identify one Read for each unique data group that is retrieved. (See also section 4.1.7 on 'Data movement uniqueness'.)
b) A Read shall not receive or exit data across the boundary or write data.
c) During a functional process, movement or manipulation of constants or variables which are internal to the functional process and that can be changed only by a programmer, or computation of intermediate results in a calculation, or of data stored by a functional process resulting only from the implementation, rather than from the FUR, shall not be considered as Read data movements.
d) A Read data movement always includes any 'request to Read' functionality (so a separate data movement shall never be counted for any 'request to Read' functionality). See also section 4.1.9.

Once identified, each Read data movement can be registered by marking the corresponding cell of the Generic Software Model matrix (appendix A) with an 'R'.

4.1.5 Identifying Writes (W)

Once identified, the candidate Write data movement must comply with the following principles:

PRINCIPLES – Write (W)
a) A Write shall move a single data group describing a single object of interest from the functional process of which the Write forms part to persistent storage. If the functional process must move more than one data group to persistent storage, identify one Write for each unique data group that is moved to persistent storage. (See also section 4.1.7 on 'Data movement uniqueness'.)
b) A Write shall not receive or exit data across the boundary, or read data.
c) A requirement to delete a data group from persistent storage shall be measured as a single Write data movement.
d) During a functional process, movement or manipulation of data that does not persist when the functional process is complete, or updating variables which are internal to

the functional process or producing intermediate results in a calculation shall not be considered as Write data movements.

Once identified, each Write data movement can be registered by marking the corresponding cell of the Generic Software Model matrix (appendix A) with a 'W'.

4.1.6 On the data manipulations associated with data movements

Sub-processes are, as defined in principle (d) of the Generic Software Model (see section 1.4), either data movements, or data manipulations. However, by a current COSMIC convention (see principle (j) of the Generic Software Model), the separate existence of data manipulation sub-processes is not recognized.

DEFINITION – Data manipulation
Anything that happens to data other than movement of the data into or out of a functional process, or between a functional process and persistent storage

The following principle determines how the COSMIC method deals with data manipulation.

PRINCIPLE – Data manipulation associated with data movements
All data manipulation in a functional process shall be associated with the four types of data movement (E, X, R, and W). By convention, the data movements of a functional process are assumed also to represent the data manipulation of the functional process

The need to define which kinds of data manipulation are associated with which types of data movement arises only when measuring *changes* to software (see section 4.4). A typical required change affects both the attributes moved and the manipulation associated with a particular data movement, but it *may* affect only the manipulation of the data, not the movement of the data. Such a change still needs to be identified and measured. So when there is any requirement to change some data manipulation in a functional process, the measurer needs to identify which data movement is associated with the change to the data manipulation.

Below are general guidelines to identify the data manipulation represented by each of the data movements.

Entry data movement

An Entry includes all data manipulation

- to enable a data group to be entered by a functional user (e.g. formatting and presentation manipulations) and to be validated
- but not any manipulations that involve another data movement, nor any manipulations after the data group has been entered and validated.

EXAMPLE: An Entry includes all manipulation, formatting and presentation on a screen of the entered data EXCEPT any Read(s) that might be required to validate some entered codes or to obtain some associated descriptions.

An Entry data movement includes any 'request to enter' functionality except when the functional process needs to inform the functional user what data to send (see section 4.1.9 for what data to send and 4.1.10 for the treatment of an empty input screen).

Exit data movement

An Exit includes all data manipulation

- to create the data attributes of a data group to be output and/or

- to enable the data group to be output (e.g. formatting and presentation manipulations) and to be routed to the intended functional user
- but not any manipulations that involve another data movement.

EXAMPLE: An Exit includes all processing to format and prepare for printing some data attributes, including the human-readable field headings[13] EXCEPT any Read(s) or Entries that might be required to supply the values or descriptions of some of the printed data attributes.

Read data movement

A Read includes all processing and/or computation needed in order to retrieve a data group from persistent storage but not any manipulations that involve another type of data movement nor any manipulations after the Read is successfully completed.

EXAMPLE: A Read includes all mathematical computation and logical processing required to retrieve a data group from persistent storage but not the manipulation of those attributes after the data group has been obtained.

A Read also always includes any 'request to Read' functionality (see section 4.1.9).

Write data movement

A Write includes all processing and/or computation to create a data group to be written but not any manipulations that involve another type of data movement nor any manipulations after the Write is successfully completed.

EXAMPLE: A Write includes all mathematical computation and logical processing required to create or to update a data group to be written, or to be deleted EXCEPT any Reads or Entries that might be required to supply the value of any data attributes included in the group to be written or deleted.

4.1.7 Data movement uniqueness and possible exceptions[14]

The Generic Software Model assumes that *normally* in any one functional process *all* data describing any one object of interest that is required by that functional process is input in one Entry data movement type and/or read in one Read data movement type and/or written in one Write data movement type and/or output in one Exit data movement type. The model further assumes that all data manipulation resulting from all possible values of the data attributes of a data group that is moved is associated with the one data movement.

EXAMPLE illustrating this latter assumption; Consider two occurrences of a given functional process (-type). Suppose that in the first occurrence the values of some attributes to be moved lead to a data manipulation sub-process (-type) 'A' and that in another occurrence of the same functional process the attribute values lead to a different data manipulation sub-process (-type) 'B'. In such circumstances, both data manipulation sub-processes 'A' and 'B' should be associated with the same one data movement and hence only the one data movement should normally be identified and counted in that functional process.

There can, however, be *exceptional* circumstances in which *different* data group types describing a given object of interest may be required (in the FUR) to be moved in a data movement of the same type (E, R, W, X) in the same functional process. Alternatively, and again exceptionally, the same data group may be required to be moved in the same data movement type (E, R, W or X) in the same functional process, but with different associated data manipulation.

---

[13] This example applies when measuring application software for use by humans, regardless of the domain. It would obviously not apply when measuring the size of re-usable objects which support the display of individual field headings on input or output screens.

[14] This section was entitled 'De-duplication of data movements' in version 2.2 of the Measurement Manual. The term 'de-duplication' was felt to be unhelpful, so the terminology has been changed

The following rules and examples cover the normal situation, possible valid exceptions and examples that might appear valid, but which are not valid.

RULE – Data movement uniqueness and possible exceptions
a) Unless the Functional User Requirements specify otherwise, all data attributes describing any one object of interest that are required to be entered into one functional process, and all associated data manipulation shall be identified and counted as one Entry (type). (Note: A functional process may, of course, be required to handle multiple Entry types, each moving a data group describing a different object of interest (type).) The same equivalent rule applies to any Read, Write or Exit data movement in any given functional process
b) More than one Entry data movement (type), each moving a data group describing the same object of interest (type) in a given functional process (type) may be identified and counted if there is a Functional User Requirement for these multiple Entries. Similarly, more than one Entry (type) moving the same data group (type) in the same functional process, but each with different associated data manipulation (types), may be identified and counted if there is a Functional User Requirement for these multiple Entries. Such FUR may arise when, in one functional process, the multiple Entries originate from different functional users who enter different data groups (each describing the same object of interest). The same equivalent rule applies to any Read, Write or Exit data movement in any given functional process
c) Repeated *occurrences* of a data movement type (i.e. each occurrence moving the same data group type with the same data manipulation) shall not be identified and counted more than once in any one functional process
d) If the multiple *occurrences* of a data movement type in a given functional process differ in their associated data manipulation because different *values* of the data attributes of the data group moved result in different processing paths being followed, the data movement type shall not be identified and counted more than once in that process.

The following examples illustrate the above rules.

EXAMPLE 1 for rule a): In any one functional process, all Reads of data describing a particular object of interest can be considered logically to return all the required attributes describing that object of interest (i.e. the whole of the required 'state vector' of that object of interest). Hence normally only one Read of any data about any one object of interest is functionally needed and should be identified in any one functional process

EXAMPLE 2 for rules a) and b): Following on from Example 1, since any data that is read must have been made persistent by a Write statement, the normal case (Rule a) is that one Write would be identified that moves a data group containing all the data attributes of an object of interest required to be made persistent in a given functional process. Exceptionally, however there may exist FUR for a single functional process to write two different data groups describing the same object of interest, e.g. for later use by two different functional users in other functional processes. An example where rule b) applies would be where a single functional process A is required to extract two sub-sets of data from a bank's current account files for later use by separate programs. The first sub-set is 'overdrawn account' details' (which includes the negative balance attribute). The second sub-set is 'high value account' details' (which only has the account holder's name and address, intended for a marketing mail-shot). Functional process A will have two Writes, one for each sub-set.

EXAMPLE 3 for rule b): It is possible that there exist FUR for a single functional process to produce two or more Exits moving different data groups describing the same object of interest, intended for different functional users. For example

when a new employee joins a company a report is produced for the employee to sign off his personal data as valid and a message is sent to Security to authorize the employee to enter the building.

EXAMPLE 4 for rule c): Suppose a Read is required in the FUR that in practice requires many retrieval occurrences, as in a search through a file. For sizing purposes, identify only one Read.

EXAMPLE 5 for rule c): Suppose in a real-time functional process, the FUR requires that data must be entered from a given functional user, e.g. a hardware device, twice at a fixed time interval in order to measure a rate of change or to check if a value has changed during the process. Provided the two Entries are identical in terms of the data group moved and the associated data manipulation, only one Entry should be identified. (Refer to section 4.1.6 for the types of data manipulation that are considered to be associated with an Entry.)

EXAMPLE 6 for rule c): See section 4.1.2, Rule d) for Entries, and section 4.1.3, Rule b) for Exits

EXAMPLE 7 for rule d): Suppose a functional process is required which provides various data manipulation options depending on the values of the data attributes of an Entry. For sizing purposes, identify only one Entry.

EXAMPLE 8: Suppose a Read of a data group is required in the FUR, but the developer decides to implement it by two commands to retrieve different sub-sets of data attributes of the same object of interest from persistent storage at different points in the functional process. Identify only one Read.

4.1.8 When a functional process moves data to or from persistent storage

This section explains the data movements involved when a functional process of a piece of application software is required to move data to or from persistent storage, when the storage is either local or remote. The examples also show how the application's storage needs are handled by other software that supports the application in another layer, such as the persistent storage device-driver software.

The examples illustrate the application of principle (g) of the Software Context Model and the principles of the Generic Software Model. The key to understanding the examples is that these principles must be applied separately to each piece of software that must be measured.

The first example deals with the data movements of an enquiry in an application where the requirement to retrieve persistent data is handled by local device-driver software in another layer. The second example shows how the data movements differ when the requirement to retrieve is first satisfied by a peer piece of software of the application.

In practical terms, the examples are applicable when the task is to measure two pieces of software that have either a hierarchical relationship (i.e. when the two pieces are in different layers) or have a client-server relationship (i.e. when the two pieces are peers of each other). The examples show how the data movements that are physically exchanged between the two pieces of software to be measured are modelled.

The examples are illustrated using the conventions of Message Sequence Diagrams. The notation of these diagrams is as follows:

- A bold vertical arrow pointing downwards represents a functional process.
- Horizontal arrows represent data movements, labeled E, X, R or W for Entry, Exit, Read and Write, respectively. Entries and Reads are shown as arrows incoming to the functional process and Exits and Writes as outgoing arrows; they appear in the sequence required, from top to bottom, of the functional process.
- A vertical dotted line represents a boundary.

EXAMPLE 1: When a functional process must move data to or from local persistent storage

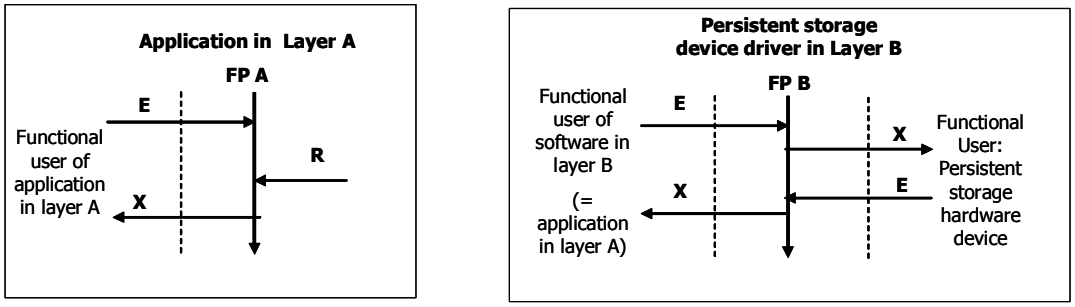
This example involves two pieces of software, namely a piece of application software 'A' and a separate piece of software 'B' that is the device driver for a persistent storage device, which the application software uses. (We ignore the probable presence of an operating system for simplicity; the operating system effectively transmits application requests to the device driver software and returns the results of requests.)

The concept of layers tells us that the two pieces of software are in different layers: the application layer and the device driver layer. Physically, there is a hierarchical relationship between the two pieces and (ignoring the operating system) a physical interface between software in the two layers, as shown for example in Fig. 2.2.4.1.

Normally, the FUR of the application 'A' will specify functional processes that include the need for Reads and Writes to persistent storage but will not be concerned with how those Reads and Writes are handled by other infrastructure software.

Applying the COSMIC models to the two pieces of software, the functional users of the software A in the application layer could be, for example, human users, whilst the functional user of the software B in the driver layer is the piece of application software A (ignoring the operating system).

Suppose an enquiry functional process 'FP A' of the software A in the application layer requires a Read data movement. Fig. 4.1.8.1 (a) shows the COSMIC model of the application enquiry. The physical retrieval of the required data from persistent storage is handled by a functional process 'FP B' of the software B in the device driver layer. Figure 4.1.8.1 (b) shows the model for this functional process of the device driver.



Figures 4.1.8.1 (a) and (b) Solution for a Read issued by software 'A' in the application layer to software 'B' in the device driver layer

Fig. 4.1.8.1 (a) shows the enquiry triggered by an Entry, followed by a Read and then an Exit with the enquiry result. FP A has no knowledge of where the data is retrieved from, nor that in practice the Read is delegated to some device driver software.

Fig. 4.1.8.1 (b) shows that functionally the Read request of the application A is received as a triggering Entry to the functional process FP B, which then retrieves the requested data from the physical persistent storage device via an Exit/Entry pair and returns the data to the application as an Exit. The application A and the persistent storage hardware device are thus functional users of the device driver software B.

The apparent mis-match between the number of data movements of the 'Read' of the software in the application layer and the 'Entry/Exit pair' of software in the device driver layer is due to the fact that by convention, a Read data movement is considered to include any 'request to read' functionality.

Exactly analogous models would apply if the functional process FP A were required to make some data persistent via a Write data movement. In this example, the Exit of the functional process FP B of the device driver software to Application A would contain any 'return code' or error message.

EXAMPLE 2: When a functional process must obtain data from a peer piece of software

In this example, the two peer pieces of software to be measured are assumed to have a 'client/server' relationship, i.e. where one piece, the client, obtains services and/or data from the other piece, the 'server, in the same layer'. Fig. 4.1.8.2 shows an example of such a relationship, in which the two pieces are major (peer) components of the same application. The same relationship would exist and the same diagram would apply if the two pieces were separate peer applications, where one needed to obtain data from the other.

Physically, the two peer components could execute on separate processors; in such a case they would exchange data via the respective operating systems and any other intermediate layers of their processors in a software architecture such as shown in Fig. 2.2.4.1. But logically, applying the COSMIC models, the two components exchange data via 'Entry/Exit pairs'. All intervening software and hardware is ignored in this model (as also shown in the right-hand side of Fig. 3.1.1).

Fig. 4.1.8.2 shows that a functional process FP C1 of the client component C1 is triggered by an Entry from its functional user which comprises, for example, the parameters of the enquiry. The FUR of component C1 will recognize that this component must ask the server component C2 for the required data, and must tell it what data is required.

To obtain the required data, therefore, FP C1 issues an Exit to component C2 containing the enquiry request parameters. Component C1 is now a functional user of component C2; hence a boundary exists between the two components. This Exit data movement crosses the boundary between C1 and C2 and so becomes the triggering Entry of a functional process FP C2 in the component C2. The functional process FP C2 of component C2 obtains the required data via a Read, and sends the data back to C1 via an Exit. Functional process FP C1 of component C1 receives this data movement as an Entry. FP C1 then passes the data on as an Exit to satisfy the enquiry of its functional user. This Example 2 enquiry therefore requires 7 data movements to satisfy the enquiry request in the application layer. This compares with the 3 data movements (1 x E, 1 x R and 1 x X) that would have been required in the application layer if component C1 had been able to retrieve the data from 'local' persistent storage as shown in Fig. 4.1.8.1 (a).

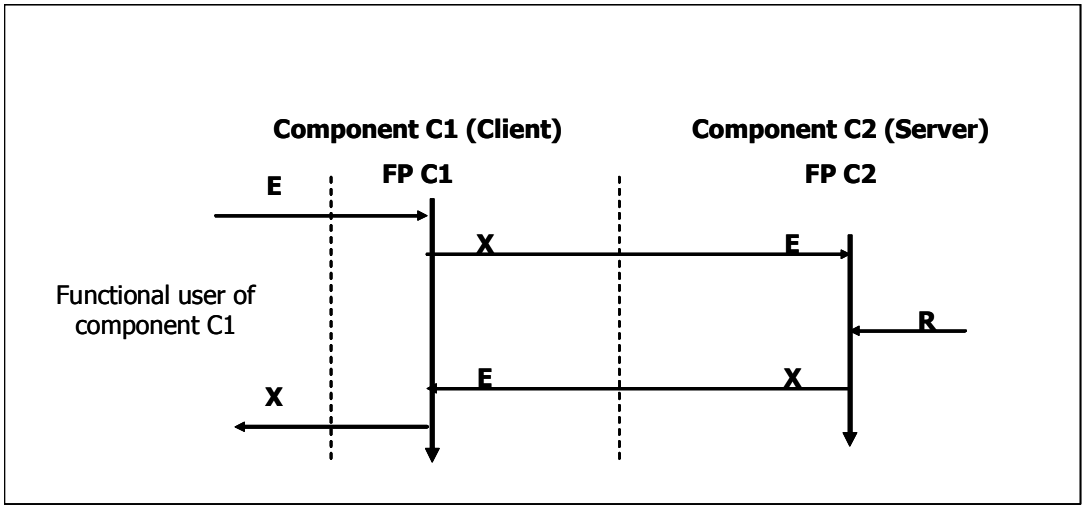


Figure 4.1.8.2 Data exchanges between peer components

Component C2 will probably, of course, use the services of some persistent storage device driver software in a lower layer of the software architecture to retrieve the data from the hardware, as in Example 1.

Comparing Examples 1 and 2, we see that in Example 1, the models of the application A and the device driver B cannot be combined as in Example 2. This is because a Read does not cross a boundary. Fig. 4.1.8.1 (b) shows that the application A is a functional user of the device driver software B. But the reverse is not true, thus demonstrating the hierarchical nature of software in different layers.

In contrast, Fig. 4.1.8.2 can show the two components in one model because they exchange data as 'peers' or equals in the same layer. Component C1 is a functional user of component C2, and vice versa, and they share a common boundary.

Note that in these two examples we have ignored, for simplicity, the generation of an Exit error message by the application A or the component C1 (in addition to the Exit containing the result of the enquiry) which might result from a 'return code' accompanying the Read data movement.

4.1.9 When a functional process requires data from a functional user

As per principle c) for an Entry (see section 4.1.2), if a functional process must obtain data from a functional user there are two cases. If the functional process does not need to tell the functional user what data to send, a single Entry is sufficient (per object of interest). If the functional process needs to tell the functional user what data to send, an Exit/Entry pair is necessary. The following rules apply:

RULES – When a functional process requires data from a functional user

a) A functional process shall obtain a data group via an Entry data movement from a functional user, *when it does not need to tell the functional user what data to send*, as in any of the following four cases:

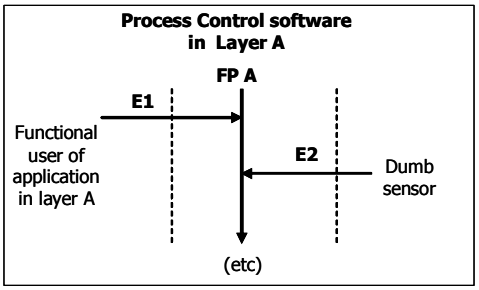
- when a functional user sends a triggering Entry which initiates the functional process;
- when a functional process, having received a triggering Entry, waits, expecting the arrival of a further Entry from the functional user (may occur when a human functional user enters data to business application software);
- when a functional process, having started, requests the function'l user, 'send me your data now, if you'have any' and the functional user sends its data;
- when a functional process, having started, inspects the state of a functional user and retrieves the data it requires.

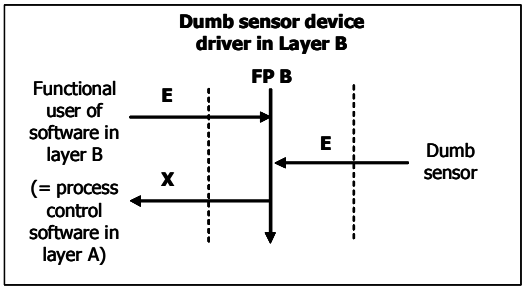
In the latter two cases (typically occurring in real-time 'polling' software), by convention no Exit from the functional process shall be identified to obtain the required data. The functional process merely needs to send a prompt message to a functional user and the functionality of that prompt message is considered to be part of the Entry. The functional process knows what data to expect. Only one Entry is needed for this case.

b) Where a functional process needs to obtain the services of a functional user (for instance to obtain data) and *the functional user needs to be told what to send* (typically where the functional user is another piece of software outside the scope of the software being measured), a pair of Exit/Entry data movements shall be identified. The Exit contains the request for the specific data; the Entry contains the returned data.''

EXAMPLE 1 of rule a), first and second bullets: Where a functional process provides a screen for data entry by a human functional user that is formatted but otherwise 'blank' except for possible default values, as in an on-line business application, the provision of the 'blank' screen is not counted as an Exit. Only the filled-in screen is counted as one or more Entries. (See also section 4.1.10.)

EXAMPLE 2 of rule a), third or fourth bullet: Suppose a functional process of a real-time process control software system is required to poll an array of identical dumb sensors. The functional process obtains its data via one Entry (type). (Since the sensors are identical only one Entry (type) is identified and counted although there are multiple occurrences.) Suppose further that the Entry must in practice be passed to a piece of device driver software in a lower layer of the software architecture, which physically obtains the required data from the sensor as illustrated in Fig. 2.2.3.2. The functional processes of the process control software and of the device driver software for the dumb sensors would be as shown in Figs. 4.1.9.1 (a) and (b) below.





Figures 4.1.9.1 (a) and (b) Solution for an Entry issued by software 'A' in the process control application layer to software 'B' in the dumb sensor device driver layer

Fig. 4.1.9.1 (a) shows that the process control software functional process 'FP A' is triggered by an Entry 'E1' e.g. from a clock tick. The functional process then obtains Entry 'E2' from the dumb sensor array to receive the multiple occurrences of the sensor readings. The dumb sensors are also functional users of the process control software.

Fig. 4.1.9.1 (b) shows that the software that drives the dumb sensor devices receives an Entry (probably in practice via an operating system) as the trigger of a functional process 'FP B'. This functional process obtains an Entry from its functional user (type), the dumb sensor (type) to obtain the sensors' data which is passed back to the process control software as an Exit. The process control software functional process then continues with its processing of the sensor data. Again, the fact that there are multiple occurrences of this cycle of gathering data from each of the identical sensors is irrelevant to the model.

The apparent mis-match between the one Entry of the process control software and the Entry/Exit pair of the device driver software is due to the convention that an Entry is considered to include any 'request to enter' functionality in this case where the functional user has no capability of responding to a message from a functional process.

EXAMPLE 3 of rule b): Suppose a functional process sends to one of its functional users such as an 'intelligent' hardware device or another peer piece of software some parameters for an enquiry or the parameters for a calculation, or some data to be compressed. The response from the functional user is obtained via an Exit/Entry pair, as described in section 4.1.8, Example 2.

4.1.10 Control commands

A 'control command' is a special category of data movement that is recognized only in the business application domain and which must be ignored when measuring a functional size. The definition is:

DEFINITION – Control command
A control command is a command that enables a functional user to control their use of the software but which does not involve any movement of data about an object of interest. NOTE: The term 'control command' is used ONLY in the context of measuring business application software. In this context, a control command is not a data movement because the command does not move data about an object of interest. Examples are 'page up/down' commands; hitting a Tab or Enter key, clicking on the 'OK' to confirm a previous action, etc.

RULE – Control commands in the business application domain
In the business application domain 'control commands' shall be ignored as they do not involve any movement of data about an object of interest.

EXAMPLES: Control commands in the business application domain are the functions that enable a functional user to control the display (or not) of a header or of sub-totals that have been calculated, navigate up and down and between physical screens, click 'OK' to acknowledge an error message or to confirm some entered data, etc. Control commands therefore also include menu commands that enable the functional user to navigate to one or more specific functional processes but which do not themselves initiate any one functional process, and commands to display a blank screen for data entry.

N.B. Outside the business application domain, the concept of a 'control command' has no special meaning and any signal or movement of data about an object of interest coming from a functional user must be accounted for, i.e. must be measured.

4.2 Applying the measurement function

This step consists in applying the COSMIC measurement function to each of the data movements identified in each functional process.

DEFINITION – COSMIC measurement function
The COSMIC measurement function is a mathematical function which assigns a value to its argument based on the COSMIC measurement standard. The argument of the COSMIC measurement function is the data movement.

DEFINITION – COSMIC measurement standard
The COSMIC measurement standard, 1 CFP (Cosmic Function Point) is defined as the size of one data movement.

According to this measurement function, each instance of a data movement (Entry, Exit, Read or Write) that is required to be added, changed or deleted and that has been identified according to section 4.1 receives a numerical size of 1 CFP.

4.3 Aggregating measurement results

This step consists in aggregating the results of the measurement function, as applied to all identified data movements, into a single functional size value. This step is accomplished according to the following rules.

4.3.1 General rules of aggregation

RULES – Aggregating measurement results

a) For any functional process, the functional sizes of individual data movements shall be aggregated into a single functional size value in units of CFP by arithmetically adding them together.

$$\text{Size (functional process}_i) = \Sigma \text{ size(Entries}_i) + \Sigma \text{ size(Exits}_i) + \Sigma \text{ size(Reads}_i) + \Sigma \text{ size(Writes}_i)$$

b) For any functional process, the functional size of changes to its Functional User Requirements shall be aggregated from the sizes of the data movements that have been added, modified or deleted in the functional process to give a size of the change in units of CFP, according to the following formula.

$$\text{Size (Change(functional process}_i)) = \Sigma \text{ size (added data movements}_i) + \Sigma \text{ size (modified data movements}_i) + \Sigma \text{ size (deleted data movements}_i)$$

For more on aggregating functional size, see section 4.3.2. For measuring the size of the changed software, see section 4.4.

c) The size of a piece of software within a defined scope shall be obtained by aggregating the sizes of the functional processes for the piece, subject to rules e) and f) below

d) The size of any change to a piece of software within a defined scope shall be obtained by aggregating the sizes of all changes to all functional processes for the piece, subject to rules e) and f) below

e) Sizes of pieces of software or of changes to pieces of software within layers may be added together only if measured at the same functional process level of granularity of their FUR.

f) Sizes of pieces of software and/or changes in the sizes of piece of software within any one layer or from different layers shall be added together only if it makes sense to do so, for the purpose of the measurement.

g) The size of a piece of software cannot be obtained by adding up the sizes of its components (regardless of how the item is decomposed) *unless* the size contributions of inter-component data movements are eliminated.

h) If the COSMIC method is extended locally (for example to measure some aspect of size not covered by the standard method), then the size measured via the local extension must be reported separately as described in section 5.1 and may NOT be added to the size obtained by the standard method, measured in CFP (see further in section 4.5)

EXAMPLE 1 for rules b) and c): A requested change to a piece of software might be: 'add one new functional process of size 6 CFP, and in another functional process add one data movement, make modifications to three other data movements and delete two data movements.' The total size of the requested change is 6 + 1 + 3 + 2 = 12 CFP.

EXAMPLE 2 for rule f): If various major parts of a piece of software are developed using different technologies, by different project sub-teams, there may be no practical value in adding their sizes together.

EXAMPLE 3 for rule g): If a piece of software is

- first measured 'as a whole', i.e. all within one scope
- then secondly the size of each of its components is measured separately, i.e. each with its own scope,

then the total size from adding up the size of all the separate components (in the second case) will exceed the size when measured 'as a whole' (in the first case) due to the contribution to size of all the inter-component data movements. These inter-component data movements are not visible when the piece is measured 'as a whole' and must therefore be eliminated to obtain the size of 'the whole'. See also the example in the section on measuring at varying levels of granularity in pure software architectures in the 'Advanced and Related Topics' document.

It is to be noted that, *within* each identified layer, the aggregation function is fully scalable. Therefore a sub-total can be generated for individual functional processes or for all the software within a layer, depending on the purpose and scope of each measurement exercise and subject to rules d), e) and f) above.

4.3.2 More about functional size aggregation

In a context where functional size is to be used as a variable in a model, to estimate effort for instance, and the software to be sized has more than one layer or peer component, aggregation will typically be performed per layer or per peer component since they are often not implemented with the same technology.

EXAMPLE 1: Consider software where the application layer is to be implemented using a 3GL and a set of existing libraries, while a driver layer might be implemented using assembly language. The unit effort associated with the construction of each layer will, most probably, be different, and, consequently, an effort estimate will be prepared separately for each layer based on its respective size.

EXAMPLE 2: If a project team has to develop a number of major pieces of software and is interested in its overall productivity, it can add together the work-hours needed to develop each piece. Similarly, it can add together the sizes of the major pieces it has developed if (but only if) those sizes satisfy the rules given above.

The reason that sizes of major pieces of software from different layers of a standard layered architecture, measured at the same functional process level of granularity, may be added together is that such an architecture has a coherently defined set of functional users. Each layer is a functional user of the 'lower' layers that it uses and any piece of software in a layer may be a functional user of any its peer pieces of software. The requirements of such an architecture impose that the FUR of the various pieces must exchange messages. It is therefore only logical and reasonable that the sizes of the various pieces may be added together, always subject to rules d), e) and f) above. However, in contrast, the size of a major piece of software may *not* be obtained by adding up the sizes of its component re-usable objects unless the inter-object data movements are eliminated, as per rule f) above.

Aggregating the measurement results by type of data movement might be useful for analyzing the contribution of each type to the total size of a layer and might thus help characterize the functional nature of the measured layer.

4.4 More on measurement of the size of changes to software

A 'functional change' to existing software is interpreted in the COSMIC method as 'any combination of additions of new data movements or of modifications or deletions of existing data movements'. The terms 'enhancement' and 'maintenance'[15] are often used for what we here call a 'functional change'.

The need for a change to software may arise from either

- a new FUR (i.e. only additions to the existing functionality), or
- from a change to the FUR (perhaps involving additions, modifications and deletions) or
- from a 'maintenance' need to correct a defect

The rules for sizing any of these changes are the same but the measurer is alerted to distinguish the various circumstances when making performance measurements and estimates.

When a piece of software is completely replaced, for instance by re-writing it, with or without extending and/or omitting functionality, the functional size of this change is the size of the replacement software, measured according to the normal rules for sizing new software. This case will not be considered further in this section. The measurer should be aware, however, of the need when making performance measurements or estimates to distinguish between projects to develop entirely new software and projects to 're-develop' or 'replace' existing software.

Often, an obsolete part of an application is deleted ('disconnected' would be a better description) by leaving the program code in place and by just removing the contact with the obsolete functionality. When the functionality of the obsolete part amounts to 100 CFP but the part can be disconnected by changing, say, 2 data movements, 100 and not 2 data movements shall be identified as the size of the functional change. We measure the size of the requirement, not the size that was implemented.[16]

Note the difference between the size of the functional change (discussed here) and the change in the functional size of the software. Usually, they are different. The size of the latter is addressed in section 4.4.3.

4.4.1 Modifying functionality

Any data movement of a given type (E, X, R and W) involves two types of functionality: it moves a single data group and it has some associated data manipulation (for the latter, see section 4.1.6). Hence for measurement purposes a data movement is considered to be functionally modified if

- the data group moved and/or
- its associated data manipulation

are modified in any way.

A data group is modified if

---

[15] A normal measurement convention is that the functional size of a piece of software does not change if the software must be changed to correct a defect so as to bring the software in line with its FUR. The functional size of the software does change if the change is to correct a defect in the FUR.

[16] Note that for estimation purposes it may be advisable to use a different productivity for this part of the functional change, since disconnecting is quite different from 'real' deletes. Alternatively, for estimating purposes it may be preferable to measure the size that will be implemented (2 CFP in the example) rather than the size of the requirement (100 CFP in the example). If the 'project size' of 2 CFP is measured, this should be clearly documented and distinguished from measurement of the FUR which require that the application should be reduced in size by 100 CFP.

- new attributes are added to this data group and/or
- existing attributes are removed from the data group and/or
- one or more existing attributes are modified, e.g. in meaning or format (but not in their values)

A data manipulation is modified if it is functionally changed in any way, for instance by changing the calculation, the specific formatting, presentation, and/or validation of the data. 'Presentation' can mean, for example the font, background colour, field length, number of decimal places, etc.

Control commands and application-general data do not involve data movements, as no data about objects of interest is moved. Therefore, changes to control commands and application-general data should not be measured. As an example, when the screen colour for all screens is changed, this change should not be measured. (See section 4.1.10 for an explanation of control commands and application-general data.)

RULES – Modifying a data movement
a) If a data movement must be modified due to a change of the data manipulation associated with the data movement and/or due to a change in the number or type of the attributes in the data group moved, one changed CFP shall be measured, regardless of the actual number of modifications in the one data movement.
b) If a data group must be modified, data movements moving the modified data group whose functionality is not affected by the modification to the data group shall not be identified as changed data movements.
NOTE 1: A modification to a value of an attribute occurrence, such as a modification to an individual code member of an attribute whose values are a coding system is not a modification of the attribute's type.
NOTE 2: A modification to any data appearing on input or output screens that are not related to an object of interest to a functional user shall not be identified as a changed CFP (see section 3.3.4 for examples of such data.)

EXAMPLE for rules a) and b): Suppose a requirement to add or to modify the data attributes of a data group D_1, such that after modification it becomes D_2. In the functional process 'A' where this modification is required, all data movements affected by the modification should be identified and counted as modified. So, as per Rule a), if the changed data group D_2 is made persistent and/or is output in functional process A, identify one Write and/or one Exit data movement respectively as modified. However, it is possible that other functional processes Read or Enter this same data group D_2, but their functionality is unaffected by the modification because they do not use the changed or added data attributes. These functional processes continue to process the data group moved as if it were still D_1. So, as per Rule (b), these data movements in the other functional processes that are not affected by the modification to the data movement(s) of functional process A must NOT be identified and counted as modified.

4.4.2 Size of the functionally changed software

After functionally changing a piece of software, its new total size equals the original size, plus the functional size of all the added data movements, minus the functional size of all the removed data movements. Modified data movements have no influence on the size of the piece of software as they exist both before and after the modifications have been made.

4.5 Extending the COSMIC measurement method

4.5.1 Introduction

The COSMIC measurement method of functional size does not presume to measure all aspects of software 'size'. Thus, the COSMIC measurement method is currently not designed to provide a

standard way of accounting for the size of certain types of Functional User Requirements, notably complex mathematical algorithms or complex sequences of rules as found in expert systems. Also, the influence of the number of data attributes per data movement on software size is not captured by this measurement method. The influence on size of data manipulation sub-processes is taken into account via a simplifying assumption that is valid only for certain software domains, as defined in section 1.1.1 on the applicability of the method.

Other parameters such as 'complexity' (however defined) might be considered to contribute to functional size. A constructive debate on this matter would first require commonly agreed definitions of the other elements within the ill-defined notion of 'size' as it applies to software. Such definitions are still, at this point, the subject of further research and of much debate.

Nevertheless, the COSMIC size measure is considered to be a good approximation for the method's stated purpose and domain of applicability. Yet, it may be that within the local environment of an organization using the COSMIC measurement method, it is desired to account for such functionality in a way which is meaningful as a local standard. For this reason, the COSMIC measurement method has provision for local extensions. When such local extensions are used, the measurement results must be reported according to the special convention presented in section 5.1.

The following sections show how to extend the method with a local standard.

4.5.2 Local extension with complex algorithms

If it is judged necessary to account for complex algorithms, a local standard may be arranged for this exceptional functionality. In any functional process where there is an abnormally complex data manipulation functional sub-process, the measurer is free to assign his or her own locally-determined Function Points.

EXAMPLE: A local extension standard could be: "In our organization, one Local FP is assigned for mathematical algorithms such as (list of locally meaningful and well-understood examples). Two Local FP's are assigned for (another list of examples), etc."

4.5.3 Local extension with sub-units of measurement

When more precision is required in the measurement of data movements, then a sub-unit of the measure can be defined. For example, a meter can be sub-divided into 100 centimeters or 1000 millimeters. By analogy, the movement of a single data attribute could be used as a sub-unit of measure. Measurements on a small sample of software in the field trials of COSMIC indicated that on the sample measured, the average number of data attributes per data movement did not vary much across the four types of data movement. For this reason and for ease of measurement reasons the COSMIC unit of measurement, 1 CFP, has been fixed at the level of one data movement. However, caution is clearly needed when comparing the sizes measured in CFP of two different pieces of software where the average number of data attributes per data movement differs sharply across the two pieces of software.

Anyone wishing to refine the COSMIC method by introducing a sub-unit of measure is free to do so but must make it clear that the resulting size measures are not expressed in standard COSMIC Function Points.

5

MEASUREMENT REPORTING

The Generic Software Model can be depicted in matrix form where rows represent functional processes (which might be grouped by layers), columns represent data groups and cells hold the identified sub-processes (Entry, Exit, Read and Write). This representation of the Generic Software Model is presented in appendix A.

COSMIC measurement results are to be reported and archived according to the following conventions.

5.1 Labeling

When reporting a COSMIC functional size it should be labeled according to the following convention, in accordance with the ISO/IEC 14143-1: 2007 standard.

RULE – COSMIC measurement labeling

A COSMIC measurement result shall be noted as "***x*** CFP (v.***y***) ", where:

- "**x**" represents the numerical value of the functional size,
- "v.**y**" represents the identification of the standard version of the COSMIC method used to obtain the numerical functional size value "x".

NOTE: If a local approximation method was used to obtain the measurement, but otherwise the measurement was made using the conventions of a standard COSMIC version, the above labeling convention shall be used, but use of the approximation method should be noted elsewhere – see section 5.2.

EXAMPLE: A result obtained using the rules of this Measurement Manual is noted as '**x** CFP (v3.0.1)'

When local extensions are used, as defined in section 4.5 above, the measurement result must be reported as defined below.

RULE – COSMIC local extensions labeling

A COSMIC measurement result using local extensions shall be noted as:

" ***x*** CFP (v. ***y***) + z Local FP", where:

- "**x**" represents the numerical value obtained by aggregating all individual measurement results according to the standard COSMIC method, version v.y,
- "v.**y**" represents the identification of the standard version of the COSMIC method used to obtain the numerical functional size value "x".
- "z" represents the numerical value obtained by aggregating all individual measurement results obtained from local extensions to the COSMIC method.

5.2 Archiving COSMIC measurement results

When archiving COSMIC measurement results, the following information should be kept so as to ensure that the result is always interpretable.

RULE– COSMIC measurement reporting

In addition to the actual measurements, recorded as in 5.1, the following attributes of each measurement should be recorded.

a) Identification of the measured software component (name, version ID or configuration ID).

b) The sources of information used to identify the FUR used for the measurement

c) The domain of the software

d) A statement of the purpose of the measurement.

e) A description of the scope of the measurement, and its relation to the overall scope of a related set of measurements, if any. (Use the generic scope categories in section 2.2)

f) The functional users of the software

g) The level of granularity of the FUR and the level of decomposition of the software.

h) The point in the project life-cycle when the measurement was made (especially whether the measurement is an estimate based on incomplete FUR, or was made on the basis of actually delivered functionality).

i) The target or believed error margin of the measurement.

j) Indications whether the standard COSMIC measurement method was used, and/or a local approximation to the standard method, and/or whether local extensions were used (see section 4.5). Use the labeling conventions of sections 5.1 or 5.2.

k) An indication whether the measurement is of developed or delivered functionality ('developed' functionality is obtained by creating new software; 'delivered' functionality includes 'developed' functionality and also includes functionality obtained by other means than creating new software, i.e. including all forms of re-use of existing software, use of existing parameters to add or change functionality, etc).

l) An indication of whether the measurement is of newly provided functionality or is the result of an 'enhancement' activity (i.e. the sum is of added, changed and deleted functionality – see 4.4).

m) A description of the architecture of layers in which the measurement is made, if applicable.

n) The number of major components, if applicable, whose sizes have been added together for the total size recorded.

o) For each scope within the overall measurement scope, one measurement matrix, as specified in appendix A.

p) The measurer's name and any COSMIC certification qualifications

APPENDIX A - DOCUMENTING A COSMIC SIZE MEASUREMENT

The structure below can be used as a repository to hold the results of a measurement for each identified component of an overall scope that has been mapped to the Generic Software Model. Each scope within the overall measurement scope has its own matrix.

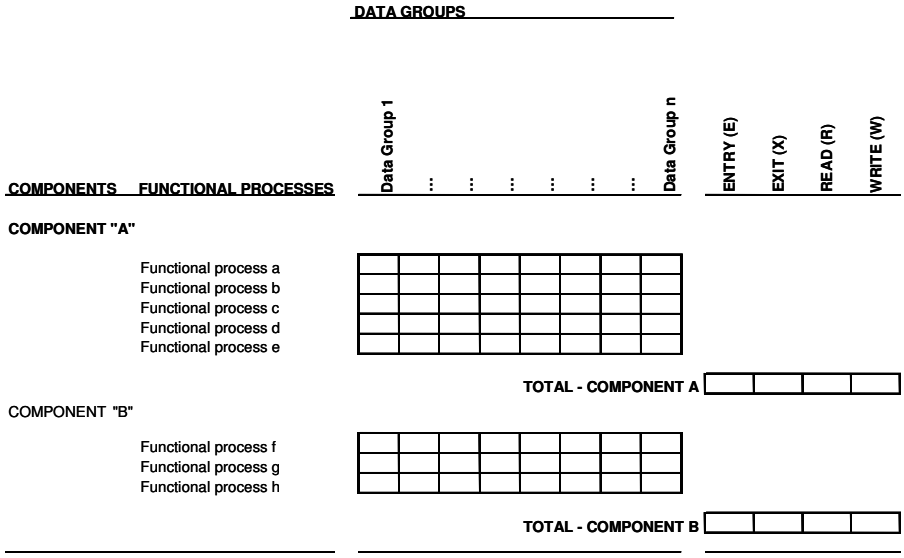


Figure A – Generic Software Model matrix

MAPPING PHASE

- Each identified data group is registered in a column
- Each functional process is registered on a specific line, grouped by identified component.

MEASUREMENT PHASE

- For each identified functional process, the identified data movements are noted in the corresponding cell using the following convention: “E” for an Entry, “X” for an Exit, “R” for a Read and “W” for a Write
- For each identified functional process, the data movements are then summed up by type and each total is registered in the appropriate column at the far right of the matrix
- The measurement summary can then be calculated and registered in the boxed cells of each component, on the “TOTAL” line.

APPENDIX B - SUMMARY OF COSMIC METHOD PRINCIPLES

The table below identifies each principle found in the COSMIC Measurement Method for the purpose of precise referencing.

ID	PRINCIPLE DESCRIPTION
P-01	**The COSMIC Software Context Model** a) Software is bounded by hardware b) Software is typically structured into **layers** c) A layer may contain one or more separate '**peer**' pieces of software and any one piece of software may further consist of separate peer components d) Any piece of software to be measured, shall be defined by its measurement **scope**, which shall be confined wholly within a single layer e) The scope of a piece of software to be measured shall depend on the **purpose** of the measurement f) The **functional users** of a piece of software shall be identified from the functional user requirements of the piece of software to be measured as the senders and/or intended recipients of data g) A piece of software interacts with its functional users via **data movements** across a **boundary** and the piece of software may move data to and from **persistent storage** within the boundary h) The FUR of software may be expressed at different **levels of granularity** i) The level of granularity at which measurements should normally be made is that of the **functional processes** j) If it is not possible to measure at the level of granularity of the functional processes, then the FUR of the software should be measured by an approximation approach and scaled to the level of granularity of the functional processes
P-02	**The COSMIC Generic Software Model** a) Software receives **input** data from its functional users and produces **output**, and/or another outcome, for the functional users b) Functional user requirements of a piece of software to be measured can be mapped into unique functional processes c) Each functional process consists of **sub-processes** d) A sub-process may be either a data movement or a **data manipulation** e) Each functional process is triggered by an **Entry** data movement from a functional user which informs the functional process that the functional user has identified an **event** f) A data movement moves a single **data group** g) A data group consists of a unique set of **data attributes** that describe a single object of interest h) There are four types of data movement. An Entry moves a data group into the software from a functional user. An **Exit** moves a data group out of the software to a functional user. A **Write** moves a data group from the software to persistent storage. A **Read** moves a data group from persistent storage to the software. i) A functional process shall include at least one Entry data movement and either a Write or an Exit data movement, that is it shall include a minimum of two data movements j) As an approximation for measurement purposes, data manipulation sub-processes are not separately measured; the functionality of any data manipulation is assumed to be accounted for by the data movement with which it is associated.

P-03	**The COSMIC measurement principle** The functional size of a piece of software is directly proportional to the number of its data movements.
P-04	**Layer** a) Software in one layer exchanges data with software in another layer via their respective functional processes. b) The 'hierarchical dependency' between layers is such that software in any layer may use the functional services of any software in any layer beneath it in the hierarchy. Where there are such usage relationships, we designate the using software layer as the 'superior' and any layer containing the used software as its 'subordinate'. The software in the superior layer relies on the services of software in these subordinate layers to perform properly; the latter rely in turn on software in their subordinate layers to perform properly, and so on, down the hierarchy. Conversely, software in a subordinate layer, together with software in any subordinate layers on which it depends, can perform without needing the services of software in any superior layer in the hierarchy. c) Software in one layer does not necessarily use all the functional services supplied by software in a subordinate layer. d) The data that is exchanged between software in any two layers is defined and interpreted differently in the respective FUR of the two pieces of software, that is, the two pieces of software recognise different data attributes and/or sub-groupings of the data that they exchange. However, there must also exist one or more commonly defined data attributes or sub-groups to enable the software in the receiving layer to interpret data that has been passed by the software in the sending layer, according to the receiving software's needs.
P-05	**Peer component** a) In a set of peer components of a piece of software in one layer there is no hierarchical dependency between the peer components as there is between layers. The FUR of all peer components of a piece of software in any one layer are at the same 'level' in the hierarchy of layers. b) All peer components of a piece of software must co-operate mutually in order that the piece of software can perform successfully. c) A data group may be exchanged directly between two peer components of a piece of software by a functional process of a first component issuing an Exit which is received as an Entry by a functional process of the second component. Alternatively, the exchange may take place indirectly by a functional process of a first component making a data group persistent via a Write that can be subsequently retrieved via a Read of a functional process of the second component
P-06	**Applying the COSMIC Generic Software Model** The COSMIC Generic Software Model shall be applied to the functional user requirements of each separate piece of software for which a separate measurement scope has been defined. 'Applying the COSMIC Generic Software Model' means identifying the set of triggering events sensed by each of the functional user (types) identified in the FUR, and then identifying the corresponding functional processes, objects-of-interest, data groups, and data movements that must be provided to respond to those events.
P-07	**Data group** a) Each identified data group shall be unique and distinguishable through its unique collection of data attributes. b) Each data group shall be directly related to one object of interest in the software's Functional User Requirements c) A data group shall be materialized within the computer system supporting the software.

P-08	**Entry (E)** a) An Entry shall move a single data group describing a single object of interest from a functional user across the boundary and into the functional process of which the Entry forms part. If the input to a functional process comprises more than one data group, identify one Entry for each unique data group in the input. (See also section 4.1.7 on 'Data movement uniqueness'.) b) An Entry shall not exit data across the boundary, or read or write data. c) Where a functional process needs to obtain data from a functional user but the latter does not need to be told what data to send, or the functional user is incapable of reacting to any incoming message, identify one Entry to the functional process for obtaining the data. Any message from the functional process to the functional user seeking to retrieve the data shall not be counted as an Exit in these cases. However, where a functional process must obtain some data from a functional user and the functional process must provide the functional user with data which the latter requires to fulfill the request, count one Exit for the request and one Entry for the return of the requested data (see further section 4.1.9).
P-09	**Exit (X)** a) An Exit shall move a single data group describing a single object of interest from the functional process of which the Exit forms part across the boundary to a functional user. If the output of a functional process comprises more than one data group, identify one Exit for each unique data group in the output. (See also section 4.1.7 on 'Data movement uniqueness'.) b) An Exit shall not enter data across the boundary, or read or write data
P-10	**Read (R)** a) A Read shall move a single data group describing a single object of interest from persistent storage to a functional process of which the Read forms part. If the functional process must retrieve more than one data group from persistent storage, identify one Read for each unique data group that is retrieved. (See also section 4.1.7 on 'Data movement uniqueness'.) b) A Read shall not receive or exit data across the boundary or write data. c) During a functional process, movement or manipulation of constants or variables which are internal to the functional process and that can be changed only by a programmer, or computation of intermediate results in a calculation, or of data stored by a functional process resulting only from the implementation, rather than from the FUR, shall not be considered as Read data movements. d) A Read data movement always includes any 'request to Read' functionality (so a separate data movement shall never be counted for any 'request to Read' functionality). See also section 4.1.9.
P-11	**Write (W)** a) A Write shall move a single data group describing a single object of interest from the functional process of which the Write forms part to persistent storage. If the functional process must move more than one data group to persistent storage, identify one Write for each unique data group that is moved to persistent storage. (See also section 4.1.7 on 'Data movement uniqueness'.) b) A Write shall not receive or exit data across the boundary, or read data. c) A requirement to delete a data group from persistent storage shall be measured as a single Write data movement. d) During a functional process, movement or manipulation of data that does not persist when the functional process is complete, or updating variables which are internal to the functional process or producing intermediate results in a calculation shall not be considered as Write data movements.

P-12	**Data manipulation associated with data movements** All data manipulation in a functional process shall be associated with the four types of data movement (E, X, R, and W). By convention, the data movements of a functional process are assumed also to represent the data manipulation of the functional process

APPENDIX C - SUMMARY OF COSMIC METHOD RULES

The table below identifies each rule found in the COSMIC measurement method for the purpose of precise referencing.

ID	RULE DESCRIPTION
R-01	**Scope** a) The scope of a Functional Size Measurement (FSM) shall be derived from the purpose of the measurement. b) The scope of any one measurement shall not extend over more than one layer of the software to be measured
R-02	**Layer** a) If software is conceived using an established architecture of layers according to the COSMIC model, then that architecture should be used to identify the layers for measurement purposes b) In the domain of MIS or business software, the 'top' layer, i.e. the layer that is not a subordinate to any other layer, is normally referred to as the 'application' layer. (Application) software in this layer ultimately relies on the services of software in all the other layers for it to perform properly. In the domain of real-time software, software in the 'top layer' is commonly referred to as a 'system', for example as in 'process control system software', 'flight control system software'.. c) Do not assume that any software that has evolved without any consideration of architectural design or structuring can be partitioned into layers according to the COSMIC model.
R-03	**Functional users** a) The functional users of a piece of software to be measured shall be derived from the purpose of the measurement b) When the purpose of a measurement of a piece of software is related to the effort to develop or modify the piece of software, then the functional users should be those for whom the new or modified functionality must be provided.
R-04	**Boundary** a) Identify the functional user(s) that interact with the software being measured. The boundary lies between the functional users and this software. b) By definition, there is a boundary between each identified pair of layers where the software in one layer is the functional user of software in another, and the latter is to be measured. c) There is a boundary between any two pieces of software, including any two components that are peers of each other; in this case each piece of software and/or each component can be a functional user of its peer.
R-05	**Functional process level of granularity** a) Functional size measurement should be made at the functional process level of granularity b) Where a functional size measurement is needed of some FUR that have not yet evolved to the level where all the functional processes have been identified and all the details of their data movements have been defined, measurements should be made of the functionality that has been defined, and then scaled to the level of granularity of functional processes. (See the 'Advanced and Related Topics'

ID	RULE DESCRIPTION
	document for methods of 'approximate sizing, i.e. of estimating a functional size early in the process of establishing FUR.)
R-06	**Functional process** a) A functional process shall be derived from at least one identifiable Functional User Requirement within the agreed scope. b) A functional process (-type) shall be performed when an identifiable triggering event (-type) occurs c) A specific event (-type) may trigger one or more functional process (-types) that execute in parallel. A specific functional process (-type) may be triggered by more than one event (-type) d) A functional process shall comprise at least two data movements, an Entry plus either an Exit or a Write. e) A functional process shall belong entirely to the measurement scope of one piece of software in one, and only one, layer. f) In the context of real-time software a functional process shall be considered terminated when it enters a self-induced wait state (i.e. the functional process has done all that is required to be done in response to the triggering event and waits until it receives the next triggering Entry). g) One functional process (-type) shall be identified even if its FUR allow that the one functional process can occur with different sub-sets of its maximum number of input data attributes, and even though such variations and/or differing input data values may give rise to different processing paths through the functional process. h) Separate event (-types) and therefore separate functional process (-types) should be distinguished in the following cases: • When decisions result in separate events that are disengaged in time (e.g. entering order data today and later confirming acceptance of the order, requiring a separate decision, should be considered as indicating separate functional processes), • When the responsibility for activities is separate (e.g. in a personnel system where the responsibility for maintaining basic personal data is separated from the responsibility for maintaining payroll data, indicating separate functional processes; or for an implemented software package where there is functionality available to a system administrator to maintain the package parameters, which is separate from the functionality available to the 'regular' functional user.)
R-07	**Entry (E)** a) The data group of a triggering Entry may consist of only one data attribute which simply informs the software that 'an event Y has occurred'. Very often, especially in business application software, the data group of the triggering Entry has several data attributes which inform the software that 'an event Y has occurred and here is the data about that particular event'. b) Clock-ticks that are triggering events shall always be external to the software being measured. Therefore, for example, a clock-tick event occurring every 3 seconds shall be associated with an Entry moving a data group of one data attribute. Note that it makes no difference whether the triggering event is generated periodically by hardware or by another piece of software outside of the boundary of the software being measured. c) Unless a specific functional process is necessary, obtaining the time from the system's clock shall not be considered to cause an Entry. d) If an occurrence of a specific event triggers the Entry of a data group comprising

ID	RULE DESCRIPTION
	up to 'n' data attributes of a particular object of interest and the FUR allows that other occurrences of the same event can trigger an Entry of a data group which has values for attributes of only a sub-set of the 'n' attributes of the object of interest, then one Entry shall be identified, comprising all 'n' data attributes.
R-08	**Exit (X)** a) All messages generated and output by software without user data (e.g. error messages) shall be considered to be values of one attribute of one object of interest (which could be named 'error indication'). Therefore, a single Exit shall be identified to represent all these message occurrences within each functional process where they are required by the FUR. b) If an Exit of a functional process moves a data group comprising up to 'n' data attributes of a particular object of interest and the FUR allows that the functional process may have an occurrence of an Exit that moves a data group which has values for attributes of only a sub-set of the 'n' attributes of the object of interest, then one Exit shall be identified, comprising all 'n' data attributes.
R-09	**Data movement uniqueness and possible exceptions** a) Unless the Functional User Requirements specify otherwise, all data attributes describing any one object of interest that is required to be entered into one functional process, and all associated data manipulation shall be identified and counted as one Entry (type). (Note: A functional process may, of course, be required to process multiple Entry types, each moving a data group describing a different object of interest (type).) The same equivalent rule applies to any Read, Write or Exit data movement in any given functional process b) More than one Entry data movement (type), each moving a data group describing the same object of interest (type) in a given functional process (type) may be identified and counted if there is a Functional User Requirement for these multiple Entries. Similarly, more than one Entry (type) moving the same data group (type) in the same functional process, but each with different associated data manipulation (types), may be identified and counted if there is a Functional User Requirement for these multiple Entries. Such FUR may arise when, in one functional process, the multiple Entries originate from different functional users who enter different data groups (each describing the same object of interest), The same equivalent rule applies to any Read, Write or Exit data movement in any given functional process c) Repeated *occurrences* of a data movement type (i.e. moving the same data group with the same data manipulation) shall not be identified and counted more than once in any one functional process d) Even if the multiple *occurrences* of a data movement type in a given functional process differ in their associated data manipulation because different *values* of the data attributes of the data group moved result in different processing paths being followed, the data movement type shall not be identified and counted more than once in that process.
R-10	**When a functional process requires data from a functional user** a) A functional process shall obtain a data group via an Entry data movement from a functional user, *when it does not need to tell the functional user what data to send*, as in any of the following four cases: • when a functional user sends a triggering Entry which initiates the functional process; • when a functional process, having received a triggering Entry, waits,

ID	RULE DESCRIPTION
	expecting the arrival of a further Entry from the functional user (may occur when a human functional user enters data to business application software); • when a functional process, having started, requests the function'l user, 'send me your data now, if you'have any' and the functional user sends its data; • when a functional process, having started, inspects the state of a functional user and retrieves the data it requires. In the latter two cases (typically occurring in real-time 'polling' software), by convention no Exit from the functional process shall be identified to obtain the required data. The functional process merely needs to send a prompt message to a functional user and the functionality of that prompt message is considered to be part of the Entry. The functional process knows what data to expect. Only one Entry is needed for this case. b) Where a functional process needs to obtain the services of a functional user (for instance to obtain data) and *the functional user needs to be told what to send* (typically where the functional user is another piece of software outside the scope of the software being measured), a pair of Exit/Entry data movements shall be identified. The Exit contains the request for the specific data; the Entry contains the returned data.''
R-11	**Control commands in the business application domain** In the business application domain 'control commands' shall be ignored as they do not involve any movement of data about an object of interest.
R-12	**Aggregation of measurement results** a) For any functional process, the functional sizes of individual data movements shall be aggregated into a single functional size value in units of CFP by arithmetically adding them together. Size (functional process$_i$) = Σ size(Entries$_i$) + Σ size(Exits$_i$) + Σ size(Reads$_i$) + Σ size(Writes$_i$) b) For any functional process, the functional size of changes to its Functional User Requirements shall be aggregated from the sizes of the data movements that have been added, modified or deleted in the functional process to give a size of the change in units of CFP, according to the following formula. Size (Change(functional process$_i$)) = Σ size(added data movements$_i$) + Σ size(modified data movements$_i$) + Σ size(deleted data movements$_i$) For more on aggregating functional size, see section 4.3.2. For measuring the size of the changed software, see section 4.4. c) The size of a piece of software within a defined scope shall be obtained by aggregating the sizes of the functional processes for the piece, subject to rules e) and f) below d) The size of any change to a piece of software within a defined scope shall be obtained by aggregating the sizes of all changes to all functional processes for the piece, subject to rules e) and f) below e) Sizes of pieces of software or of changes to pieces of software within layers may be added together only if measured at the same functional process level of granularity of their FUR. f) Furthermore, sizes of pieces of software and/or changes in the sizes of piece of software within any one layer or from different layers shall be added together only if it makes sense to do so, for the purpose of the measurement. g) The size of a piece of software cannot be obtained by adding up the sizes of its components (regardless of how the item is sub-divided) *unless* the size contributions of inter-component data movements are eliminated. h) If the COSMIC method is extended locally (for example to measure some aspect

ID	RULE DESCRIPTION
	of size not covered by the standard method), then the size measured via the local extension must be reported separately as described in section 5.1 and may NOT be added to the size obtained by the standard method, measured in CFP (see further in section 4.5)
R-13	**Modifying a data movement** a) If a data movement must be modified due to a change of the data manipulation associated with the data movement and/or due to a change in the number or type of the attributes in the data group moved, one changed CFP shall be measured, regardless of the actual number of modifications in the one data movement. b) If a data group must be modified, data movements moving the modified data group whose functionality is not affected by the modification to the data group shall not be identified as changed data movements. NOTE 1: A modification to a value of an attribute occurrence, such as a modification to an individual code member of an attribute whose values are a coding system is not a modification of the attribute's type. NOTE 2: A modification to any data appearing on input or output screens that are not related to an object of interest to a functional user shall not be identified as a changed CFP (see section 3.3.4 for examples of such data.)
R-14	**COSMIC measurement labeling** A COSMIC measurement result shall be noted as "***x*** CFP (v.***y***) ", where: • "**x**" represents the numerical value of the functional size, • "v.**y**" represents the identification of the standard version of the COSMIC method used to obtain the numerical functional size value "x". NOTE: If a local approximation method was used to obtain the measurement, but otherwise the measurement was made using the conventions of a standard COSMIC version, the above labeling convention shall be used, but use of the approximation method should be noted elsewhere – see section 5.2.
R-15	**COSMIC local extensions labeling** A COSMIC measurement result using local extensions shall be noted as: " ***x*** CFP (v. ***y***) + z Local FP", where: • "**x**" represents the numerical value obtained by aggregating all individual measurement results according to the standard COSMIC method, version v.y, • "v.**y**" represents the identification of the standard version of the COSMIC method used to obtain the numerical functional size value "x". • "z" represents the numerical value obtained by aggregating all individual measurement results obtained from local extensions to the COSMIC method.
R-16	**COSMIC measurement reporting** In addition to the actual measurements, the following attributes of each measurement should be recorded. a) Identification of the measured software component (name, version ID or configuration ID). b) The sources of information used to identify the FUR used for the measurement c) The domain of the software d) A statement of the purpose of the measurement. e) A description of the scope of the measurement, and its relation to the overall scope of a related set of measurements, if any. (Use the generic scope categories in section 2.2) f) The functional users of the software g) The level of granularity of the FUR and the level of decomposition of the software. h) The point in the project life-cycle when the measurement was made (especially whether the measurement is an estimate based on incomplete FUR, or was made on the basis of actually delivered functionality). i) The target or believed error margin of the measurement.

ID	RULE DESCRIPTION
	j) Indications whether the standard COSMIC measurement method was used, and/or a local approximation to the standard method, and/or whether local extensions were used (see section 4.5). Use the labeling conventions of sections 5.1 or 5.2. k) An indication whether the measurement is of developed or delivered functionality ('developed' functionality is obtained by creating new software; 'delivered' functionality includes 'developed' functionality and also includes functionality obtained by other means than creating new software, i.e. including all forms of re-use of existing software, use of existing parameters to add or change functionality, etc). l) An indication of whether the measurement is of newly provided functionality or is the result of an 'enhancement' activity (i.e. the sum is of added, changed and deleted functionality – see 4.4)). m) A description of the architecture of layers in which the measurement is made, if applicable. n) The number of major components, if applicable, whose sizes have been added together for the total size recorded. o) For each scope within the overall measurement scope, one measurement matrix, as specified in appendix A. p) The measurer's name and any COSMIC certification qualifications

APPENDIX D - HISTORY OF COSMIC METHOD RELEASES

This Appendix contains a summary of the principal changes made in deriving version 3.0 and this version 3.0.1 of the COSMIC functional size measurement method from version 2.2. Version 2.2 of the method was fully described in the 'Measurement Manual v2.2' (abbreviated to 'MM'), but from v3.0 onwards, the method documentation is now distributed over four documents, only one of which is named the 'Measurement Manual'.

The purpose of the Appendix is to enable a reader who is familiar with the MM v2.2 to trace the changes that have been made for versions 3.0 and 3.0.1 and where necessary to understand their justification. (For the changes made in deriving version 2.2 from version 2.1, please see version 2.2 of the Measurement Manual, Appendix E.)

From version 2.2 to version 3.0

In the table below showing the principal changes made in deriving v3.0 from v2.2, reference is made to two 'Method Update Bulletins' (or 'MUB's'). A MUB is published by COSMIC to propose improvements to the method between major releases of the method definition. The two MUB's are:

- MUB 1 "Proposed Improvements to the definition and characteristics of a software 'Layer'", published in May 2003.
- MUB 2 "Proposed Improvement to the definition of an 'object of interest'", published in March 2005.

In the process of updating the method from v2.2 to v3.0, an effort has been made to rationalize guidance as between 'principles' and 'rules' and to separate all examples from the principles and rules. Changes such as these and many editorial improvements are not described in the following.

V2.2 Ref	V3.0 Ref	Change
COSMIC Method re-structuring		
2.2, 2.7, 3.1, 3.2	1.5, Chapter 2	A 'Measurement Strategy' phase has been separated out as the first phase of what is now a three-phase method. The Measurement Strategy phase now includes consideration of 'layers', 'boundaries' and '(functional) users', which were considered to be part of the Mapping phase in the MM v2.2
Measurement Manual re-structuring		
		In producing v3.0 of the COSMIC Method, the MM V2.2 has been split into four documents for ease of use • 'COSMIC Method v3.0: Documentation Overview and Glossary of Terms'' (new) • 'COSMIC Method v3.0: Method Overview' (Chapters 1 and 2 from MM v2.2) • 'COSMIC Method v3.0: Measurement Manual' (Chapters 3, 4 and 5 and the Appendices from MM v2.2 • 'COSMIC Method v3.0: Advanced and Related Topics' (Chapters 6 and 7 from MM v2.2)

		References to background and research papers have been removed from the MM. They are now available from www.gelog.etsmtl.ca/cosmic-ffp. The only references remaining in these four documents are given in footnotes
Chapter 4	Chapter 4	The chapter on the Measurement Phase has been considerably re-structured to provide a more logical exposition
Appendix D	--	This Appendix on 'Further information on software layers' has been removed, as being incompatible with MUB 1 and adding little value. See also the notes below on changes to take account of MUB 1
Changes of names and terminology		
General		The 'COSMIC-FFP' name of the method has been simplified to the 'COSMIC' method
5.1	4.2	The name of the unit of measure, 'COSMIC functional size unit' (abbreviated as 'Cfsu') has been changed to 'COSMIC Function Point' (abbreviated as 'CFP'). See the Foreword of this MM v3.0 for an explanation for this change
4.1	4.1.7	The 'Data movement de-duplication' rule has been re-named the 'Data movement uniqueness' rule
New, replaced and removed concepts		
2.7	2.3	The concepts of 'abstraction', 'viewpoint', 'Measurement Viewpoint', the 'End User Measurement Viewpoint' and the 'Developer Measurement Viewpoint' have been removed. They are replaced by the more general concept that the functional size of a piece of software to be measured depends on the functionality that is made available to the 'functional user(s)' of the software. These should be identifiable in the Functional User Requirements for the software to be measured. Defining the 'functional user(s)' is therefore a pre-requisite for defining which size must be measured or for interpreting an existing size measurement. See the Foreword of this MM v3.0 for a more detailed explanation for this change
3.2	2.3	The concept of 'user' (as defined in ISO/IEC 14143/1) has been replaced by the concept of 'functional user' which is a more restricted concept. See the Foreword of this MM v3.0 for a more detailed explanation for this change. In 2.3.2, two examples have been introduced of where the functional size varies with the type of functional user identified in the FUR, for a mobile phone and for a business application software package
--	2.2.2	The 'level of decomposition' of the software to be measured has been introduced into discussion of the measurement scope
--	2.4	The 'level of granularity' of the functional user requirements of the software to be measured has been introduced into discussion of the measurement strategy. A comprehensive example is given. See the Foreword of this MM v3.0 for a more detailed explanation for this change
--	2.4.3	A 'functional process level of granularity' has been defined and rules and a recommendation are given
3.4	4.2	The concept of 'data group persistence' including three levels of persistence, namely 'transient', 'short' and 'indefinite' has been removed as it proved unnecessary. The concept of 'persistent storage' has been introduced
4.1	--	The concept of 'de-duplication' has been removed as it proved to be unnecessary
Improved and refined definitions, principles and rules		
2.3	2.2	The definition of 'Functional User Requirements' has been changed to conform to the 2007 edition of ISO/IEC 14143/1
2.4.1	1.3	The 'Software Context Model' has been extended and refined as a statement of principles

2.4.2	1.4	The 'Generic Software Model' has been extended and refined as a statement of principles
2.4, 3.1	2.2.3, 2.2.4, 3.1	The definition of 'layer' has been updated to take account of MUB 1. Figures 2.4.1.1, 2.4.1.2 and 2.4.1.3 of the MM v2.2 which were used to illustrate the interaction of users with software in 'layers' have been replaced in v3.0 by Figures 2.2.3.1 and 2.2.3.2 illustrating typical physical layered architectures and Figures 3.1.1 and 3.1.2 illustrating the logical interaction of functional users with software in layers. The aim of this change is to more clearly distinguish the *physical* view of typical layered software architectures from the *logical* view of a functional user interacting with a piece of software to be measured according to the COSMIC model
--	3.2.4	The concept of a 'peer component' has been defined and its principles established taking account MUB 1
2.5	4.2, 4.3	The 'Characteristics' of the measurement process described in v2.2 have been re-written as a set of principles and rules in v3.0
2.6	2.4	Material on 'sizing early in a project life: measurement scaling' has been dealt with partly in the MM v3.0 section 2.4 under 'Level of Granularity' and is elaborated in the document 'COSMIC Method v3.0: Advanced and Related Topics'
2.7	2.2	A note has been added to the definition of 'scope' to distinguish the 'overall scope' of a measurement exercise (which may include several separate pieces of software whose size must be measured) from the 'scope' of an individual size measurement.
3.2	4.1.8	Three Examples that illustrate the interaction of users across a boundary with software in different layers and from different 'measurement viewpoints' have been moved and re-written in light of the removal of 'measurement viewpoints'. See 4.1.8 (of the MM V3.0) below
--	3.1	A principle has been added concerning the application of the Generic Software Model to the software to be measured
3.3	3.2.1	The definition of 'functional process' has been refined to take into account the introduction of the concept of a 'functional user' which replaces 'actor' in this definition
3.3	3.2.1	The definition of 'triggering event' has been refined
--	3.2.1	The relationship between a triggering event, a functional user, a triggering Entry and a functional process has been clarified in Figure 3.1.1
3.1	3.2.2	The principles and rules for a 'functional process' have been merged into a revised set of rules
--	3.2.3, 3.2.4, 3.2.5	Many examples of functional processes and of how to distinguish them have been introduced
--	3.3.1	Definition of an 'object of interest' has been introduced, in line with MUB 2
3.4	3.3.3	Examples of the identification of objects of interest and data groups have been separated from the rules for data groups. Some domain-specific material on entity-relationship data analysis conventions has been removed to the 'Guideline for sizing Business Application Software v1.0'.
--	3.3.4	Guidance is given on 'data or groups of data that are not candidates for data movements'
--	3.3.5	Guidance is given on when, typically in real-time software, it may not be worth distinguishing a 'functional user' form an 'object of interest' about which data is moved
3.5	3.4	The discussion of 'data attributes' has been reduced and simplified since consideration of data attributes is not a mandatory part of the method
4.1	4.1.1	The definition of a 'data movement' has been rationalized
4.1	4.1.7	Rules on 'data movement de-duplication' (now re-named as rules on 'data movement uniqueness and possible exceptions') have been much clarified with several examples added
4.1	--	Domain-specific rules on Read and Write data movements in 'Update'

		Functional Processes have been removed to the 'Guideline for sizing Business Application Software v1.0'.
4.1	4.1.8	'Rules for correspondence of data across boundaries' (which in the MM v2.2 applied in the 'Developer Measurement Viewpoint', which has now been eliminated) have been deleted but the concepts have been combined with the Cases given previously in the MM v2.2 section, 3.2, to produce a new section on 'when a functional process moves data to or from persistent storage'
4.1	4.1.6	'Data manipulation' has been defined and a principle added on 'data manipulation associated with data movements'. Guidelines on the data manipulation associated with the different types of data movements have been expanded
4.1.1	4.1.2	Principles and rules for an Entry have been rationalized. A new principle has been added concerning 'request to enter' functionality
4.1.2	4.1.3	Principles and rules for an Exit have been rationalized including removal of reference to the 'end-user measurement viewpoint'
4.1.3	4.1.4	Principles for a Read have been rationalized. A new principle has been added concerning 'request to read' functionality
4.1.4	4.1.5	Principles for a Write have been rationalized
--	4.1.9	New rules have been added on 'when a functional process requires data from a functional user'
--	4.1.10	A definition and rule have been introduced for the concept of a 'control command' which is valid only in the business application software domain
4.1.5	4.5	The discussion of 'local extensions to the method' has been expanded
4.3	4.3	Principles of 'aggregation of measurement results' have been changed to 'rules' and expanded to cover the rules for obtaining the size of a piece of software by adding up the sizes of its components. The MM v2.2 referred to such rules only in the context of the 'developer measurement viewpoint'. This restriction has been removed
--	4.4	A new section on' measurement of the size of changes to software' and new rules have been added
5.1	5.1	Rules for labeling measurement results have been changed to recognize the change of the unit of measure from 'Cfsu' to 'CFP'
5.2	5.2	Rules on 'measurement reporting' have been expanded to list more items
App. B	App. B	Updated to v3.0 Principles
App. C	App. C	Updated to v3.0 Rules

From version 3.0 to version 3.0.1

The most important changes in deriving version 3.0.1 from version 3.0 are where it has been found desirable to improve the wording of a few definitions, principles and rules, and some parts of the text. With the exception of one error, the improvements have all been made for reasons of clarity. Most of the changes had been published in three Method Update Bulletins prior to version 3.0.1.

- MUB 3: 'Correction of an error in Fig. 4.1.8.1 (b) of the COSMIC Method v3.0 Measurement Manual', published in June 2008
- MUB 4: 'Clarification of the principle and rules for 'Request to enter' functionality in the COSMIC Method v3.0 Measurement Manual', published in June 2008
- MUB 5: 'Proposed improvements to (a) Definitions of 'Level of Decomposition' and of a 'Peer Component', and (b) Principle c) for a 'Peer Component', published in February 2009

Additionally, several editorial improvements have been made. Principally these involve separating the Examples more clearly from the main text, by using a different type-font and introducing more sub-sections.

Summary of main changes:

V3.0.1 Ref	Change
2.2.3	Definition of 'Level of Decomposition' changed for clarity (MUB 5).
2.2.4	In the definition of 'Layer', the term 'software architecture' has been replaced by 'software system', since the term 'architecture' implies that the software is already partitioned. Rule c) removed. This was a general rule of software design and not specific to layers and measurement. The reference to 'Appendix D' in v3.0 was incorrect.
2.2.5	Definition of 'Peer' inserted and definition of 'Peer Component' changed for clarity. Principle c) for 'Peer component' changed to a principle that is more relevant to FSM. Figure 2.2.5.1 added to clarify the relationship between peer components and peer pieces of software. (MUB 5).
2.3.2	Rule c) for Boundary changed for clarity. (Consequence of MUB 5).
2.4.3	In Figure 2.4.3.1, in the bottom two boxes in the right-hand corner, 'payment method' is changed to 'means-of-payment', as this is a better term for e.g. cheque, credit card, etc.
3.2.2	Rule e) for a functional process has been constrained by adding the words in italics. It is now: 'A functional process shall belong entirely to *the measurement scope of one piece of* software in one, and only one, layer'. This constraint existed already in the Guideline for Sizing Business Application Software' v1.1, but is valid for software from any domain.
3.2.6	New section added entitled 'The functional processes of peer components'. This text is taken largely from the 'Guideline for Sizing Business Application Software' v1.1, but is valid for software from any domain. This is related to the change to Rule e) in 3.2.2.
4.1.2	Principle c) for an Entry changed to clarify how to account for 'Request to enter' functionality (MUB 4).
4.1.7	The opening sentence of this section on 'Data Uniqueness and Possible Exceptions' has been changed to clarify the intended meaning and to make it consistent with Rule a). Example 2 has also been extensively modified for clarity.
4.1.8	Figure 4.1.8.1 (b) changed to correct an error on how a device driver software interacts with hardware (MUB 3).
4.1.9	Rules for 'When a functional process requires data from a functional user', and the related examples changed for clarity (MUB 4).

APPENDIX E - COSMIC CHANGE REQUEST AND COMMENT PROCEDURE

The COSMIC Measurement Practices Committee (MPC) is very eager to receive feedback, comments and, if needed, Change Requests for the COSMIC Measurement Manual. This Appendix sets out how to communicate with the COSMIC MPC.

All communications to the COSMIC MPC should be sent by e-mail to the following address:

mpc-chair@cosmicon.com

Informal General Feedback and Comments

Informal comments and/or feedback concerning the Measurement Manual, such as any difficulties of understanding or applying the COSMIC method, suggestions for general improvement, etc should be sent by e-mail to the above address.

Messages will be logged and will generally be acknowledged within two weeks of receipt. The MPC cannot guarantee to action such general comments.

Formal Change Requests

Where the reader of the Measurement Manual believes there is an error in the text, a need for clarification, or that some text needs enhancing, a formal Change Request ('CR') may be submitted.

Formal CR's will be logged and acknowledged within two weeks of receipt. Each CR will then be allocated a serial number and it will be circulated to members of the COSMIC MPC, a world wide group of experts in the COSMIC method. Their normal review cycle takes a minimum of one month and may take longer if the CR proves difficult to resolve.

The outcome of the review may be that the CR will be accepted, or rejected, or 'held pending further discussion' (in the latter case, for example if there is a dependency on another CR), and the outcome will be communicated back to the Submitter as soon as practicable.

A formal CR will be accepted only if it is documented with all the following information.

- Name, position and organisation of the person submitting the CR
- Contact details for the person submitting the CR
- Date of submission
- General statement of the purpose of the CR (e.g. 'need to improve text…')
- Actual text that needs changing, replacing or deleting (or clear reference thereto)
- Proposed additional or replacement text
- Full explanation of why the change is necessary

A form for submitting a CR is available from the www.cosmicon.com site.

The decision of the COSMIC MPC on the outcome of a CR review and, if accepted, on which version of the Measurement Manual the CR will be applied to, is final.

Questions on the application of the COSMIC method

The COSMIC MPC regrets that it is unable to answer questions related to the use or application of the COSMIC method. Commercial organisations exist that can provide training and consultancy or tool support for the method. Please consult the www.cosmicon.com web-site for further details