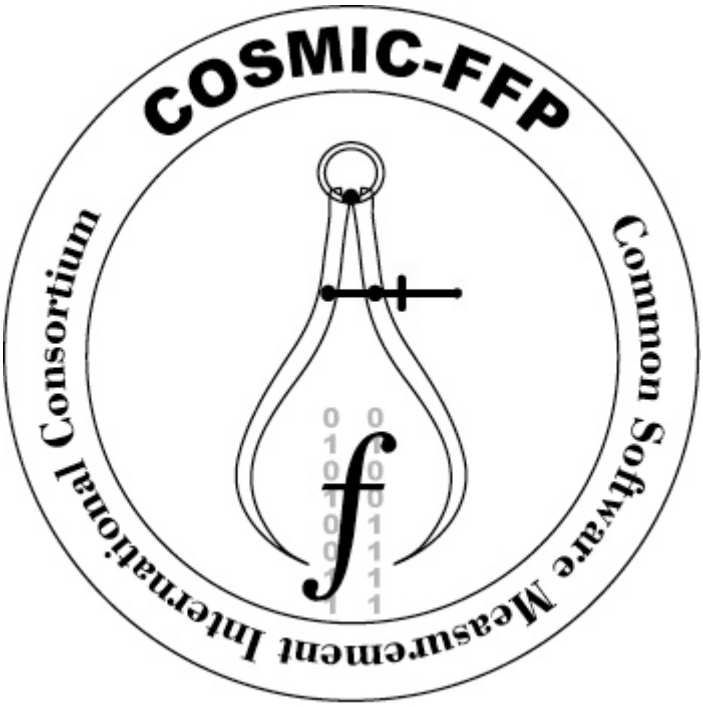


MEASUREMENT MANUAL

(THE COSMIC IMPLEMENTATION GUIDE FOR ISO/IEC 19761: 2003)

VERSION 2.2

January 2003

Acknowledgements

COSMIC CORE TEAM AUTHORS, VERSION 2.0 (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 2.2 REVIEWERS (2002/2003)

- ALAIN ABRAN, ÉCOLE DE TECHNOLOGIE SUPÉRIEURE, UNIVERSITÉ DU QUÉBEC, CANADA
- MORITSUGU ARAKI, SOFT METRICS JIN INC., JAPAN
- JEAN-MARC DESHARNAIS, SOFTWARE ENGINEERING LAB IN APPLIED METRICS – SELAM, CANADA
- PETER FAGG, SOFTWARE MEASUREMENT SERVICES – SMS, UNITED KINGDOM*
- PEKKA FORSELIUS, SOFTWARE TECHNOLOGY TRANSFER FINLAND OY, FINLAND
- VINH T. HO, INSTITUT FRANCOPHONE D'INFORMATIQUE – IFI, VIETNAM
- ROBERTO MELI, DATA PROCESSING ORGANIZATION SRL, ITALY
- ARLAN LESTERHUIS, SOGETI, THE NETHERLANDS
- BERNARD LONDEIX, TELMACO, UNITED KINGDOM
- PAM MORRIS, TOTAL METRICS, AUSTRALIA *
- MARIE O'NEILL, SOFTWARE MANAGEMENT METHODS, IRELAND
- SERGE OLIGNY, BELL CANADA
- TONY ROLLO, SOFTWARE MEASUREMENT SERVICES LTD., UNITED KINGDOM
- DENIS ST-PIERRE, ALCYONIX, CANADA
- CHARLES SYMONS, SOFTWARE MEASUREMENT SERVICES , UNITED KINGDOM
- FRANK VOGELEZANG, SOGETI, THE NETHERLANDS

VERSION 2.0 REVIEWERS 1998/1999 (alphabetical order)

- MORITSUGU ARAKI, JECS SYSTEMS RESEARCH, JAPAN
- FRED BOOTSMA, NORTEL, CANADA
- DENIS BOURDEAU, BELL CANADA, CANADA
- PIERRE BOURQUE, , ÉCOLE DE TECHNOLOGIE SUPÉRIEURE, CANADA
- GUNTER GUERHEN, BÜRHEN & PARTNER, GERMANY
- SYLVAIN CLERMONT, HYDRO QUÉBEC, CANADA
- DAVID DÉRY, CGI, CANADA
- GILLES DESOBLINS, FRANCE
- MARTIN D'SOUZA, TOTAL METRICS, AUSTRALIA
- REINER DUMKE, UNIVERSITY OF MAGDEBURG, GERMANY
- PETER FAGG, UNITED KINGDOM *
- THOMAS FETCKE, GERMANY
- ERIC FOLTIN, UNIVERSITY OF MAGDEBURG, GERMANY
- ANNA FRANCO, CRSSM, CANADA
- PAUL GOODMAN, SOFTWARE MEASUREMENT SERVICES LTD.,, UNITED KINGDOM
- NIHAL KECECI, UNIVERSITY OF MARYLAND, UNITED STATES
- ROBYN LAWRIE, AUSTRALIA
- GHISLAIN LÉVESQUE, UQAM, CANADA
- ROBERTO MELI, DATA PROCESSING ORGANIZATION SRL, ITALY
- PAM MORRIS, TOTAL METRICS, AUSTRALIA*
- RISTO NEVALAINEN, SOFTWARE TECHNOLOGY TRANSFER FINLAND OY, FINLAND *
- JIN NG, HMASTER, AUSTRALIA
- PATRICE NOLIN, HYDRO QUÉBEC, CANADA
- MARIE O'NEILL, SOFTWARE MANAGEMENT METHODS, IRELAND
- JOLIJN ONVLEE, THE NETHERLANDS *
- LAURA PRIMERA, UQAM, CANADA
- PAUL RADFORD, CHARISMATEK, AUSTRALIA
- EBERHARD RUDOLPH, GERMANY
- GRANT RULE, SOFTWARE MEASUREMENT SERVICES LTD., UNITED KINGDOM*
- RICHARD STUTZKE, SCIENCE APPLICATIONS INT'L CORPORATION, UNITED STATES
- ILIONAR SYLVA, UQAM, CANADA
- VINH T. HO, UQAM, VIETNAM

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

[1] A public domain version of the COSMIC-FFP Measurement Manual and other technical reports, including translations into other languages can be found on the Web at URL: http://www.cosmicon.com or http://www.lrgl.uqam.ca/cosmic-ffp.html

Version control

The following table summarizes the changes to 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 workshop[2]
99-10-29	COSMIC core team members	Revised, final comments before publishing "field trial" Version 2.0.
01-05-01	COSMIC Core Team members	Revised for conformity to ISO/IEC 14143-1: 1998 + clarifications on measurement rules to Version 2.1.
03-01-31	COSMIC Measurement Practices Committee members	Revised for conformity to ISO/IEC FDIS 19761: 2002 + further clarifications on measurement rules to Version 2.2

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

Foreword

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

The COSMIC-FFP method was accepted by ISO/IEC JTC1 SC7 in 2000, as a new work item for standardization. Over a period of a year, members of the COSMIC Core Team worked with and via their National Body representatives on Working Group 12 of ISO/IEC JTC1 SC7 to bring the terminology of the COSMIC-FFP method fully into line with existing ISO/IEC terminology, and to make further clarifications of the method. The need for these other clarifications had become apparent from practical use of the method around the world.

The outcome of this effort was the approval in December 2002 of the Final Draft International Standard ISO/IEC 19761 'Software Engineering – COSMIC-FFP – A functional size measurement method' (hereinafter referred to as 'ISO/IEC 19761'). This Version 2.2 therefore brings the Measurement Manual back into line again with the ISO/IEC definition of the method.

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.

No changes of principle in the COSMIC-FFP method have been found necessary as a result of this work of producing the International Standard. These principles have remained unchanged since they were first published in the first draft of the Measurement Manual 1999.

The changes made in producing this Version 2.2 of the Measurement Manual comprise

- the addition of one new principle concerning 'Measurement Viewpoints' and associated definitions. This new principle results in the need for additional conditions for the aggregation of functional sizes of different pieces of software
- an addition to an existing principle requiring the aggregation of data movements at the functional process level before aggregating to the layer level
- two new rules concerning the identification of data movements in a functional process, and one concerning the correspondence of data movements across boundaries
- many editorial improvements and additions to improve consistency of terminology and to improve understanding

These changes are summarized in Appendix E. Here, reference is also made to a paper which describes further reasoning on the necessity for many of the changes.

Two new chapters have also been added in this Version 2.2: chapter 6 (convertibility) and 7 (early COSMIC-FFP).

A new procedure for submitting comments or Change Requests for the Measurement Manual is also added as Appendix F.

COSMIC Metrics Practices Committee Members
January 2003

Table of Contents

Acknowledgments. 2

Version control. 4

Foreword. 5

Table of Contents. 6

GLOSSARY. 8

1. INTRODUCTION. 13

2. OVERVIEW OF THE COSMIC-FFP MEASUREMENT METHOD. 15

2.1 APPLICABILITY OF THE COSMIC-FFP METHOD. 15
2.2 COSMIC-FFP Measurement Process Model. 15
2.3 Extracting functional users requirements. 17
2.4 COSMIC-FFP Mapping phase. 18
2.5 COSMIC-FFP measurement phase. 23
2.6 Sizing early in a project life-cycle: Measurement Scaling. 24
2.7 Functional size measurement context: Purpose, Scope and Measurement Viewpoint. 25

3. THE MAPPING PHASE – RULES AND METHOD. 30

3.1 IDENTIFYING SOFTWARE LAYERS. 31
3.2 IDENTIFYING SOFTWARE BOUNDARIES. 32
3.3 IDENTIFYING FUNCTIONAL PROCESSES. 35
3.4 IDENTIFYING DATA GROUPS. 37
3.5 IDENTIFYING DATA ATTRIBUTES. 41

4. THE MEASUREMENT PHASE – RULES AND METHOD. 43

4.1 IDENTIFYING THE DATA MOVEMENTS. 43
4.2 APPLYING THE MEASUREMENT FUNCTION. 50
4.3 AGGREGATING MEASUREMENT FUNCTION RESULTS. 50

5. COSMIC-FFP MEASUREMENT REPORTING. 52

5.1 LABELLING. 52
5.2 ARCHIVING COSMIC-FFP MEASUREMENT RESULTS. 53

6. COSMIC-FFP CONVERTIBILITY. 54

6.1 Convertibility: Practice and Theory. 54
6.2 A Process for Converting to COSMIC-FFP Size Measurements. 54
6.3 IFPUG Function Points Version 4.1. 55
6.4 MarkII FP Method Version 1.3.1. 56
6.5 Full Function Points Version 1.0. 56
6.6 Relative Sizes on the Four Functional Size Scales. 56

7. EARLY COSMIC-FFP. 57

Appendix A - COSMIC-FFP GENERIC SOFTWARE MODEL. 60

Appendix B - COSMIC-FFP PRINCIPLES IDENTIFICATION. 61

Appendix C - COSMIC-FFP RULES IDENTIFICATION. 64

Appendix D - FURTHER INFORMATION ON SOFTWARE LAYERS. 69

Appendix E - COSMIC-FFP SUB-RELEASES. 72

Appendix F - COSMIC-FFP CHANGE REQUEST AND COMMENT PROCEDURE. 77

***REFERENCES*. 79**

GLOSSARY

The following terms will be used throughout the COSMIC-FFP Measurement Manual, according to the definitions found in this section. Whenever a term was already defined by ISO, such as "Functional Size Measurement" or "Scale", the ISO definition has been adopted for this Measurement Manual.

For many of the terms listed in the Glossary, when appropriate, the suffix '-type' is shown. Since any Functional Size Measurement Method aims to identify 'types' and not 'occurrences' of data or functions, almost invariably throughout this Measurement Manual we will be concerned with 'types' and not 'occurrences'. Consequently, in the main text we will drop the suffix 'type' from these terms for the sake of readability, except when we specifically need to distinguish type and occurrence. This is also the convention adopted in the International Standard (ISO/IEC 19761) definition of the method. But this convention does lead to occasional difficulties when drafting these definitions – see Note 2 of the definition of 'Data movement type' below, which does not appear in the International Standard.

Abstraction: The process of suppressing irrelevant detail to establish a simplified model, or the result of that process.[3]

Base Functional Component (BFC): A Base Functional Component is an elementary unit of the Functional User Requirements defined by an FSM Method for measurement purposes[4].

NOTE The COSMIC-FFP FSM Method defines a data movement type as a BFC.

Base Functional Component Type (BFC Type): A defined category of BFCs[3]. The COSMIC-FFP Measurement Method has four BFC Types, the Entry, Exit, Read and Write (-Types)

Boundary: A conceptual interface between the software under study and its 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 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.

Data attribute type (Synonym 'data element type'): A data attribute type is the smallest parcel of information, within an identified data group type, carrying a meaning from the perspective of the software's Functional User Requirements.

Data group type: A data group type is a distinct, non empty, non ordered and non redundant set of data attribute types where each included data attribute type describes a complementary aspect of the same object of interest (see definition). A data group type is characterized by its persistence (see definition).

Data movement type: A Base Functional Component which moves one or more data attribute types belonging to a single data group type.

NOTE 1 There are four sub-types of data movement types: Entry, Exit, Read and Write (-Types).

NOTE 2 To be precise, it is an *occurrence* of a data movement, not the *type*, that actually *moves* the data attribute *instances* (not *types*). This comment also applies to the definitions of Entry, Exit, Read and Write.

Developer viewpoint: A Measurement Viewpoint that reveals all the functionality of each separate part of the software that has to be developed and/or delivered to meet a particular statement of FUR.

NOTE. For this definition, the 'U' of 'FUR' means strictly 'User' as referred to in Note 1 of the definition of User in the Glossary of this Measurement Manual

---

[3] From ISO/IEC 10746-2. 'Information technology -- Open Distributed Processing -- Reference Model: Foundations'

[4] From "ISO/IEC 14143-1:1998 Information Technology – Software measurement – Functional size measurement – Part 1: Definition of concepts", clause 3.1.

End User viewpoint: A Measurement Viewpoint that reveals only the functionality of application software that has to be developed and/or delivered to meet a particular statement of FUR. It is the viewpoint of Users who are either humans who are aware only of the application functionality that they can interact with, or of peer application software that is required to exchange or share data with the software being measured, or of a clock mechanism that triggers batch application software. It ignores the functionality of all other software needed to enable these Users to interact with the application software being measured.

Entry type: a data movement type that moves a data group type from a user across the boundary into the functional process type where it is required

NOTE 1 In COSMIC-FFP, an entry type is considered to include certain associated data manipulations (e.g. validation of the entered data) – see section 4.1 for details.

NOTE 2 A data manipulation is anything that happens to data other than a movement

Event type: See "Triggering event".

Exit type: a data movement type that moves a data group type from a functional process type across the boundary to the user that requires it.

NOTE An Exit type is considered to include certain associated data manipulations (e.g. formatting and routing associated with the data to be exited) – see section 4.1 for details.

Functional process type (Synonym 'Transaction-type'): A functional process type is an elementary component of a set of Functional User Requirements comprising a unique cohesive and independently executable set of data movement types. It is triggered by one or more Triggering event types either directly, or indirectly via an 'actor'. It is complete when it has executed all that is required to be done in response to the Triggering event type.

NOTE. An 'actor' is a User of the system being measured, acting as an intermediary to convey data about a triggering event to the functional process that has to respond to that event.

Functional Size: A size of the software derived by quantifying the Functional User Requirements[5]

Functional Size Measurement (FSM): Functional Size Measurement is the process of measuring Functional Size[6].

Functional Size Measurement Method: A Functional Size Measurement Method is a specific implementation of FSM defined by a set of rules, which conforms to the mandatory features of ISO/IEC 14143-1: 1998[3].

Functional User Requirements (FUR): The term Functional User Requirements is a sub-set of the user requirements. The FUR represent the user practices and procedures that the software must perform to fulfill the user's needs. FUR exclude Quality Requirements and any Technical Requirements[7].

Input: Data for which the values are independent of the software and which are used by the software at some point during its operation. The generic definition of input used in this manual is notably different from the specific definition used by the International Function Point Users Group (IFPUG). In COSMIC-FFP, input consists of all the entries involved in a particular functional process type.

Layer: A layer is the result of the functional partitioning of the software environment such that all included functional process types perform at the same level of abstraction.

In a multi-layer software environment, software in one layer exchanges data with software in another layer through their respective functional process types. These interactions are hierarchical in nature; when considered in pairs, one layer is a "client" to the other. A "client" layer uses the functional services provided

---

[5] From "ISO/IEC 14143-1:1998 – Information Technology – Software measurement – Functional size measurement – Part 1: Definition of concepts", clause 3.6.

[6] From "ISO/IEC 14143-1:1998 – Information Technology – Software measurement – Functional size measurement – Part 1: Definition of concepts", clause 3.7.

[7] From "ISO/IEC 14143-1:1998 – Information Technology – Software measurement – Functional size measurement – Part 1: Definition of concepts", clause 3.8.

by other subordinate layers. Software items in the same layer can also exchange data. This type of data exchange is usually called "peer-to-peer" data exchange.

Measurement function (COSMIC-FFP): The COSMIC-FFP measurement function is a mathematical function which assigns a value to its argument based on the COSMIC-FFP measurement standard. The argument of the COSMIC-FFP measurement function is the data movement type.

Measurement method[8]: A measurement method is a logical sequence of operations, described generically, used in the performance of measurements.

Measurement procedure[9]: A measurement procedure is a set of operations, described specifically, used in the performance of particular measurements according to a given method.

Measurement process[10]: The process of establishing, planning, performing and evaluating software measurement within an overall project or organisational measurement structure.

Measurement standard (COSMIC-FFP): The COSMIC-FFP measurement standard, 1 Cfsu (Cosmic Functional Size Unit), is defined as one elementary data movement type.

Measurement viewpoint: A Viewpoint of the Functional User Requirements of software defined for the purposes of Functional Size Measurement.

Model[11]: A model is a description or analogy used to help visualize a concept that cannot be directly observed.

Object of interest type: An object of interest type is identified from the point of view of the Functional User Requirements and may be any physical thing, as well as any conceptual objects or parts of conceptual objects in the world of the user about which the software is required to process and/or store data.

NOTE: An Object of interest type is a synonym of 'entity-type' on an entity-relationship diagram, and has the same meaning as the subject of a relation in Third Normal Form.

Operating environment (software): The software operating environment is the set of software operating concurrently on a specified computer system.

Output: Data for which the value depends on the operation of the software and which is therefore created or otherwise modified by the software during its operation. The generic definition of output used in this manual is notably different from the specific definition used by the International Function Point Users Group (IFPUG). In COSMIC-FFP, output consists of all the exits involved in a particular functional process type.

Persistence (of a data group): The persistence of a data group is a quality describing how long the data group is retained in the context of the Functional User Requirements. Three types of persistence are defined: transient (exists only for the life of a functional process), short (beyond the life of a functional process for as long as the software is operational) and indefinite (beyond the duration of operation of the software).

Persistent storage: Storage which enables a functional process to store data beyond the life of the functional process and/or which enables a functional process to retrieve data 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-FFP model, because Persistent Storage is on the software side of the boundary, it is not considered to be a User of the software being measured.

NOTE 2 An example of 'some other process' would be in the manufacture of read-only memory

Purpose of a Measurement: A statement that defines why the measurement is being undertaken, and/or what the result will be used for.

---

8 From "International Vocabulary of Basic and General Terms in Metrology", International Organization for Standardization, Switzerland, 2nd edition, 1993, ISBN 92-67-01075-1

9 From "International Vocabulary of Basic and General Terms in Metrology", International Organization for Standardization, Switzerland, 2nd edition, 1993, ISBN 92-67-01075-1

10 From ISO/IEC 15939:2002, definition 3.24

11 Adapted from Merriam Webster's Collegiate Dictionary, 10th Edition.

Quantity (base)[12]: A base quantity is one of the quantities that, in a system of quantities, are conventionally accepted as functionally independent of one another.

Read type: a data movement type that moves a data group type from persistent storage within reach of the functional process type which requires it.

NOTE A Read type is considered to include certain associated data manipulations necessary to achieve the Read – see section 4.1 for details.

Scale (reference-value): A reference-value scale is, for particular quantities of a given kind, an ordered set of values, continuous or discrete, defined by convention as a reference for arranging quantities of that kind in order of magnitude[13].

Scope of the FSM[14]: The set of Functional User Requirements to be included in a specific Functional Size Measurement instance.

NOTE The Scope of the FSM is determined by the purpose for measuring software. For example, if an organisation needs to know the size of its software portfolio, then the Scope of the FSM will include all Functional User Requirements currently utilised. However, if a project manager is seeking to determine the size of a particular release of software, then the scope will include only those Functional User Requirements impacted by the project

Software[15]: A set of computer instructions, data, procedures and maybe documentation operating as a whole, to fulfill a specific set of purposes, all of which can be described from a functional perspective through a finite set of Functional User Requirements, Technical and Quality Requirements.

Sub-process type: A COSMIC-FFP sub-process type is a data movement type occurring during the execution of a functional process type. There are four sub-types of a COSMIC-FFP sub-process type: Entry, Exit, Read and Write, each of which includes specific associated data manipulation sub-processes – see 4.1 for details. A COSMIC-FFP sub-process type is equivalent to an ISO Base Functional Component Type (BFC Type). A COSMIC-FFP sub-process type expresses only functional user requirements and excludes quality and technical requirements.

Triggering event type: A triggering event type is an event type that occurs outside the boundary of the measured software and initiates one or more functional process types. In a set of Functional User Requirements, each event-type which triggers a functional process type is indivisible for that set of FUR.

NOTE 1 Clock and timing events can be triggering events.

NOTE 2 An event has either happened, or it has not; it is instantaneous.

Unit of measurement[16]: A unit of measurement is a particular quantity, defined and adopted by convention, with which other quantities of the same kind are compared in order to express their magnitudes relative to that quantity. It is to be noted that units of measurement have conventionally assigned names and symbols.

User: A User is any person that specifies Functional User Requirements and/or any person or thing that communicates or interacts with the software at any time[17].

---

[12] From "International Vocabulary of Basic and General Terms in Metrology", International Organization for Standardization, Switzerland, 2nd edition, 1993, ISBN 92-67-01075-1

[13] From "International Vocabulary of Basic and General Terms in Metrology", International Organization for Standardization, Switzerland, 2nd edition, 1993, ISBN 92-67-01075-1

[14] From "ISO/IEC 14143-1:1998 – Information Technology – Software measurement – Functional size measurement – Part 1: Definition of concepts", clause 3.11.

[15] Adapted from Merriam Webster's Collegiate Dictionary, 10th Edition, and La Petit Larousse Illustré, 1996 Edition.

[16] From "International Vocabulary of Basic and General Terms in Metrology", International Organization for Standardization, Switzerland, 2nd edition, 1993, ISBN 92-67-01075-1

[17] From "ISO/IEC 14143-1:1998 – Information Technology – Software measurement – Functional size measurement – Part 1: Definition of concepts", clause 3.17.

NOTE 1 In this Measurement Manual, the term User is restricted to the second of the two meanings given in this definition, i.e. 'any person or thing that communicates or interacts with the software at any time'.

NOTE 2 Users can be human beings, software or engineered devices.

Value (of a quantity)[18]: The value of a quantity is the magnitude of that particular quantity, generally expressed as a unit of measurement multiplied by a number.

Viewpoint: A form of abstraction achieved using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within a system.[19]

Write type: a data movement type that moves a data group type lying inside a functional process type to persistent storage

NOTE A Write type is considered to include certain associated data manipulations necessary to achieve the Write– see section 4.1 for details.

[18] From "International Vocabulary of Basic and General Terms in Metrology", International Organization for Standardization, Switzerland, 2nd edition, 1993, ISBN 92-67-01075-1

[19] From ISO/IEC 10746-2. 'Information technology -- Open Distributed Processing -- Reference Model: Foundations'

1

INTRODUCTION

Software is a major component of many corporate budgets. Organizations recognize the importance of controlling software expenses and analyzing the performance of the amounts allocated to software development and maintenance in order to benchmark against the best in the field. To do so, measures and models using these measures are needed.

Measures are needed for analyzing both the quality and the productivity associated with developing and maintaining software. On the one hand, technical measures are needed by developers to quantify the technical performance of products or services. Technical measures can be used for efficiency analysis; to improve the performance of designs, for instance.

On the other hand, functional measures are needed by developers to quantify the performance of products or services and also from a user's or owner's perspective; for productivity analysis, for instance. Functional measures must be independent of technical development and implementation decisions. They can then be used to compare the productivity of different techniques and technologies.

Function Points Analysis (FPA)[20] is an example of a functional measure. It is available for MIS domain software, where it has been used extensively in productivity analysis and estimation (Abran, 1996; Desharnais, 1988; Jones, 1996; Kemerer, 1987). It successfully captures the specific functional characteristics of MIS software.

However, FPA has been criticized as not being universally applicable to all types of software [Conte, 1986; Galea, 1995; Grady, 1992; Hetzel, 1993; Ince, 1991; Jones, 1988; Jones, 1991; Kan, 1993; Whitmire, 1992]. Here is how D.C. Ince describes Function Points Analysis (FPA) in this regard:

> *"A problem with the function point approach is that it assumes a limited band of application types: typically, large file-based systems produced by agencies such as banks, building societies and retail organizations, and is unable to cope with hybrid systems such as the stock control system with a heavy communication component." [*Ince, 1991, page 283].

It is therefore held that FPA does not capture all the functional characteristics of real-time software. When FPA is applied to such software, it does, of course, generate a value, but this value does not constitute an adequate size measurement. Thus, until the release of FFP version 1.0 in September 1997, there was no FPA equivalent with detailed measurement procedures for real-time software.

Full Function Points (version 1.0) was proposed in 1997[21] with the aim of offering a functional size measure specifically adapted to real-time software. Since then, Full Function Points measurement practice in many organizations and field tests led by UQAM's Software Engineering Management Research Laboratory and the Software Engineering Laboratory in Applied Metrics have demonstrated that Full Function Points not only has the ability to capture the functional size of real-time software, but also to capture the functional size of

---

[20] Albrecht A.J., Gaffney Jr. J.E., "Software function, source lines of code and development effort prediction: a software science validation", IEEE Transactions on Software Engineering, Vol. SE-9, pp. 639-648, November 1983.

[21] St-Pierre D., Maya M., Abran A., Desharnais J.-M., Bourque P., "Full Function Points: Counting Practices Manual", Technical Report 1997-04, Université du Québec à Montréal, Montréal, Canada. Available on the Web at URL: www.lrgl.uqam.ca/ffp.html

technical and system software. Furthermore, these field tests have shown that Full Function Points is also suited to measuring the functional size of MIS software, leading, in such applications, to similar results[22].

The field test results, coupled with the feedback received from organizations which have used Full Function Points since version 1.0 was released in 1997, have motivated the authors to improve the method. Many improvements proposed in this manual were also inspired by the work of the COSMIC group[23] which proposed the principles for a second generation of Functional Size Measurement methods. The results of these efforts are contained in this document, which constitutes version 2.2 of the COSMIC-FFP measurement method.

From Version 2.0 onwards, the COSMIC-FFP measurement method has also been designed to ensure its full compliance with the ISO/IEC 14143-1: 1998 standard and with the COSMIC Group principles.

ABOUT THE COSMIC INITIATIVE

Given the explosive growth and diversity of software contracting and outsourcing, suppliers and customers need more accurate ways of estimating and of measuring performance which must work equally reliably across all types of software. Current methods for measuring the size of software are not always of sufficient strength to meet market needs, or work only for restricted types of software. Industry urgently needs software size measures which are demonstrably more accurate and more widely usable.

The COSMIC group, which started in 1998 aims to meet these needs of a) software suppliers facing the task of translating customer requirements into the size of software to be produced as a key step in their project cost estimating and, b) customers who want to know the functional size of delivered software as an important component of measuring supplier performance.

COSMIC, the COmmon Software Measurement International Consortium, is a voluntary initiative of a truly international group of software measurement experts, both practitioners and academics, from Asia/Pacific, Europe and North America. The aims of the COSMIC-FFP project are to develop, test, bring to market and to seek acceptance of new software sizing methods to support estimating and performance measurement.

The principles of the COSMIC-FFP method were first laid down in 1999. Field trials were successfully conducted in 2000/01 with several international companies and academic institutions. Please see the References section for papers describing trial results. The process of developing an International Standard for the COSMIC-FFP method was started in 2001, the standard was approved in December 2002 and it should be published by ISO in early 2003.

For further information about COSMIC, please visit www.cosmicon.com, or www.lrgl.uqam.ca/cosmic-ffp.

---

[22] Oligny, S.; Abran, A.; Desharnais, J.-M.; Morris, P. , *Functional Size of Real-Time Software: Overview of Field Tests*, in Proceedings of the13th International Forum on COCOMO and Software Cost Modeling, Los Angeles, CA, October 1998.

[23] See www.cosmicon.com for further information.

2

OVERVIEW OF THE COSMIC-FFP MEASUREMENT METHOD[24]

The COSMIC-FFP measurement method is a standardized measure of software functional size. This chapter presents and discusses the applicability of the COSMIC-FFP method and describes the general measurement process model and the COSMIC-FFP models and principles underlying the mapping and measurement phases when this model is used for measuring software. It also discusses the context of software and measurement objectives in which Functional Size Measurement is performed.

2.1 APPLICABILITY OF THE COSMIC-FFP METHOD

The COSMIC-FFP measurement method is designed to be applicable to 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 its complexity 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, lifts and car engines, for process control and automatic data acquisition, and within the operating system of computers.
- Hybrids of the above, as in real-time reservation systems for airlines or hotels for example.

The COSMIC-FFP measurement method has not yet been designed to take into account the functional size of pieces of software, or their parts, which:

- are characterized by complex mathematical algorithms or other specialized and complex rules, such as may be found in expert systems, simulation software, self-learning software, weather forecasting systems, etc.
- process continuous variables such as audio sounds or video images, such as found, for instance, in computer game software, musical instruments and the like.

For such software it is possible, though, to define local extensions to the COSMIC-FFP measurement method. Section 4.1.5 of this manual explains in what contexts such local extensions should be used and provides an example of a local extension. When used, local extensions must be reported according to the conventions presented in section 5 of this manual.

2.2 COSMIC-FFP Measurement Process Model

The COSMIC-FFP measurement method involves applying a set of models, rules and procedures to a given piece of software as it is perceived from the perspective of its Functional User Requirements. The result of the application of these models, rules and procedures is a numerical "*value of a quantity*"[25] representing the functional size of the software, as measured from its Functional User Requirements. These models, rules and procedures are described in chapters 2, 3 and 4 of this manual and are summarized in Appendices B and C.

---

[24] The terms software, input, output, data attributes and users are used in this section of the manual according to the definitions found in the Glossary.

[25] As defined by ISO, see Glossary

The COSMIC-FFP measurement method is designed to be independent of the implementation decisions embedded in the operational artifacts of the software to be measured. To achieve this characteristic, measurement is applied to the Functional User Requirements (or 'FUR') of the software to be measured expressed in the form of the COSMIC-FFP 'generic software model'. This form of the FUR is obtained by a mapping process from the FUR as supplied in or implied in the actual artifacts of the software. Figure 2.2.1, below, depicts this process.

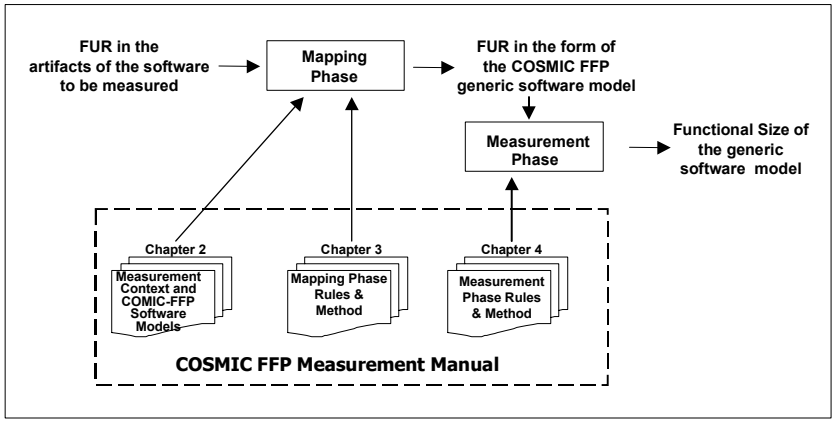


Figure 2.2.1 – COSMIC-FFP measurement process model

This chapter 2 describes and defines certain models and concepts that must be understood before the detailed principles and rules of the mapping phase can be given in the next chapter.

Section 2.3 draws attention to the fact that FUR always exist in some form, or can be derived from, the artifacts of software regardless of whether the software is only in the stage of having its requirements determined, or it is being developed, or after it exists. No detailed rules are given for possible processes to extract FUR to provide the starting point for the mapping process as these would depend on the huge variety of forms of software artifacts and ways of expressing FUR that exist in practice.

Section 2.4 describes the two COSMIC-FFP models that are needed for the Mapping Phase. The first model, the 'software context model' recognizes that the software to be measured may exist in a context of hardware and/or other software layers, and will be separated from its users by a boundary. The software to be measured may itself consist of various layers and peer items within layers. We need to have a common understanding of these concepts and how they relate to each other.

The second model is the COSMIC-FFP 'generic software model' in which the FUR must be expressed before applying the measurement process.

It is important to note that these two models are valuable tools for expressing FUR, even if there is no intention of carrying out any measurements. The two models are compatible with modern requirements determination methods and their consistent use will help produce coherent, unambiguous FUR.

Section 2.5 then describes the principles of the measurement phase and the characteristics of the resulting measurements.

In section 2.6 we digress to deal briefly with the principles of sizing early in the life of a project when the full details of the FUR are not yet known, so it is not yet possible to produce the detailed generic software model of the FUR. We show how it is still possible to estimate an approximate size according to the COSMIC-FFP method using locally defined concepts and scaling factors based on the generic software model.

Finally, Section 2.7 deals with the subjects of the Purpose, Scope and Measurement Viewpoint of the measurement, which we call the 'measurement context'. These three parameters must be carefully considered before starting any measurement process – they constitute the first steps of the measurement process.

The details of the definitions, principles and rules for mapping software FUR to the COSMIC-FFP generic software model are presented in chapter 3 of this manual, and those for measuring the COSMIC-FFP generic software model are presented in chapter 4.

2.3 Extracting functional users requirements

There are many aspects of software. From the perspective of the COSMIC-FFP measurement method, the aspect of interest is the 'functionality' it delivers to its users, by which we mean 'the information processing that the software must perform for the users'. The functionality delivered by software to its users is described through the Functional User Requirements (FUR). These state 'what' the software must do for the users and exclude any technical or quality requirements that say 'how' the software must perform. In practice, FUR sometimes exist in the form of a specific document (requirements specifications, for instance), but often they have to be derived from other software engineering artifacts. As illustrated in Figure 2.3.1 below, FUR can be derived from software engineering artifacts that are produced before the software exists (typically from architecture and design artifacts). Thus, the functional size of software can be measured prior to its implementation on a computer system.

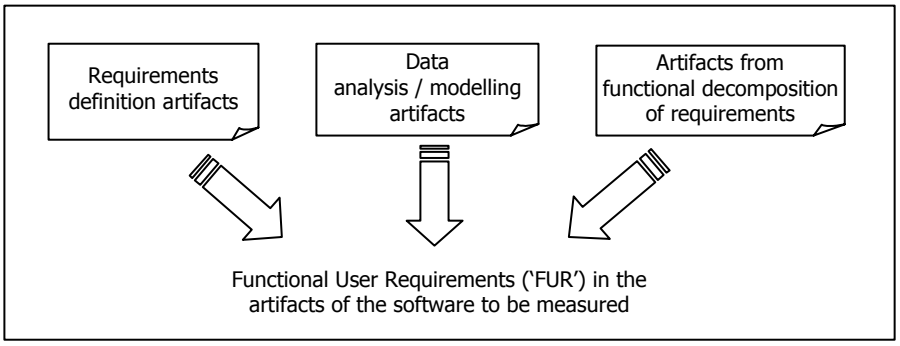


Figure 2.3.1 – COSMIC-FFP pre-implementation Functional User Requirements (FUR) model

In other circumstances, software might be used without there being any, or with only a few, architecture or design artifacts available, and the FUR might not be documented (legacy software, for instance). In such circumstances, it is still possible to derive the software FUR from the artifacts installed on the computer system even after it has been implemented, as illustrated in Figure 2.3.2.

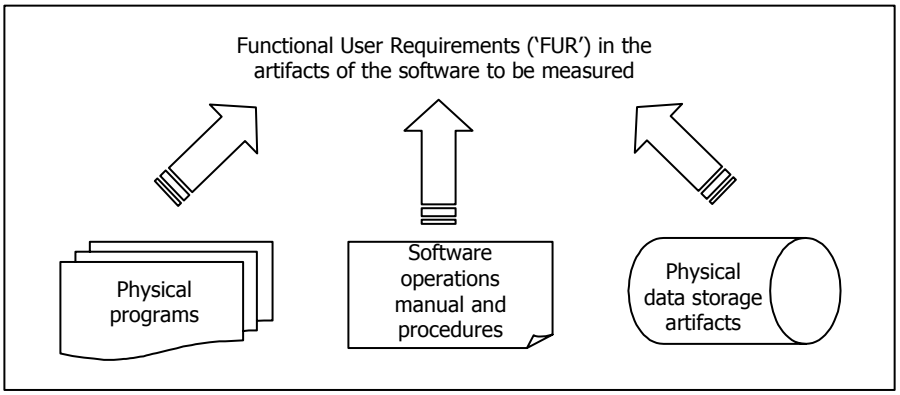


Figure 2.3.2 – COSMIC-FFP post-implementation Functional User Requirements (FUR) model

The effort required to extract the FUR from different types of software engineering artifacts and to express them in the form of the COSMIC-FFP generic software model will obviously vary. The nature of the FUR when so expressed remains constant, though, since regardless of the type of software engineering artifacts from which the FUR are extracted, the resulting FUR always convey only the definition of the functionality delivered by the software to its users.

2.4 COSMIC-FFP Mapping phase

The COSMIC-FFP mapping phase takes as input a statement of Functional User Requirements of a piece of software, obtained as described in section 2.3 above, and, using a defined set of rules and procedures, produces a specific software model (an instance of the COSMIC-FFP generic software model) suitable for measuring functional size. The generic software model produced corresponds to the set of the FUR to be included in the specific FSM instance, as determined by the Purpose, Scope and Measurement Viewpoint of the measurement (see 2.7).

Before describing the COSMIC-FFP generic software model, we must first describe the COSMIC-FFP software context model.

2.4.1 COSMIC-FFP Software Context Model

A key aspect of software Functional Size Measurement is the establishment of what is considered to be part of the software and what is considered to be part of the software's operating environment. Figure 2.4.1.1 below illustrates the generic flow of data from a functional perspective from which the following can be observed:

- Software is bounded by hardware. In the so-called "front-end" direction, software used by a human user is bounded by I/O hardware such as a mouse, a keyboard, a printer or a display, or by engineered devices such as sensors or relays. In the so-called "back-end" direction, software is bounded by persistent storage hardware like a hard disk and RAM and ROM memory.

- The functional flow of data attributes can be characterized by four distinct types of movement. In the "front end" direction, two types of movement (ENTRIES and EXITS) allow the exchange of data with the users across a 'boundary'. In the "back end" direction, two types of movement (READS and WRITES) allow the exchange of data attributes with the persistent storage hardware.

- Different abstractions are typically used for different measurement purposes. For business application software, the abstraction commonly assumes that the users are one or more humans who interact directly with the business application software across the boundary; the 'I/O hardware' is ignored. In contrast for real-time software, the users are typically the engineered devices that interact directly with the software, that is the users ARE the 'I/O hardware'.

- The reader is reminded that in the COSMIC-FFP model, the 'boundary' is defined as 'a conceptual interface between the software under study and its users'. This boundary should not be confused with any line that can be drawn around a box depicting software, or around a software layer in the diagrams of this Measurement Manual. Persistent storage is not considered as a user of the software and is therefore on the software side of the boundary, as seen from the perspective of the software's users.

As an FSM method, COSMIC-FFP is aimed at measuring the size of software based on identifiable Functional User Requirements. Once identified, those requirements are allocated to hardware and software from the unifying perspective of a system integrating these two "components". Since COSMIC-FFP is aimed at sizing software, only those requirements allocated to the software are considered. (Note that in principle, the COSMIC-FFP method could be applied to FUR before they are allocated to software or to anything else, regardless of the eventual allocation decision. However this assertion has still to be tested in practice.)

We will now consider two cases of allocation of software functional requirements: typical business application software and "multi-piece" software. The two cases will help understanding of how the concepts of the software context model can cope with different measurement needs.

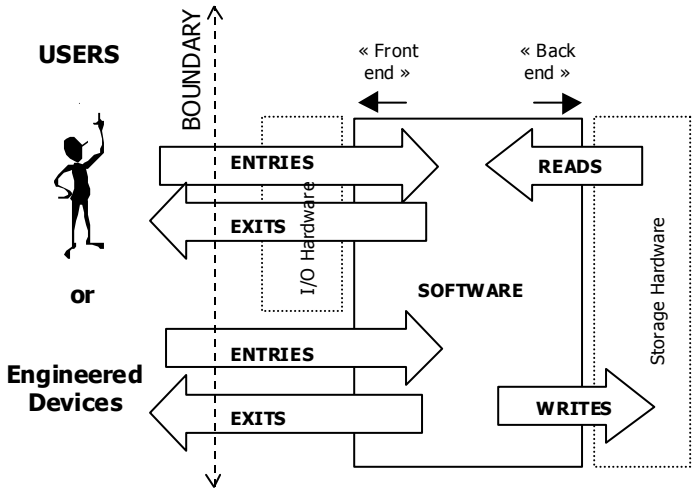


Figure 2.4.1.1 – Generic flow of data attributes through software from a functional perspective

CASE 1 – TYPICAL BUSINESS APPLICATION SOFTWARE

In this example, Functional User Requirements are typically allocated at the software application "level" ("Application X" in Figure 2.4.1.2). The other pieces of software in the environment are generally required and used without modification. The example in Figure 2.4.1.2 below illustrates this.

Since: a) all Functional User Requirements are allocated to a single piece of software, and b) this piece of software can be easily distinguished from other pieces of software within the environment through a well-defined functional interface, it is a relatively simple matter to identify and measure the functionality offered by Application X to its (human) users. In this example and for the purpose of Functional Size Measurement, an abstraction can be made by ignoring the I/O hardware and all software in the environment except Application X. The boundary is then the interface between Application X and its (human) users.

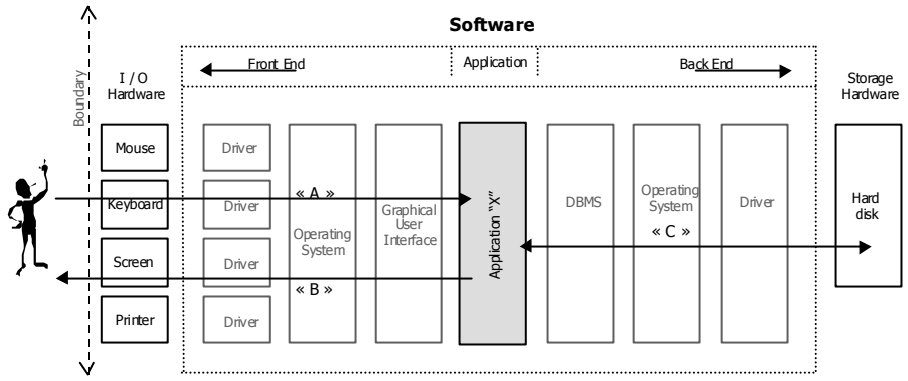


Figure 2.4.1.2 – Example of requirements allocation for Business Application software

CASE 2 – "MULTI-PIECE" SOFTWARE

In this Case 2, let us suppose that the Functional User Requirements allocated to the business application ('Application X in Fig. 2.4.1.2 above) are taken by the Developer and are allocated to a three-component architecture of:

- Graphical user interface
- Business rules
- Data services

The three components are peer components – they are at the same level of abstraction and they exchange data according to an established Application Program Interface – so they are in the same layer. The Developer wishes to size the three components separately from his/her perspective because these three

components will be developed with different technologies and may be implemented on different processors. The FUR of Application X are refined to give the FUR of each of the three components separately.

The other pieces of software in the environment (in other layers) are used without modification and the three software components simply make use of these pieces as needed. The example in Figure 2.4.1.3 below illustrates this

This Figure shows the interactions between the components:

- the three peer components are Users of each other in the one application layer
- in addition, the graphical user interface component of the application layer has the operating system in a different layer as another User
- and the data services component of the application layer has the DBMS layer as another User

There are Boundaries, in the COSMIC-FFP sense, between all these components and their 'neighbours' in the other layers. The COSMIC-FFP measurement process can be applied to size the three components separately as seen from this Developer's perspective.

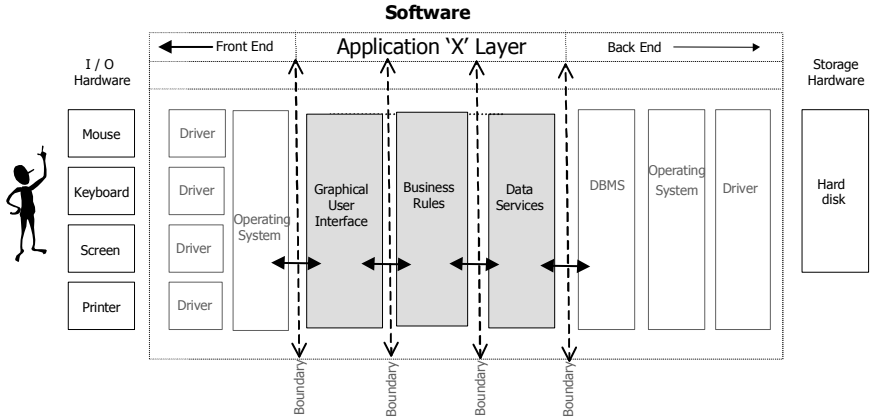


Figure 2.4.1.3 – Example of requirements allocation to "multi-tier" software from the Developer's perspective

Measuring from this perspective, we do not need to invoke the interactions with the human user.

In software engineering practice, Functional User Requirements are not necessarily allocated to a single piece of software. In cases where a software architecture[26] exists early in the construction process, such as the layered and 'tiered' architecture illustrated above, the allocation of Functional User Requirements to specific pieces in this architecture has an impact on the functional size of each piece.

For practitioners interested only in the global size of these requirements, it is not necessary to consider how the requirements are allocated (Case 1 above). For practitioners who are interested in the functional size of portions of the requirements (Case 2 above), this allocation scheme becomes relevant to the identification of the software boundaries – see further on Measurement Viewpoints in section 2.7. Note that the sizes obtained from these two Cases are different size measures.

The COSMIC-FFP measurement method thus provides a tool (the concepts of software layers[27] and tiers) to help differentiate Functional User Requirements allocated at different levels of functional abstraction. More formally, the COSMIC-FFP Software Context Model thus offers the following characteristics:

---

[26] There are many software architecture models in use. The layered model has been used here for the purpose of illustration. Other models could be used if they provide a functional view of the software.

[27] The term "layer" is used in this manual according to the definition found in the Glossary. Although it embodies a concept similar to the one found in software architecture, its use in this manual is not restricted to that form of software architecture.

Characteristic 1 – Layers and Peer software

The software to be measured can be partitioned into one or more pieces so that each piece operates at a different level of functional abstraction in the software's operating environment, and there is a hierarchical relationship between levels of abstraction based on the functional exchanges among them. Each such level is designated as a distinct layer. Each layer encapsulates functionality useful to the layer using its services in the hierarchical relationship and uses the functionality provided by the "lower" layer in this relationship. Note that software within a layer may be developed and need to be sized as separate 'pieces'. Communications between separate pieces of software within the same layer is known as 'peer-to-peer' communication. A more detailed discussion on software layers is presented in section 3.1 and in Appendix D.

Characteristic 2 – Boundary

In any given layer, the piece of software to be measured can be clearly distinguished from software in any other layer with which it exchanges data by a boundary. An implicit boundary thus exists between each identified pair of layers that exchange data. The boundary is a set of criteria, perceived through the Functional User Requirements of the software, which allows a clear distinction to be made between the items that are part of the software (*inside the boundary*) and the items that are part of the software operating environment (*outside the boundary*). By convention, all users of a piece of software being measured lie outside the boundary of this piece of software.

Characteristic 3 – Software users

It is possible to identify one or more users benefiting from the functionality provided by a piece of software within any layer. It follows from the definition that users can be human beings, engineered devices or other software. Consequently, pieces of software within the immediate neighboring layers are considered as users when interacting with the measured piece.

Characteristic 4 – Functional User Requirements

The parts of the software requirements describing the nature of the functions to be provided are designated as Functional User Requirements and are used as the exclusive perspective from which the functional size of the software is to be measured. The parts of the requirements describing how the software functions are to be implemented, like quality or technical requirements for instance, are NOT considered for the purpose of measuring the functional size of the software[28].

2.4.2 COSMIC-FFP Generic Software Model

The COSMIC-FFP generic software model assumes that the following general principles hold true for the software being measured:

General Principle 1: *The software to be mapped and measured is fed by input and produces useful output, or a useful outcome, to users.*

General Principle 2: *The software to be mapped and measured manipulates pieces of information designated as data groups which consist of data attributes.*

Figure 2.4.2.1, below, illustrates the generic software model proposed by the COSMIC-FFP measurement method. According to this model, software functional user requirements can be decomposed into a set of functional processes. Each of these functional processes is a unique set of sub-processes performing either a data movement or a data manipulation.

[28] According to "ISO/IEC 14143-1:1998 – Information Technology – Software measurement – Functional size measurement – Part 1 : Definition of concepts".

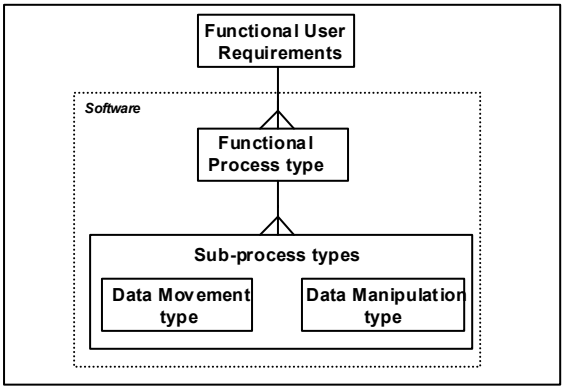


Figure 2.4.2.1 – A generic software model for measuring functional size

The generic COSMIC-FFP software model distinguishes four types of data movement sub-process: entry, exit, read and write, as identified in the context model (Figure 2.4.1.1). All data movement sub-processes move data contained in exactly one data group. Entries move data from the users across the boundary to inside the functional process; exits move data from inside the functional process across the boundary to the users; reads and writes move data from and to persistent storage. These relationships are illustrated in Figure 2.4.2.2 below.

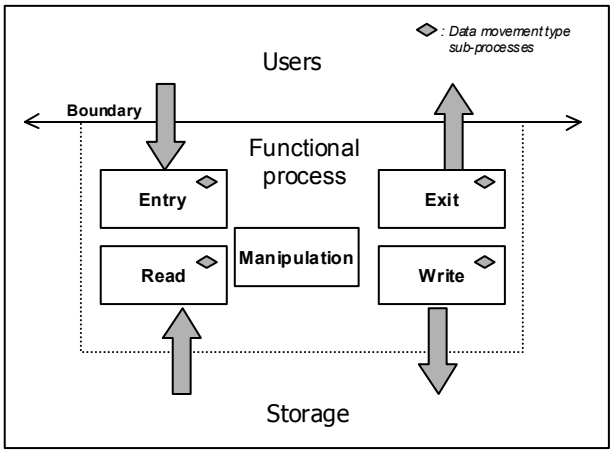


Figure 2.4.2.2 – The sub-process types and some of their relationships

As a first approximation in this version of the measurement method, data manipulation-type sub-processes, illustrated in Figure 2.4.2.2, are not recognized separately but are considered to be associated with or part of specific data movement sub-processes. (From now on, for brevity, we will refer to 'data movements', rather than to 'data movement sub-processes'.). The reason is that the necessary concepts and definitions, required to measure data manipulations are still the subject of much debate and discussion among software engineering specialists. The specific type of data manipulation functionality considered to be included within each type of data movement is described in section 4.1. Given this approximation it is easy to see why the standard COSMIC-FFP method is suitable for sizing 'movement-rich' types of software, such as in most business applications and much real-time software, but is unsuited for sizing 'manipulation-rich' (or 'algorithm-rich') software.

However, section 4.1.5 provides a mechanism for defining a local extension to the COSMIC-FFP method that enables an organization to account explicitly for data manipulation functionality. For such cases, special reporting conventions are presented in section 5.1.

By using the concepts, definitions and structure of the COSMIC-FFP measurement method, the Functional User Requirements extracted from the artifacts of a piece of software are mapped onto the COSMIC-FFP generic software model, thereby instantiating it. This instantiated model will contain all the elements required for measuring its functional size, while hiding information not relevant to Functional Size Measurement.

The COSMIC-FFP measurement rules and procedures are then applied to this instantiated COSMIC-FFP generic software model in order to produce a value of a quantity representing the functional size of the software. Therefore, two distinct and related phases are necessary to measure the functional size of software: mapping the artifacts of the software to be measured onto the COSMIC-FFP generic software model and then measuring the specific elements of this model.

The mapping phase takes as input the Functional User Requirements extracted from the artifacts of the software to be measured (as they are found/documented within the organization or as they are inferred from the existing physical software) and produces as output an instance of the COSMIC-FFP generic software model. This generic software model is instantiated using the concepts and definitions found in chapter 3 and summarized in Appendices B and C. The COSMIC-FFP 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. This representation of the COSMIC-FFP generic software model is presented in Appendix A.

2.5 COSMIC-FFP measurement phase

The COSMIC-FFP measurement phase takes as input an instance of the COSMIC-FFP generic software model and, using a defined set of rules and procedures, produces a value of a quantity the magnitude of which is directly proportional to the functional size of the model, based on the following principle:

PRINCIPLE – The COSMIC-FFP Measurement Principle
The functional size of software is directly proportional to the number of its Data Movements.

By convention, this numerical value of a quantity is then extended to represent the functional size of the software itself.

Among the characteristics of the set of rules and procedures governing the production of this value of a quantity are the following:

Characteristic 1 – Measurement function

Each instance of a data movement is assigned a numerical quantity, according to its type, through a measurement function.

Characteristic 2 – Unit of measure

The measurement standard, that is, 1 Cfsu (Cosmic Functional Size Unit), is defined by convention as equivalent to a single data movement.

Characteristic 3 – Additivity

The functional size of a functional process is defined as the arithmetic sum of the values of the measurement function, as applied to each of its constituent data movements. By extension, the functional size of any piece of software in any layer in the software model is the arithmetic sum of the functional sizes of the functional processes of that piece of software.

The functional size of any required functional *change* to a piece of software is by convention the arithmetic sum of the functional sizes of all the added, changed and deleted functional data movements of that piece of software.

Characteristic 4 – Sub-unit of measure

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-FFP 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-FFP unit of measurement, 1 Cfsu, has been fixed at the level of one data movement. However, caution is clearly needed when comparing the sizes measured in Cfsu of two

different pieces of software where the average number of data attributes per data movement differs sharply across the two pieces of software.

According to these characteristics and using 1 Cfsu as the unit of measure, it is worth noting that the smallest theoretical functional size for a piece of software is 2 Cfsu, based on General Principle 1, as the smallest functional process must have at least one Entry, and either one Exit or one Write. Again according to these characteristics, there is no upper limit to the functional size of a piece of software and, notably, there is no upper limit to the functional size of any of its functional processes.

The functional size is therefore derived from the FUR expressed in the COSMIC-FFP generic software model by applying the documented set of rules and procedures presented in chapter 4 and summarized in Appendices B and C.

The COSMIC-FFP measurement of functional size does not presume to measure all aspects of software 'size'. Thus as we have seen, the influence of the number of data attributes per data movement on software size is not captured by this measurement method. Also, 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 2.1 on the applicability of the method. Nevertheless, the COSMIC-FFP size measure is considered to be a good approximation for the method's stated purpose and domain of applicability.

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 upon 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.

2.6 Sizing early in a project life-cycle: Measurement Scaling

The COSMIC-FFP measurement method is currently documented at a level where all data movements are clearly identified. This is illustrated in Figure 2.6.1 below. Early in a software development project life-cycle however, the FUR may not have been worked out at the level of detail of data movements. Yet still the size has to be estimated from specifications that may exist at higher levels of abstraction, such as 'sub-systems' or 'Use Cases'.

By analogy, a house can be roughly sized before it is built by measuring surfaces from draft plans, before all details of have been worked out. Such plans are drawn to a defined scale. Whatever scale is chosen, it is always documented (1:4 or 1:10, for instance) and there is a strict and constant equivalence between the dimensions measured on the plan and the corresponding dimensions measured on the actual building.

This well-defined and constant scaling factor is absent from the artifacts produced by software engineering processes. In this regard, software engineering is more analogous to biology where dimensions are measured at various "levels" (molecular, cell, tissue, organ, system). Although there are some scale factors that are inherent to these "levels", the ratios are not constant.

Returning to software measurement, using the COSMIC-FFP measurement method on artifacts carrying a higher level of abstraction also requires the use of a scaling factor; such an application of the COSMIC-FFP measurement method will be described in chapter 7 of the manual. Work is in progress to identify these scaling factors in general terms. Results will be reported in a later edition of this document.

Meanwhile, if scaling factors are required, they should be determined by local definitions and calibration on existing software. As a very simple example, local calibration might determine that a (locally-defined) Use Case comprises, on average, 3.5 functional processes, each of average size 8 Cfsu. Hence the average size of a Use Case (or its 'scaling factor'), according to this local definition, is 3.5 x 8 = 28 Cfsu per Use Case. Thus, with this calibration, identifying the number of Use Cases early in a development project's life will provide a basis for making a preliminary estimate of software size in units of Cfsu. .

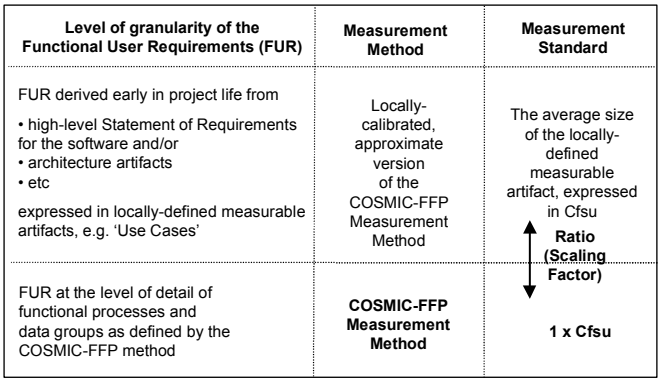


Figure 2.6.1 – COSMIC-FFP scaling

2.7 Functional Size Measurement context: Purpose, Scope and Measurement Viewpoint

This section addresses three vital aspects of software Functional Size Measurement, namely the Purpose, Scope and Viewpoint of a measurement.

Before starting a measurement using the COSMIC-FFP method it is imperative to carefully define the Purpose, the Scope and the Measurement Viewpoint. This may be considered as the first step of the measurement process. Equally, it is important to record the data listed in section 5.2 arising from this step when recording the result of the measurement. Failure to define and to record these parameters consistently will lead to measurements that cannot be interpreted reliably and compared, or be used for estimating.

The first part of this section gives some background explanation of why these three factors are important and uses analogies to show why these factors are taken for granted in other fields of measurement, and hence must also be considered in the field of software Functional Size Measurement.

The second part of this section gives the formal definitions of the three factors and gives examples of how to apply them before starting a measurement using the COSMIC-FFP method.

Background

There are many reasons to measure the functional size of software, just as there are many reasons to measure the surface areas of a house. In a particular context, 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, it will be useful to measure the functional size of software some time after it has been put into production, just as it might be useful to measure the surface areas of a house after it had been delivered and the owner has moved in.

The reason why a measure is taken has an impact, albeit at times subtle, on what is being measured, either the object itself or its representation, 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 the regular geometrical definition of a rectangular surface: the product of its length and width.

Likewise, measuring the functional size of software prior to its development is based on the software "plans"; a collection of artifacts produced prior to development. The required dimensions (numbers of Base Functional Components) are extracted from the artifacts using appropriate conventions (from the perspective of the Functional User Requirements, excluding technical and quality aspects) and the size is calculated according to a specific measurement function.

To further pursue the house analogy, measuring its surface areas after it has been constructed entails a somewhat different measurement procedure when the required dimensions (length and width) are extracted

from the building itself using a different tool (measuring tape). Although the physical object being measured differs (the house rather than its plans), the dimensions, the unit of measure (including the scaling convention) and the measurement principles will remain the same. Alternatively, the house can still be measured based on the plans, provided of course that the plans have been kept up to date.

Likewise, measuring the functional size of software after it has been put into production entails a somewhat different measurement procedure when the required dimensions are extracted from the various artifacts. Although the nature of these artifacts differs, the dimensions, the unit of measure and the measurement principles remains 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. Clearly, the three sizes could be different. The same reasoning applies to the measurement of software.

This COSMIC-FFP Measurement Manual defines and describes dimensions, units of measure and measurement principles for measuring the functional size of software. It also provides some guidelines for helping to identify those dimensions in some generally recognized artifacts. Based on the purpose of the measurement, it is up to the measurer to determine the most appropriate artifacts to use to perform a specific measurement.

Equally, it is important to define the Scope of the measurement, which is derived from the Purpose, before commencing a particular measurement exercise. Example: if the purpose is to measure the functional size of software delivered by a particular project team, it will first be necessary to define the Functional User Requirements of all the various components to be delivered by the team. These might include the FUR of software which was used once only to convert data from software which is being replaced. If then the purpose is changed to measure the size which the users have available once the new software is operational, this would be smaller, as the FUR of the software used for conversion would not be included in the scope of the measured size.

Finally, it is essential to define the Measurement Viewpoint, which again may follow from the Purpose. The Measurement Viewpoint, in general terms, determines the level of detail that can be seen and therefore measured, within the Scope. The Measurement Viewpoint is highly significant, because in general measurements taken from different Measurement Viewpoints cannot meaningfully be compared or added together.

As an analogy in the measurement of the floor areas of an office, by convention this can be undertaken according to four different sets of rules involving different levels of detail, as follows. (N.B. the Scope – the particular office – is the same for all four Measurement Viewpoints.)

- The Building Owner has to pay taxes on the office. From this Viewpoint, 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 has another Viewpoint. He 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 has another Viewpoint. He 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 has another Viewpoint. He is interested in the 'Net-net Sq. M.', i.e. only the space usable for offices.

In software, we do not yet have standard definitions of Measurement Viewpoints, but they are needed for our measurement purposes. The most obvious Measurement Viewpoint needed is that of the 'Developer', who needs to identify the functionality of all the components that have to be developed to meet the FUR as he/she receives them, e.g. for the purposes of project planning and estimating.

In addition, there is a Measurement Viewpoint needed for measuring application software which is that of the 'End Users'. End Users (see definition below) who are human will only be aware of the functionality provided at the application level and that is all they may be concerned with. But this may be just 'the tip of the iceberg' compared with the functionality seen from the Developer Measurement Viewpoint.

Measurements taken using the COSMIC-FFP method from these two different Measurement Viewpoints will therefore give quite different sizes.

(N.B. The names of 'End Users' and 'Developer' are intended to be generic terms given to these two types of Measurement Viewpoints; they should not be confused with the 'Sponsor' of the measurement. It is perfectly possible that a particular (human) user of the software might sponsor size measurements from both the End User and Developer Measurement Viewpoints. Whilst it may be important for local management reasons to know who sponsored a specific measurement, it should not matter who sponsored it for understanding the result.)

Users of the COSMIC-FFP method are therefore alerted to the importance of defining the Purpose of the measurement and of defining the Scope and the Measurement Viewpoint before commencing a particular measurement exercise.

Definitions and application to use of the COSMIC-FFP method

The terms 'Purpose' and 'Scope' are used according to their normal English meaning, but 'Measurement Viewpoint' has a specialized meaning in this context.

DEFINITION – Purpose of a Measurement
A statement that defines why the measurement is being undertaken, and/or what the result will be used for.

DEFINITION – Scope of a Measurement
The set of Functional User Requirements to be included in a specific Functional Size Measurement exercise.

DEFINITION – Measurement Viewpoint[29] (and Viewpoint and Abstraction)
A Viewpoint of the Functional User Requirements of software defined for the purposes of Functional Size Measurement, where 'Viewpoint' is defined in ISO/IEC 10746-2. 'Information technology -- Open Distributed Processing -- Reference Model: Foundations') as: 'A form of abstraction achieved using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within a system,' (The word 'system' in this context is interpreted as meaning 'the software being measured'.) and where 'abstraction' is defined in ISO/IEC 10746-2 as: 'The process of suppressing irrelevant detail to establish a simplified model, or the result of that process.'

The following examples illustrate the above.

Purpose of a Measurement. Examples would be:

- To measure the size of the FUR as they evolve, as input to an estimating process

[29] In the FDIS ISO/IEC 19761 standard for the COSMIC-FFP method, there is a non-normative Note which states that 'The Purpose and Scope of a specific FSM when combined are referred to as the 'Viewpoint'. Subsequently, it was realized that 'Measurement Viewpoint' is a separate concept and that for the purposes of this Measurement Manual, we should build upon the existing definitions of ISO/IEC 10746-2., which include the term 'Viewpoint'

- 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.

The Purpose helps the Measurer to determine:

- the Scope to be measured and hence the artifacts which will be needed for the measurement
- the Measurement Viewpoint to be used for the measurement
- 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-FFP measurement method should be used, or whether a locally-derived approximation version of the COSMIC-FFP method should be used (e.g. early in a project's life-cycle, before the FUR are fully elaborated)

Scope of a Measurement. The following are generic types of Scope:

- A contractually-agreed statement of FUR
- A Project Team's *delivered* work-output (i.e. including that obtained by exploiting existing software parameters and re-usable code, 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 software used for data conversion and subsequently discarded, and utilities and testing software developed specifically for this project, but *excluding* all new functionality obtained by changing parameters and exploiting re-usable code)
- A Layer
- A Re-usable Object
- A Software Package
- An Application
- An Enterprise Portfolio

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.

Specific Measurement Viewpoints. As outlined above, for consistent measurement, measurers need to define and use consistently a very limited number of Measurement Viewpoints. The two Measurement Viewpoints that most obviously need to be standardized are the Measurement Viewpoint of the 'End Users' of an item of application software and the Measurement Viewpoint of the Developer of the software to be provided to meet the FUR.

DEFINITION – End User Measurement Viewpoint
A Measurement Viewpoint that reveals only the functionality of application software that has to be developed and/or delivered to meet a particular statement of FUR. It is the viewpoint of Users who are either humans who are aware only of the application functionality that they can interact with, or of peer application software that is required to exchange or share data with the software being measured, or of a clock mechanism that triggers batch application software. It ignores the functionality of all other software needed to enable these Users to interact with the application software being measured.

DEFINITION – Developer Measurement Viewpoint
A Measurement Viewpoint that reveals all the functionality of each separate part of the software that has to be developed and/or delivered to meet a particular statement of FUR. NOTE. For this definition, the 'U' of 'FUR' means strictly 'User' as referred to in Note 1 of the definition of User in the Glossary of this Measurement Manual

The Developer may see that a statement of FUR implies that more than one separate 'major component' has to be developed and/or delivered. This can arise if, due to the requirements, parts of the software have to be developed using different technologies, and/or will execute on different processors and/or belong to different layers of a given architecture, and/or are to be realized as separate peer items in the same layer'. Note also, that for any one such major component, this Measurement Viewpoint is also that of all Users (as defined in the COSMIC-FFP method) of that major component.

The Measurement Viewpoint of the 'End User' is the one from which '1st Generation' Function Point methods such as that of IFPUG and MkII FPA, were originally designed to measure a functional size (though this comment is made with hindsight; the concept of a Measurement Viewpoint did not exist when the methods were designed). This Measurement Viewpoint will be of interest in the domain of Business Application software and particularly when comparing measurements made with existing Function Point methods against those using the COSMIC-FFP method using this Measurement Viewpoint. But this Measurement Viewpoint can also be of interest for measuring the interactions of humans with real-time software. Conventions needed for sizing from this Measurement Viewpoint will be given in various places in this Measurement Manual.

The Measurement Viewpoint of the Developer of 'Major Components' is the one that will probably be of most general interest, across all software domains, for those needing measurement data for practical performance measurement and estimating purposes. Given the above definition of Measurement Viewpoint, measurements should ideally only be made in 'a selected set of architectural concepts and structuring rules'. Examples would be:

- Multi-layer architectures used in the domain of telecoms software
- Multi-tier architectures such as are commonly used to achieve interfaces between, say, a PC and existing application software over the Internet, where all the 'major components' are peer items in the same layer.

3

THE MAPPING PHASE – RULES AND METHOD

The COSMIC-FFP Measurement Method considers the measurement of the functional size of software through two distinct phases: the mapping of the software to be measured to the COSMIC-FFP generic software model and the measurement of specific aspects of this generic software model.

This chapter presents the rules and method[30] for the mapping process. The general method for mapping software to the COSMIC-FFP generic software model is summarized in Figure 3.1.

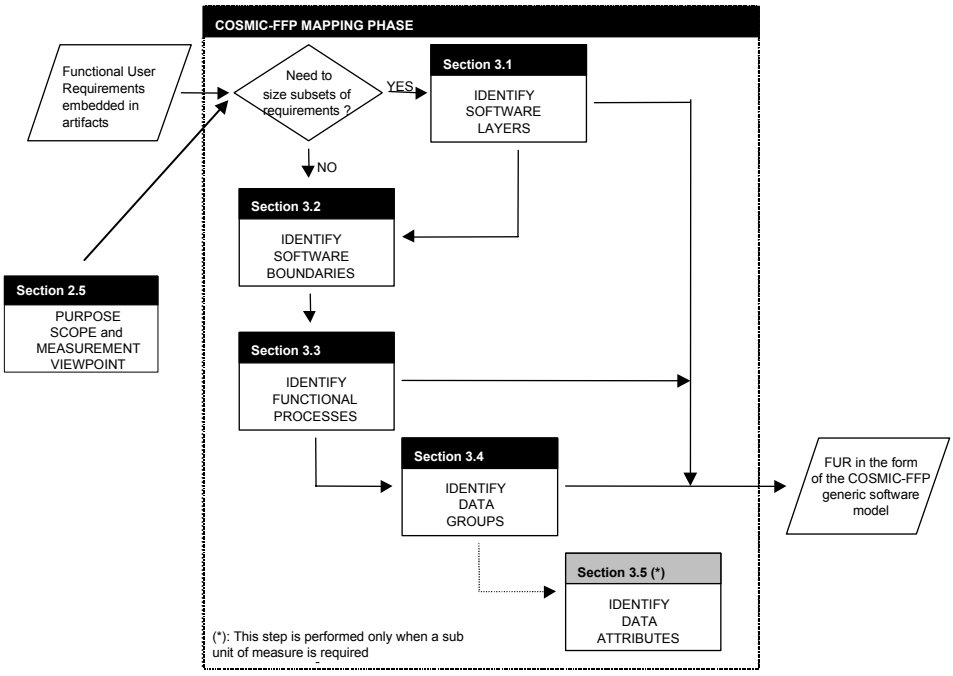


Figure 3.1 – General method of the COSMIC-FFP mapping process

Each step in this method is the subject of a specific sub-section (indicated in the step's title bar in Figure 3.1) where the definitions and rules are presented, along with some guiding principles and examples.

The general method outlined in Figure 3.1 above is designed to be applicable to a very broad range of software artifacts.

A more systematic and detailed procedure would provide precise mapping rules for a larger collection of highly specific artifacts, thus diminishing the level of ambiguity when generating the COSMIC-FFP generic software model. Such a procedure 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. Future

[30] The term "method" is used as defined by ISO. See Glossary for details

versions of the COSMIC-FFP Measurement Method could include mapping procedures for the known and most widely available software engineering methodologies.

3.1 IDENTIFYING SOFTWARE LAYERS

Functional User Requirements may state explicitly, may imply, or the measurement analyst may infer, that the FURs apply to software in different layers or peer items. Alternatively, the measurement analyst may be faced with sizing existing software which appears to be in different layers or peer items. In both cases, let us assume that the Purpose, Scope and Measurement Viewpoint (see Section 2.7) indicate that these layers must be measured separately. For example, the Purpose is estimating, where the layers will be developed using different technologies. We then need guidance to help decide if the FUR or the software comprises one or more layers or peer items. Layers may be identified according to the following definitions and principles.[31].

DEFINITION – Layer

A layer is the result of the functional partitioning of the software environment such that all included functional processes perform at the same level of abstraction.

In a multi-layer software environment, software in one layer exchanges data with software in another layer through their respective functional processes. These interactions are hierarchical in nature; when considered in pairs, one layer is subordinate to the other. Software in a subordinate layer provides functional services to software in other layers using its services. The Measurement Method defines "peer-to-peer" exchanges as two items of software in the same layer exchanging data.

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:

PRINCIPLES – Layer

a) Software in each layer delivers functionality to its 'own' Users (a User may be a human, a physical device or other software, e.g. software in another layer)
b) Software in a subordinate layer provides functional services to software in another layer using its services.
c) Software in a subordinate layer could perform without assistance from software in the layer using its services.
d) Software in one layer might not perform properly if software in a subordinate layer on which it depends is not performing properly.
e) Software in one layer does not necessarily use all the functionality supplied by software in a subordinate layer.
f) In a hierarchy of layers, software in any one layer can be a subordinate to a higher layer for which it provides services.
g) Software in one layer and in a subordinate layer can physically share and exchange data. However, the software in each layer will interpret the data attributes differently and/or group them in different data groups.
h) Software that shares data with other software shall not be considered to be in different layers if they identically interpret the data attributes that they share.

[31] The concept of layers presented in this manual is different from the concept of "layered architecture". Although there are elements common to both concepts, the COSMIC-FFP layers are meant as a tool to help the practitioner in the identification of the separate components that have to be sized and of their boundaries. If a specific architectural paradigm is used within an organization, then the measurer should establish the equivalence between specific architectural objects in this paradigm and the concept of layers as defined in this manual.

Once identified, each layer can be registered in an individual line in the generic software model matrix (Appendix A), with the corresponding label. Further discussion of the layer concept can be found in Appendix D.

RULES – Layers

a) Functional service software packages such as database management systems, operating systems or device drivers, are generally considered as distinct layers.
b) If software is conceived using an established architectural paradigm of layers as meant here, then use that paradigm to identify the layers.
c) The software application level is generally considered to occupy the highest layer level.
d) When in doubt, use the concepts of coupling and cohesion (see Appendix D) to distinguish between interacting layers.

The concept of software layers is a tool to help differentiate Functional User Requirements allocated at different levels of functional abstraction. There are many software architecture models in use. The layered model is used here to provide a functional view of the software. Other models could be used if they provide a functional view of the software, fully or partly.

3.2 IDENTIFYING SOFTWARE BOUNDARIES

This step consists in identifying the boundary of each piece of software (depending on the Measurement Viewpoint, e.g. each layer or each peer item within a layer if the 'Developer' Measurement Viewpoint) to be measured.

DEFINITION – Boundary

The boundary is defined as a conceptual interface between the software under study and its users.

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 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.

DEFINITION – Users

A User is defined as any person or thing that communicates or interacts with the software (being measured) at any time.

NOTE 1 In this Measurement Manual, the term User is restricted to the second of the two meanings given in this definition, i.e. 'any person or thing that communicates or interacts with the software at any time'.

NOTE 2 Users can be human beings, software or engineered devices.

Boundary identification is an iterative process. The exact boundary will be refined as the mapping process progresses. Once identified, the candidate boundary must comply with the following principle:

PRINCIPLE – Boundary

By definition, there is a boundary between each identified pair of layers where one layer is the user of another, and the latter is to be measured. Similarly, there is a boundary between any two distinct pieces of software in the same layer if they exchange data in 'peer-to-peer' communications; in this case each piece can be a user of its peer.

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

RULES – Boundary

a) Identify triggering events, then identify the functional processes enabled by those events. The boundary lies between the triggering events and those functional processes.

b) For real-time or technical software, use the concept of layers to assist in the identification of the boundary (section 3.1).

ABOUT THE IDENTIFICATION OF A BOUNDARY AND THE USE OF LAYERS

The following three example Cases illustrate typical approaches used to identify software boundaries with the help of the layer concept when measurements are made from two different measurement viewpoints.

CASE 1: BOUNDARY OF APPLICATION SOFTWARE FROM THE END-USER MEASUREMENT VIEWPOINT

The first Case arises when carrying out measurements on application software from the End User Measurement Viewpoint. In this case we identify the data movements through the I/O interface with the (human) users. In this abstraction, the boundary lies between the users and the software, as illustrated in Figure 3.2.1. The I/O devices are 'invisible' in this abstraction and are ignored. Data movements to and from persistent storage are identified from the perspective of the users and take place 'inside' the software. The users are not aware of any persistent storage devices; functionally, all the user wants is for data to be made persistent, or to be retrieved from persistent storage.

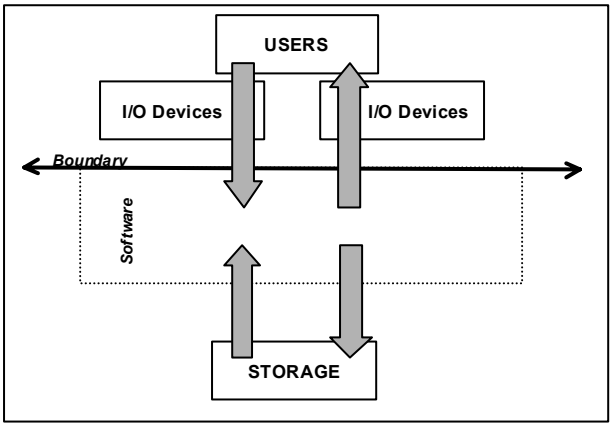


Figure 3.2.1 – Illustration of the boundary for a measurement from the End User measurement viewpoint

CASE 2: BOUNDARIES OF SOFTWARE FROM THE DEVELOPER MEASUREMENT VIEWPOINT

Case 2 is of a need to measure functional sizes as seen from the Developer Measurement Viewpoint, where the FUR distinguish the functional processes to be provided in the application software and in the persistent storage device driver software. (We ignore the probable presence of an operating system for simplicity.)

The concept of layers would then lead to the identification of an additional boundary between these two layers: the application layer and the device driver layer. The users of the application layer could be either the

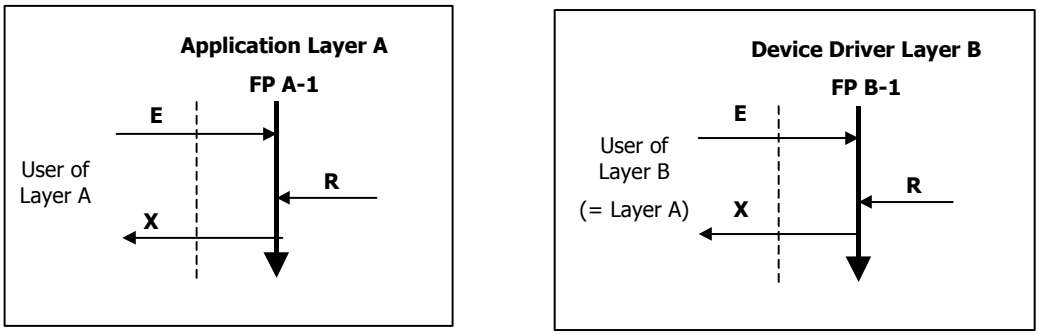
human users as in Case 1 or the I/O devices; it is immaterial to this Case. The application layer is the user of the driver layer.

When we have to map FUR involving multiple layers to the COSMIC-FFP generic software model it is important to understand how any individual data movement appears on either side of an inter-layer boundary. For a Write data movement this is very simple. In Case 2, a Write by the application layer corresponds to an Entry in the device driver layer.

But the case of a Read is slightly more complicated and is best illustrated with the conventions of 'message sequence diagrams'. Figs. 3.2.2 a) and b) present the solution to the Case 2 situation where an enquiry functional process in the application layer results in a Read being issued by the application layer, and the Read is handled by the device driver layer. Figs. 3.2.2 a) and b) show each layer separately, using 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



Figures 3.2.2 a) and b). Solution to Case 2 for a Read issued by an application layer, presented as two message sequence diagrams, one for each layer

Note that a "read" in the application layer as seen by its user corresponds to an "entry and exit pair" of data movements in the driver layer as seen by its user. Data movements that take place between the application layer and the device driver layer must be identified in the software in each layer since specific functionality must be provided within each of the two layers to handle these movements. This functionality is thus accounted for in each layer. The apparent mis-match between the number of data movements of the 'read' of the application layer and the 'entry/exit pair' of 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 (see further section 4.1).

CASE 3: BOUNDARIES OF SOFTWARE FROM THE DEVELOPER MEASUREMENT VIEWPOINT FOR A READ HANDLED BY CLIENT/SERVER SOFTWARE

Using this diagramming convention, we can now take this illustration further into Case 3 which shows the functional processes when the Read is handled by a client/server solution. In this Case, application layer A calls on the services of software component C1 in Layer C to satisfy the Read. This component has a client/server relationship with another peer component, C2, in the same layer, from which it obtains the Read data.

The model for the application layer is the same as for Case 2 as shown in Fig. 3.2.2 a).

Fig. 3.2.3 shows that a Functional Process FP C1-1 of Component C1 receives the Read request from its user, application layer A, as an Entry (as in Case 2, Fig. 3.2.2 b)) and issues an Exit to component C2. This data movement crosses the boundary between C1 and C2 and hence corresponds to an Entry to component C2. C2 obtains the data via a Read, and sends it back to C1 via an Exit. A separate Functional Process C1-2 of component C1 receives this data movement as an Entry. C1-2 passes the data on as an Exit to satisfy the Read request of application layer A. Components C1 and C2 are 'peer users' of each other in this exchange. Case 3 therefore requires 7 data movements to satisfy the Read request compared with the 3 data movements required by the device driver of Layer B in Case 2 which obtains the data 'locally'.

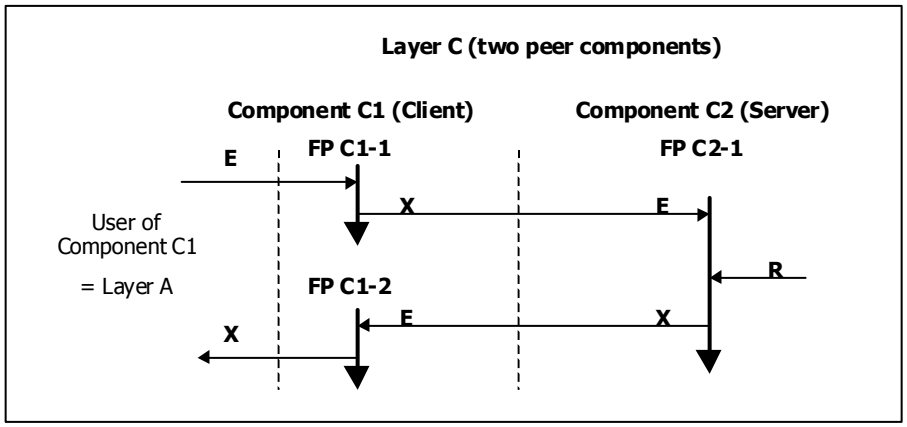


Figure 3.2.3 Case 3 solution where the Read is obtained by a data exchange between a client and server components

Note that in the Cases 2 and 3 above we have ignored, for simplicity, the possible use of 'return codes' in a Read data movement, which might lead to the generation of error messages in this Developer Measurement Viewpoint.

Regardless of the Measurement Viewpoint chosen, the model of entries and exits across a boundary from and to users, and reads and writes within the software from and to persistent storage, must be applied consistently to each layer (or peer item within a layer) from the perspective of the users and the FUR of that layer (or peer item). See 4.1 for a summary of the rules for the correspondence of data movements across boundaries.

3.3 IDENTIFYING FUNCTIONAL PROCESSES

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

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 one or more triggering events either directly, or indirectly via an 'actor'. It is complete when it has executed all that it is required to be done in response to the triggering event (-type).

NOTE. An 'actor' is a User of the system being measured, acting as an intermediary to convey data about a triggering event to the functional process that has to respond to that event.

DEFINITION – Triggering event

A triggering event is an event(-type) that occurs outside the boundary of the measured software and initiates one or more functional processes. In a set of Functional User Requirements, each event(-type) which triggers a functional process is indivisible for that set of FUR.
NOTE 1 Clock and timing events can be triggering events.
NOTE 2 As far as software is concerned, an event has either happened, or it has not; it is instantaneous.

Once identified, candidate functional processes must comply with the following principles:

PRINCIPLES – Functional process

a) A functional process is derived from at least one identifiable Functional User Requirement,
b) A functional process is performed when an identifiable triggering event occurs,
c) A functional process comprises at least two data movements, an entry plus either an exit or a write,
d) A functional process belongs to one, and only one, layer
e) A functional process terminates when a point of asynchronous timing is reached. A point of asynchronous timing is reached when the final (terminating) data movement in a sequence of data movements is not synchronized with any other data movement.

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

RULES – Functional process

a) By definition, a triggering event-type gives rise to a triggering Entry-type, that is, the movement of a data group-type defined as comprising a certain number of data attribute-types. If it can happen that certain occurrences of the triggering Entry data group contain only sub-sets of the values of all the data attribute-types, then the possible existence of such occurrences does not imply the existence of a different event-type or functional process-type.
   For instance, if an occurrence of a specific event-type triggers the entry of a data group comprising data attributes A, B and C, and then another occurrence of the same event-type triggers an entry of a data group which has values for attributes A and B only, this is not considered to be a different triggering event-type. It is considered to be the same for the purpose of identifying COSMIC-FFP functional processes. Consequently, only one entry and one functional process are identified, manipulating data attributes A, B and C.
b) In the context of real-time software, a functional process is also triggered by an event. It terminates when a point of asynchronous timing is reached. A point of asynchronous timing is equivalent to a self-induced wait state.

Once identified, each functional process can be registered on an individual line, under the appropriate layer, in the generic software model matrix (Appendix A), under the corresponding label.

3.4 IDENTIFYING DATA GROUPS

This step consists in identifying the data groups referenced by the software to be measured.

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 (see definition). A data group is characterised by its persistence (see definition).

DEFINITION – Data group persistence[32]

Persistence of a data group is a quality describing how long the data group is (to be) retained in the context of the Functional User Requirements. Three types of persistence can be defined:

Transient: The data group does not survive the functional process using it.
Short: The data group survives the functional process using it, but does not survive when the software using it ceases to operate.
Indefinite: The data group survives even when the software using it ceases to operate.

In practice COSMIC-FFP currently only distinguishes 'Transient' from 'Persistent' (i.e. short or indefinite) data groups

Data persistence is a characteristic used to help distinguish between the four types of sub-processes, as further explained in chapter 4 of this manual. Once identified, each candidate data group must comply with the following principles:

PRINCIPLES – Data group

a) A data group must be materialized within the computer system supporting the software.
b) Each identified data group must be unique and distinguishable through its unique collection of data attributes.
c) Each data group must be directly related to one Object of interest described in the software's Functional User Requirements. (Attention is drawn to 4.1.3 Principle e) for a Read, and to 4.1.4, principle f) for a Write.)

Once identified, each data group can be registered in an individual column in the generic software model matrix (Appendix A), under the corresponding label.

ABOUT THE MATERIALIZATION OF A DATA GROUP

In practice, the materialization of a data group takes three forms, as illustrated in Figure 3.4.1:

a) through a physical record structure on a persistent storage device (file, database table, ROM memory, etc.); this form of data group has short or indefinite persistence,

[32] Adapted from early draft version for ISO/IEC 14143-5 Information Technology – Software measurement – Functional size measurement method – Part 5: Definition of Functional Domains.

b) through a physical structure within the computer's volatile memory (data structure allocated dynamically or through a pre-allocated block of memory space); this form of data group has short persistence or is transient,

c) through the clustered presentation of functionally related data attributes on an I/O device (display screen, printed report, control panel display, etc.), this form of data group has varying persistence, depending on the functional use of the data and the medium used by the device.

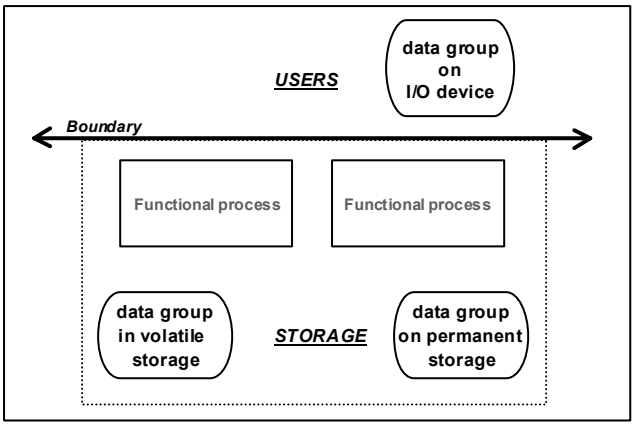


Figure 3.4.1 – COSMIC-FFP data groups and their relation to other aspects of the COSMIC-FFP generic software model

ABOUT THE IDENTIFICATION OF A DATA GROUP

The data group definition and principles are intentionally broad in order to be applicable to the widest possible range of software. This quality has a drawback in the fact that their application to the measurement of a specific piece of software might be difficult. Therefore, the following rules, drawn from Functional Size Measurement practice, might assist in the application of the principles to specific cases.

RULES – Data group

a) APPLICATION TO BUSINESS APPLICATION SOFTWARE. Measurement practice has established that, in business application software, a data group is identified for each 'entity-type' (or 'Third Normal Form' relation) found in the normalized data model of the measured software. These are usually data groups showing indefinite persistence and the software is required to store data about the entity-types concerned.

In COSMIC-FFP, we use the term 'Object of interest' instead of 'entity-type' or 'TNF relation' in order to avoid using terms related to specific software engineering methods.

Examples: in the domain of management information software, an Object of interest could be 'employee' (physical) or 'order' (conceptual) – the software is required to store data about employees or orders.

Furthermore, data groups showing transient persistence (i.e. data groups about transient Objects of interest) are formed whenever there is an ad hoc enquiry which asks for data about some 'thing' about which data is not stored with indefinite persistence, but which can be derived from data stored with indefinite persistence. In such cases the transient Object of interest is the subject of the entry data movement in the ad hoc enquiry (the selection parameters to derive the required data) and of the exit data movement containing the desired attributes of the transient Object of interest.

Example: we form an ad hoc enquiry against a personnel database to extract a list of names of all employees aged over 35. This group is a transient Object of interest. The entry is a data group containing the selection parameters. The exit is a data group containing the list of names.

b) APPLICATION TO REAL-TIME SOFTWARE. Real-time software measurement practice has established that data groups for this type of software often take the following forms:

- 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 cause an interrupt.
- A message-switch may receive a message data group as an Entry and route it forward unchanged as an Exit. The attributes of the message data group could be, for example, 'sender, recipient, route_code and message_content', and its Object of interest is 'Message'.

- Common data structure, representing Objects of interest that are mentioned in the Functional User Requirements, which are held in volatile memory and accessible to most of the functional processes found in the measured software,

- 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,

- Files, commonly designated as "flat files", representing Objects of interest mentioned in the Functional User Requirements, which are held in a persistent storage device.

EXAMPLES OF DATA GROUP IDENTIFICATION

This section presents three business application examples aimed at illustrating how data groups are identified from Functional User Requirements.

EXAMPLE 1

Consider the following Functional User Requirements:

"Permanent data must be maintained about employees and the departments in the organization. From this data the software must be able to supply the employment history of an employee, that is, the list of departments in which this employee has worked in the past".

Let us look at the data groups mentioned in this Requirement. The two most obvious groups are "Employee" and "Department", holding data about their respective Objects of interest as shown in Figure 3.4.2. There are also two implicit relationships between these two data groups: an employee may be attached to many departments over his/her career and a department may contain many employees at any given time. These two relationships are also illustrated in Figure 3.4.2 by the notation of the two lines between 'employee' and 'department', respectively.

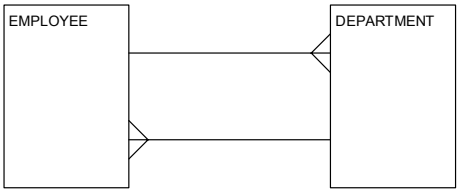


Figure 3.4.2 – Examples of data groups with a "many to many" relationship

These two 'one-to-many' relationships, when combined, confirm that the employee/department relationship is actually 'many-to-many'. In practice, such a many-to-many relationship always implies the existence of another Object of interest, which we can call the 'department/employee history' that has attributes of its own (for example the date an employee joined a department and the date he/she left). This relationship therefore has its own data group and, according to modern software engineering practices, it will be materialized as an independent data structure, as illustrated in Figure 3.4.3 below.

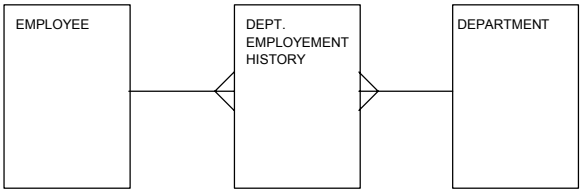


Figure 3.4.3 – Examples of data groups as identified by COSMIC-FFP

For the purpose of measuring the functional size of this Functional User Requirement, the COSMIC-FFP measurement rules would identify three distinct Objects of interest and hence data groups in this example. Those three data groups would be characterized by indefinite persistence.

EXAMPLE 2

Consider the following Functional User Requirement:

"The software must support ad hoc inquiry against a personnel database to extract a list of names of all employees aged over 35. "

Let us look at the data groups mentioned in this Requirement. The most obvious data group is "Employee"; it is characterized (implicitly) by indefinite persistence. The selection parameters, delivered by an entry, are the sole attribute of a second data group, characterized by no persistence. A third data group, also characterized by no persistence, is formed by the query results, displayed on the screen following the inquiry, since they characterized another Object of interest that could be labeled "Query results". The query parameters and the query results are both data groups about the same transient Object of interest, namely 'the group of employees aged over 35'. These three data groups, along with their associated data movements, are illustrated in Figure 3.4.4.

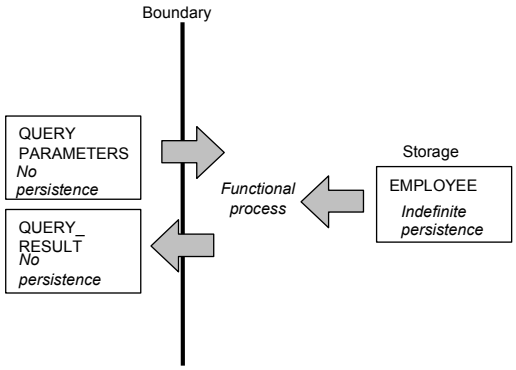


Figure 3.4.4 – Examples of data groups as identified by COSMIC-FFP

EXAMPLE 3

Consider the following Functional User Requirement:

"The software holds all data about the Company's personnel in a file. An attribute of each employee is the Department ID to which each employee currently belongs. A separate table lists all Department ID's, giving also the Division name to which each Department belongs. A report is required that lists all current Company employees by name, sorted by Division and by Department-within-Division. The report should show sub-totals of the number of employees for each Department ID and for each Division name, and the total number for the whole Company"

To produce this report requires one functional process, which must be triggered by an Entry. The functional process must involve two Read Data Movements, of the 'Employee' and of the 'Department' Objects of

interest, to obtain the data it needs from persistent storage. As the report will show the names of all employees, there must also be an Exit for the 'Employee' Object of interest since all names must be printed.

However, the report must also show the total number of employees at three levels of aggregation, namely Department, Division and Company. These totals are attributes of the three Object of interest-types Department, Division and Company respectively.

Hence we must identify one Exit for each of these. In total, therefore, this functional process has one Entry, two Reads and four Exits, i.e. its size is 7 Cfsu. (The viewpoint is that of the End-user, and we ignore the possible need for an error message.)

The Example 3 illustrates how the COSMIC-FFP method accounts for the varying size of outputs in business application software. The same reasoning should be applied to inputs. The general lesson of this Example is that the 'input' and 'output' of every functional process must be analysed to identify the persistent and transient Objects of interest ('any … thing … about which the software is required to process and/or store data'). Each Object of interest identified in the input and output then corresponds to one Entry or Exit respectively.

3.5 IDENTIFYING DATA ATTRIBUTES

This step consists in identifying the set of data attributes referenced by the software to be measured. In this version of the Measurement Method, it is not mandatory to identify the data attributes. They might be identified if a sub-unit of measure is required, as presented in section 2.5 above (Characteristic 4).

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.

DEFINITION – Types of data attribute

Among all data attributes referenced by the software to be measured, three types are distinguished. Two of them are valid for the purpose of measuring the functional size of software and one of them is not valid for this purpose.

VALID TYPES OF DATA

- **Data describing or representing the user's world**. This type of data attribute has a meaning that is independent of the way it is processed. An employee's name and salary have a meaning without consideration of how they will be processed by a set of functional processes, which would print pay cheques, for instance.

- **Contextual data**. This type of data specifies what, when or how other data are to be processed by the measured software. Its functional meaning is fully understood when considering the context in which it is referenced. Consider the following Functional User Requirement, for instance: "*print report XYZ at 10:00*". The specified time (10:00) does not bear its functional meaning without consideration of the context of "report XYZ". It determines when this report is to be printed.

INVALID TYPES OF DATA

- **Implementation technical data**. This type of data is created only because a specific implementation technique has been chosen.

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

PRINCIPLES – Data attribute

a) A data attribute must represent a valid type of data.
b) A data attribute must represent the smallest piece of data referenced by the measured software from the perspective of Functional User Requirements.

EXAMPLES – Data attribute

a) IN THE CONTEXT OF BUSINESS APPLICATION SOFTWARE
Data element types registered in the data dictionary,
Data attributes appearing in a conceptual or logical data model,

b) IN THE CONTEXT OF REAL-TIME APPLICATION SOFTWARE
A voltage signal received from a single sensor

ABOUT THE ASSOCIATION OF DATA ATTRIBUTES AND DATA GROUPS

Within the context of software's Functional User Requirements, there is a distinction between an Object of interest and the data necessary to describe it; the former is represented by one or more data groups, while the latter is represented by the data attributes in those data groups. Therefore, in theory, a data group might contain only one data attribute if this is sufficient, from the perspective of the functional requirements, to describe the Object of interest. In practice, such cases occur commonly in real-time type of software (e.g. the Entry recording the tick of a real-time clock); they are less common in business application type of software.

4

THE MEASUREMENT PHASE – RULES AND METHOD

The COSMIC-FFP Measurement Method considers the measurement of the functional size of software through two distinct phases: the mapping of the software to be measured to the COSMIC-FFP generic software model and the measurement of specific aspects of this model.

The derivation of functional size of the software being measured is independent of the effort required to develop or maintain the software, of the method used to develop or maintain the software and of any physical or technological components of the software.

This chapter presents the rules and method for the measurement phase. The general method for measuring a piece of software from the COSMIC-FFP generic software model is summarized in Figure 4.1 below.

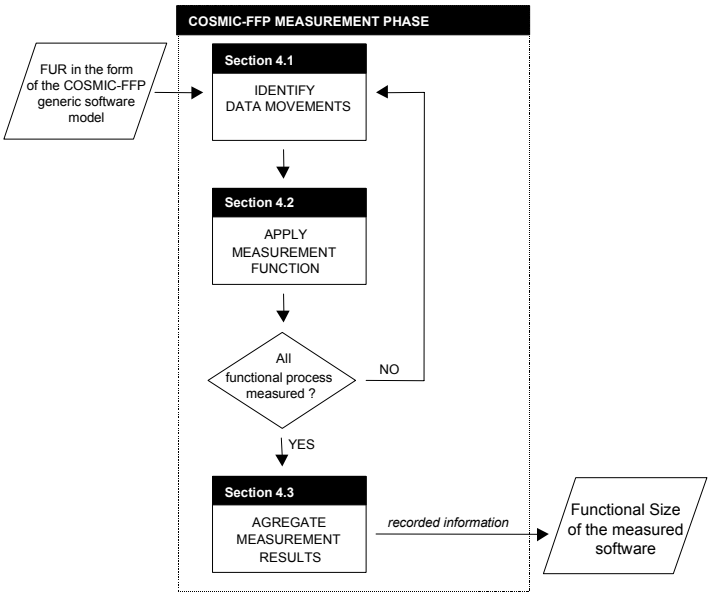


Figure 4.1 – General procedure for the COSMIC-FFP measurement phase

Each step in this method is the object of a specific section of this chapter where the definitions and principles to apply are presented, along with some rules and examples. A summary of the principles and rules can also be found in Appendices B and C respectively.

4.1 IDENTIFYING THE DATA MOVEMENTS

This step consists in identifying the data movement sub-process-types (Entry Exit Read and Write-types) of each functional process-type.

DEFINITION – COSMIC-FFP Data Movement

A COSMIC-FFP data movement is a component of a functional process that moves one or more data attributes belonging to s single data group.

A COSMIC-FFP data movement occurs during the execution of a functional process. There are four sub-types of a COSMIC-FFP data movement: entry, exit, read and write, each of which includes specific associated data manipulation. A COSMIC-FFP data movement type is equivalent to an ISO Base Functional Component Type (BFC Type).

DEFINITION – Entry (E)

An ENTRY (E) is a data movement type that moves a data group from a user across the boundary into the functional process s where it is required.

An ENTRY (E) does not update the data it moves. Note also that in COSMIC-FFP, an entry is considered to include certain associated data manipulations (e.g. validation of the entered data).

DEFINITION – Exit (X)

An EXIT (X) is a data movement type that moves a data group from a functional process across the boundary to the user that requires it.

An EXIT (X) does not read the data it moves. Note also that in COSMIC-FFP, an exit is considered to include certain associated data manipulations (e.g. formatting and routing associated with the data to be exited).

DEFINITION – Read (R)

A READ (R) is a data movement type that moves a data group from persistent storage within reach of the functional process which requires it.

Note also that in COSMIC-FFP, a READ is considered to include certain associated data manipulation sub-processes necessary to achieve the Read.

DEFINITION – Write (W)

A WRITE (W) is a data movement type that moves a data group lying inside a functional process to persistent storage.

Note also that in COSMIC-FFP, a WRITE is considered to include certain associated data manipulation sub-processes necessary to achieve the Write.

RULE – Data Movement 'de-duplication'

(The term 'de-duplication' means 'the process of eliminating duplicate copies from a list'.)

A data movement of a given type (E, X, R or W) moving any one Data Group (that is, data about a single Object of interest), is *in general*, identified only once in any one Functional Process. Examples:

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. For sizing purposes, we identify only one Read.

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, we identify only one Read, because we identify data movement types, not occurrences.

However, there may be legitimate exceptions to this rule. Example:

In real-time software, we can envisage a Functional Process where there is a *Functional User Requirement* to repeat a Read before the process ceases in order to check that the data have not changed since the first Read in the same process. In this case, if there is additional and/or different data manipulation associated with the second Read following discovery that the data have changed, then the Read should be identified twice in the one functional process.

RULE – Read and Write Data movements in 'Update' Functional Processes

Most commonly in on-line business application software, a FUR to update persistent data results in *two* separate functional processes. The first is a 'retrieve-before-update' in which the data about the Object(s) of interest to be updated is/are first Read and displayed. This is followed by the 'update' functional process in which the changed data is made persistent by one or more Write data movements. The retrieve-before-update functional process containing only the Read(s) and no Write(s) allows the human user to verify that the correct record(s) is/are being addressed. The subsequent 'update' functional process allows the user to enter the data that should be changed, added or deleted and to complete the updating via the Write(s).

However, there are also various circumstances which can lead to a FUR for *one* functional process in which a Read is followed by a Write of the 'same data', without human intervention. By 'same data' we mean either:

- the Data Group that is written is identical to the Data Group that was read but its values have been updated, or
- the Data Group that is written is for the same Object of interest as the Data Group that was read, but the Data Group that is written differs from the one that was read due, for example, to the addition of data attributes.

In such cases a Read and a Write of the 'same data' should be identified in the one functional process.

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

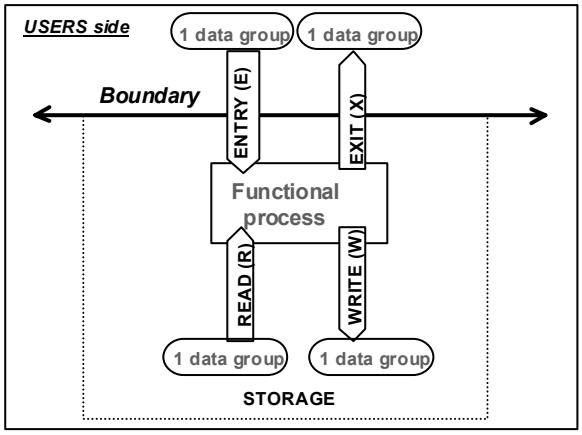


Figure 4.1.1 – The four types of data movement and their relationship with the functional process and data groups

Arising from the description of the correspondence of data movements across boundaries between layers and between peer items within a layer given in 3.2, we have the following rules.

RULES – Correspondence of Data Movements across Boundaries

In the Developer Measurement Viewpoint, when one item of software is the user of another software item:

a) An Entry to one item of software from a peer item in the same layer corresponds to an Exit in the peer item
b) An Exit from one item of software to a peer item in the same layer corresponds to an Entry in the peer item
c) A Write is a data movement to persistent storage. It does not cross the boundary of the functional process to which it belongs. But if the Write is delegated to software in another 'lower' layer, this will correspond to an Entry across the boundary of the lower layer
d) A Read is a data movement from persistent storage. It does not cross the boundary of the functional process to which it belongs. But if the Read is delegated to software in another 'lower' layer, this will correspond to an Entry/Exit pair across the boundary of the lower layer
e) In all four of the above cases, include the data movements on each side of the boundary in the size of their respective components

ON THE DATA MANIPULATIONS ASSOCIATED WITH DATA MOVEMENTS

Sub-processes are, as shown in Fig. 2.4.2.1, elementary functional data movements or data manipulations occurring inside the boundary of the measured software. However, by current COSMIC-FFP convention, we do not recognize the separate existence of data manipulation sub-processes. Instead, depending on its type, each data movement represents, by convention, the following detailed data manipulation functionality:

ENTRY DATA MOVEMENT

An Entry includes all formatting and presentation manipulations required by the users along with all associated validations of the entered data, to the extent where these manipulations do not involve another

data movement. For instance[33], an Entry includes all cited manipulations EXCEPT the Read(s) that might be required to validate some entered codes or to obtain some associated descriptions.

An Entry also includes any 'request to receive the Entry' functionality.

EXIT DATA MOVEMENT

An Exit includes all data formatting and presentation manipulations required by the users, including processing required for routing the output to these users, to the extent where these manipulations do not involve another data movement. For instance[32], an Exit includes all processing to format and prepare for printing some data attributes, including the human-readable field headings EXCEPT the Read(s) or Entries that might be required to supply the value (descriptions, for instance) of some of the printed data attributes.

READ DATA MOVEMENT

A Read includes all processing and/or computation needed in order to read the data, to the extent where these manipulations do not involve another type of data movement. For instance, a Read includes all mathematical computation and logical processing required to retrieve a data group or a number of data attributes from a data group in persistent storage but not the manipulation of those attributes after the data group has been obtained.

A Read also includes any 'request to Read' functionality

WRITE DATA MOVEMENT

A Write includes all processing and/or computation to create the data attributes of a data group to be written to the extent where these manipulations do not involve another type of data movement. For instance, a write includes all mathematical computation and logical processing required to update a group of data attributes to be written EXCEPT any Reads or Entries that might be required to supply the value of other data attributes included in the group to be written.

4.1.1 IDENTIFYING ENTRIES (E)

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

PRINCIPLES – Entry (E)
a) The data movement receives data attributes lying outside the software boundary, from the user side. b) The data movement receives data attributes from only one data group, that is data about a single Object of interest. If input from more than one data group is received, identify one ENTRY for each group. c) The data movement does not exit data across the boundary, read or write data. d) Within the scope of the functional process where it is identified, the ENTRY is unique, that is, the processing and data attributes identified are different from those of other ENTRIES included in the same functional process.

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

RULES – Entry (E)
a) Clock-triggering events are always external to the software being measured. Therefore, an event occurring every 3 seconds is associated with an ENTRY moving one data attribute, for instance. Note that even if such a triggering event is generated periodically not by hardware, but by a software functional process, the latter can be ignored in the measurement since it occurs, by definition, outside of the boundary of the software being

[33] This example applies when measuring from the End User Measurement Viewpoint, and from the Developer Measurement Viewpoint when measuring major components. 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.

[32]

measured.

b) Unless a specific functional process is necessary, obtaining the time from the system's clock is not considered as an ENTRY. For instance, 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".

4.1.2 IDENTIFYING EXITS (X)

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

PRINCIPLES – Exit (X)

a) The data movement sends data attributes outside the boundary, to the user side.
b) The data movement sends data attributes belonging to only one data group, that is, data about a single Object of interest. If data belonging to more than one data group are sent outside the software boundary, identify one EXIT for each referenced data group.
c) The data movement does not receive data from outside the boundary, or read or write data.
d) Within the scope of the functional process where it is identified, the sub-process is unique; that is, the processing and data attributes identified are different from those of other EXITS included in the same functional process.

The following rule might be useful to confirm the status of a candidate EXIT data movement:

RULE – Exit (X)

When measuring size from the End User Measurement Viewpoint, by convention all software messages generated without user data (e.g. confirmation and error messages) are considered to be separate occurrences of one message-type. Therefore, a single EXIT is identified to represent all these messages within the scope of the functional process where these messages are identified.

For instance, consider functional processes A and B identified within the same layer. "A" can potentially issue 2 distinct confirmation messages and 5 error messages to its users and "B" can potentially issue 8 error messages to its users. In this example, one EXIT would be identified within functional process "A" (handling 5+2=7 messages) and a separate EXIT would 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".

4.1.3 IDENTIFYING READS (R)

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

PRINCIPLES – Read (R)

a) The data movement retrieves data attributes from a data group in persistent storage.
b) The data movement retrieves data attributes belonging to only ONE data group, that is, data about a single Object of interest. Identify one READ for each Object of interest for which data attributes are retrieved in any one functional process (see also the 'De-duplication' Rule which may over-ride this rule).
c) The data movement does not receive or exit data across the boundary or write data.
d) Within the scope of the functional process where it is identified, the data movement is

unique, that is, the processing and data attributes identified are different from those of any other READ included in the same functional process.

e) During a functional process, the Read (or a Write) of a data group can only be performed on the data describing an Object of Interest to the User. Constants or variables which are internal to the functional process, or intermediate results in a calculation, or data stored by a functional process resulting only from the implementation, rather than from the FUR, are not data groups and are not taken into account in the functional size.

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".

4.1.4 IDENTIFYING WRITE (W)

Once identified, the candidate WRITE sub-process must comply with the following principles:

PRINCIPLES – Write (W)

a) The data movement moves data attributes to a data group on the persistent storage side of the software.
b) The data movement moves the values of data attributes belonging to only ONE data group, that is data about a single Object of interest. Identify one WRITE for each Object of interest for which data attributes are referenced in any one functional process (see also the 'De-duplication' Rule which may over-ride this Rule).
c) The data movement does not receive or exit data across the boundary, or read data.
d) Within the scope of the functional process where it is identified, the data movement is unique, that is, the processing and data attributes identified are different from those of any other WRITE included in the same functional process.
e) A requirement to delete a data group from persistent storage is measured as a single Write data movement.
f) During a functional process, the step of storing a data group that does not persist when the functional process is complete is not a Write; examples are updating variables which are internal to the functional process or producing intermediate results in a calculation.

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".

4.1.5 LOCAL EXTENSIONS

The COSMIC-FFP 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. However, it may be that within the local environment of an organization using the COSMIC-FFP Measurement Method, it would be possible to account for this functionality in a way which is meaningful as a local standard.

For this reason, the COSMIC-FFP Measurement Method has provision for local extensions. 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 functional size units for this exceptional functionality. An example of a local extension standard could be:

> *In our organization, we assign one Local FSU for mathematical algorithms such as (list of locally meaningful and well-understood examples…). We assign two Local FSU's for (another list of examples), etc.*

When such local extensions are used, the measurement results must be reported according to the special convention presented in section 5.1 of this manual.

4.2 APPLYING THE MEASUREMENT FUNCTION

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

DEFINITION – COSMIC-FFP measurement function

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

DEFINITION – COSMIC-FFP measurement standard

The COSMIC-FFP measurement standard, 1 Cfsu (Cosmic Functional Size Unit) is defined as the size of one elementary data movement.

According to this measurement function, each instance of a data movement (entry, exit, read or write) identified in step 1 (Table 4.1) receives a numerical size of 1 Cfsu.

4.3 AGGREGATING MEASUREMENT FUNCTION 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 principles and rule.

PRINCIPLES – Aggregation of measurement results

a) For each functional process, the functional sizes of individual data movements are aggregated into a single functional size value by arithmetically adding them together.

$$Size_{Cfsu}(\text{functional process}_i) = \sum \text{size}(\text{entries}_i) + \sum \text{size}(\text{exits}_i) + \sum \text{size}(\text{reads}_i) + \sum \text{size}(\text{writes}_i)$$

b) For any functional process, the functional size of changes to the Functional User Requirements is aggregated from the sizes of the corresponding modified data movements according to the following formula.

$$Size_{Cfsu}(\text{Change}(\text{functional process}_i)) = \sum \text{size}(\text{added data movements}) + \sum \text{size}(\text{modified data movements}) + \sum \text{size}(\text{deleted data movements})$$

c) The size of each piece of software to be measured within a layer shall be obtained by aggregating the size of the new and any changed functional processes within the identified FUR for each piece

d) Sizes of layers or of pieces of software within layers may be added together only if measured from the same Measurement Viewpoint.

e) Furthermore, sizes of pieces of software within any one layer or from different layers may be added together only if it makes sense to do so, for the Purpose of the measurement. (For example, if various major components are developed using different technologies, by different project sub-teams, then there may be no practical value in adding their sizes together.)

An example of the consequences of Principles b) and c) above, a requested change to a piece of software might be: "add one new Functional Process of size 6 Cfsu, and in another Functional Process add one Data Movement, make changes to three other Data Movements and delete two Data Movements." The total size of the requested change is 6 + 1 + 3 + 2 = 12 Cfsu.

RULE – Aggregation of measurement results
When using the measurement matrix found in Appendix A, aggregation can be performed by adding the number of data movements found in each row to obtain the size of the functional process and then by adding the sizes of the functional processes to obtain the size of the layer.

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 the whole layer, depending on the purpose and scope of the each measurement exercise.

Furthermore, 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.

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, aggregation will typically be performed at the layer level since layers are often not implemented with the same technology. For instance, consider software where the application layer is to be implemented using 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.

On the other hand, if a project team has to develop a number of major components and is interested in its overall productivity, it can add together the work-hours needed to develop each component. Similarly, it can add together the sizes of the major components it has developed if (but only if) those sizes meet the principles laid down above.

The reason that sizes of major components from different layers of a standard layered architecture, measured from the same Measurement Viewpoint, may be added together is that such an architecture has a coherently defined set of Users (in the COSMIC-FFP sense). Each layer is a user of its adjacent layers and each major component in a layer is a user of other peer major components in the same layer. The requirements of the architecture impose that the FUR of the major components must exchange messages. It is therefore only logical and reasonable that the sizes of the components may be added together.

However, in contrast, the size of a major component measured from the Developer Measurement Viewpoint may *not* be obtained by adding up the sizes of its component re-usable objects. The reason for this is best described by an analogy.

The surface area of a wall cannot be obtained by adding up the surface areas of the bricks used to build the wall. For obvious reasons, some parts of the brick surfaces are used for 'inter-brick interfaces' and do not contribute to the wall's surface area. Similarly, the size of a major software component cannot be obtained by adding up the sizes of its 'basic building blocks'. The total from adding up the latter sizes would include all the data movements between 'basic building blocks'. These 'inter-block' data movements result from decisions of the software designer, not from the FUR of the major component. Therefore they would not be identified when measuring the size of the major component, seen from the measurement viewpoint of its Developer.

This is a clear example where the measurement viewpoint (or level of abstraction) of the developer of the major component differs from that of the developer of the re-usable components.

5

COSMIC-FFP MEASUREMENT REPORTING

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

5.1 LABELLING

When reporting a COSMIC-FFP functional size it should be labeled according to the following convention, in accordance with the ISO/IEC 14143-1: 1998 standard.

RULE[34] – COSMIC-FFP Measurement Labelling

A COSMIC-FFP measurement result is noted as "***x*** Cfsu (v. ***y***) ", where:

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

Example: a result obtained using the rules of this Measurement Manual is noted as '**x** Cfsu (COSMIC-FFP v2.2)'

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-FFP version, use the above labelling convention, but note the use of the approximation method elsewhere – see section 5.2.

When local extensions are used, as defined in section 4.1.5 above, the measurement result must be reported as defined below.

RULE[35] – COSMIC-FFP Local Extensions Labelling

A COSMIC-FFP measurement result using local extensions is noted as:

" ***x*** Cfsu (v. ***y***) + z Local FSU", where:

- "**x**" represents the numerical value obtained by aggregating all individual measurement results according to the standard COSMIC-FFP method, version v.y,
- "v.**y**" represents the identification of the standard version of the COSMIC-FFP 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-FFP method.

[34] From: Morris P., Desharnais J.-M., "*Measuring ALL the software, not just what the Business uses*", Proceedings of the IFPUG Conference, Orlando, 1998.

[35] From: Morris P., Desharnais J.-M., "*Measuring ALL the software, not just what the Business uses*", Proceedings of the IFPUG Conference, Orlando, 1998.

5.2 ARCHIVING COSMIC-FFP MEASUREMENT RESULTS

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

CONVENTION – COSMIC-FFP measurement detail

A distinct record of each measured layer or of each component within a layer is kept. This record contains at least the following information:

a) Identification of the measured software component (name, version ID or configuration ID)
b) a description of the layer's function,
c) a statement of the Purpose of the measurement
d) the Scope of the measurement, using the generic categories as in 2.7 above
e) the Measurement Viewpoint
f) 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)
g) the target or believed accuracy of the measurement
h) indications whether the standard COSMIC-FFP measurement method was used, and/or a local approximation to the standard method, and/or whether local extensions were used (see 4.1.5). Use the labeling conventions of sections 5.1 or 5.2
i) 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)
j) 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 4.3 b))
k) a description of the architecture of layers in which the measurement is made, if applicable, seen from the Measurement Viewpoint
l) the number of major components, if applicable, whose sizes have been added together for the total size recorded.
m) A catalogue of the functional processes identified showing for each functional process:
    - the name
    - the number of ENTRY data movements,
    - the number of EXIT data movements,
    - the number of READ data movements,
    - the number of WRITE data movements.

6

COSMIC-FFP CONVERTIBILITY

Users of '1st Generation' functional sizing methods may wish to understand if their data can be converted to size measures according to the COSMIC-FFP measurement method. This chapter sets out some considerations on converting from three functional sizing methods (IFPUG, MkII and FFP V1.0) to the COSMIC-FFP method.

6.1 Convertibility: Practice and Theory

It would be ideal if sizes measured with other methods could be converted to COSMIC-FFP sizes by mathematical formulae, but there are practical and theoretical reasons why this may not be easy, and why it is not recommended for certain measurements.

The practical reason is that only when a lot of functional sizes have been measured on the '1st Generation' method and on the '2nd Generation' COSMIC-FFP method will there be enough data to develop statistically-based conversion formulae. Such repeat measurements require a lot of effort and few organizations have done so to date (specific results are mentioned below).

The theoretical reason is that if there is no definite conceptual mapping between the Base Functional Components (or BFC's) of one method and of the other, then no *exact* mathematically-based conversion formula is likely to be possible. For example, sizes measured with the COSMIC–FFP method from, say, the Developer Measurement Viewpoint take into account concepts, e.g. layering, that are not normally captured by functional sizing methods such as IFPUG and MkII FPA. The latter were designed to measure business application software from the 'End User Viewpoint' (though this aspect of measurement was not explicitly recognized when the measurement methods were originally designed).

Thus, suppose we have a software development project involving a new business application and also changes to multiple layers of the infrastructure software. Measuring software size with the IFPUG and MkII FP methods may well give a result similar to that using the COSMIC-FFP method according to the End User Measurement Viewpoint. But if the goal is to measure the size of all components affected by the project according to the COSMIC-FFP method from the Developer Measurement Viewpoint, it is extremely unlikely that we could obtain a result by converting from the size measurements according to either the IFPUG or the MkII FP methods. The latter would normally consider only the business application within their scope, and ignore the size of changes to the other layers. If contributions to size are missing, there is no proper basis to work from in any conversion process.

6.2 A Process for Converting to COSMIC-FFP Size Measurements

Subject to this limitation (which does not apply to measurements made with FFP V1.0), if an organisation has an existing base of measurements with an '1st Generation' method and wishes to adopt the COSMIC-FFP method, then the following process should be used to convert the existing measurements.

- Re-measure a sample of some FUR with the COSMIC-FFP method from the same Measurement Viewpoint
- Develop local statistically-based 'average conversion' formulae
- Apply these formulae derived from the sample to convert the bulk of the measurements to the new scale

- Examine the components of the original measurements (before conversion) for any reasons as to why the result converted with the 'average conversion' formulae should be inaccurate.

The last point is important. Statistically-based 'average conversion' formulae may be 'on average' accurate enough, but not in all cases. We need therefore to understand the factors that may, for a given 1st generation method, cause measurements for certain types of software to be inaccurately converted with this statistical approach. (We leave aside the important consideration of whether the sample used to derive the conversion formulae is statistically representative of the full set of measurements to be converted.)

In the following description of factors that drive convertibility, we use expressions such as 'relatively high proportion', or ' large numbers of', etc. These expressions are relative to 'average' or 'typical' software, for the parameter concerned, for the software of the organisation carrying out the conversion.

6.3 IFPUG Function Points Version 4.1

The IFPUG method takes into account in its functional size measure three types of 'elementary process' BFC's, which are roughly equivalent to the functional processes of COSMIC-FFP, and two types of 'file' BFC's. The functional size is then multiplied by a 'Value Adjustment Factor' to give an overall size. The COSMIC-FFP method does not have such a 'VAF adjustment', so conversion to a COSMIC-FFP size is only possible from an IFPUG 'Unadjusted Function Point size'. The IFPUG VAF is therefore ignored.

Figure 6.3.1 below shows 14 measurements according to the IFPUG v4.1 and COSMIC-FFP v2.2 methods, from two sources. The first source is that of Fetcke (1999) and the second is a major European financial services organisation. In both cases the measurements were made from the End-user viewpoint on business application sub-systems, in Fetcke's case from one warehousing system, and in the other case from ten financial/administrative systems.

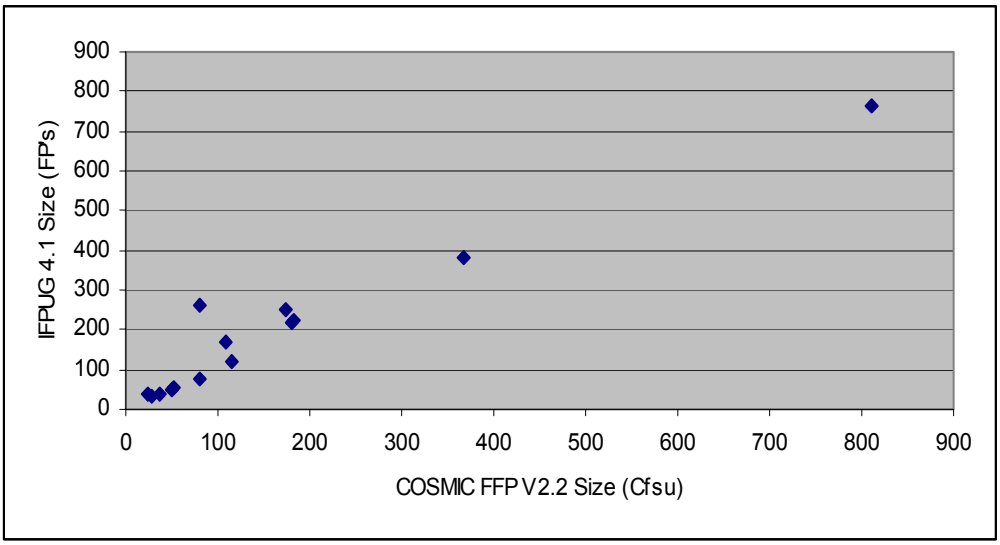


Figure 6.3.1 – IFPUG v4.1 versus COSMIC-FFP v2.2 sizes for 14 subsystems from 3 systems

In most cases, the sizes from the two methods are very similar, but there are a few significant outliers.

- Two main factors may give rise to divergences between IFPUG and COSMIC-FFP sizes. If the software being measured has a high proportion of files, even though each is not much referenced by the processes, these will tend to result in a higher size on the IFPUG scale than on the COSMIC-FFP scale

- The IFPUG size scale allocates function points within limited ranges to each component. For example, an External Input can have a size in the range 3 to 6 function points. In the COSMIC-FFP method, there is no upper limit to the size of a functional process. Examples of real functional processes have

been observed with over 100 data movements, i.e. size greater than 100 Cfsu. For such cases applying an 'average conversion' formula to an IFPUG size could be grossly misleading.

The first factor cited above is the main cause of the divergences from the straight-line correlation in Fig. 6.3 above. Those responsible for the measurements at the financial services organisation considered that the COSMIC-FFP sizes better reflected the relative functionality of its sub-systems than the IFPUG sizes.

6.4 MarkII FP Method Version 1.3.1

The MkII FP method regards FUR as a set of 'logical transactions' which are identical in intent to the functional processes of COSMIC-FFP. Each logical transaction is broken down into input, processing and output components.

The size of a processing component of a logical transaction in the MkII FP method is defined to be proportional to the number of entity-types referenced during the processing. Since an entity-reference is generally equivalent to a Read or Write, the contribution to size of the processing component should be roughly equivalent on both scales.

However, the size of the input and output components of a MkII FP logical transaction is defined to be proportional to the number of data element types (or DET's) on the components respectively. Hence software with an exceptionally low proportion of DET's per logical transaction, or with an exceptionally high proportion will, if its size is converted by an 'average conversion' formula, be under-sized or over-sized respectively.

The smallest size of a MkII FP logical transaction is 2.5 MkII function points and there is no upper limit to its size (cf 2 Cfsu and no upper limit on the COSMIC-FFP method). In general, therefore, one would expect a good, average linear correlation of MkII FP and COSMIC-FFP sizes, with fluctuations about the average caused by the relative numbers of DET's on the inputs and outputs of the logical transactions.

6.5 Full Function Points Version 1.0

The FFP method comprises two parts. The first part uses the standard IFPUG concept to size business applications. The second part is an extension of six BFC's, conceptually similar to the IFPUG BFC's, that can be used to size real-time and multi-layer, infrastructure software. The FFP method can therefore be used to size software FUR from either the End User or from the Developer Measurement Viewpoints (although this distinction, as noted in 6.1 above, was not made when the method was originally designed).

Given this structure, conversion of measurements from the End-User Measurement Viewpoint on business applications using the FFP method to the COSMIC-FFP scale involve the same factors as described in 6.1 and 6.2 above. Results of measurements (Abran et al, 1998) using the FFP V1.0 method on real-time software, effectively from the Developer Measurement Viewpoint, versus attempts to measure with the IFPUG method confirm the observations in 6.1 on the difficulties of converting such measurements.

6.6 Relative Sizes on the Four Functional Size Scales

Note that all functional size scales are arbitrary in absolute terms, so whether a set of sizes measured on one scale happens to be bigger or smaller, on average, than when measured on the other scale is immaterial. Existing Functional Size Measurement methods have not in general been designed to give similar sizes, but it happens that the four methods of interest here (IFPUG, MkII FPA, FFP v1.0 and COSMIC-FFP v2.2) all give roughly similar sizes on average. The only important questions when considering convertibility are (a) do sets of measurements on any two size scales happen to correlate in any statistically significant sense, and if so, how?, and (b) what causes an individual pair of measurements to depart significantly from a best-fit correlation curve?

7

EARLY COSMIC-FFP

As discussed in section 2.6, it may be necessary in practice to determine a COSMIC-FFP size early in a project life-cycle before all detailed information has become available to produce a size according to the detailed rules given in this Measurement Manual.

In these circumstances we can use a locally-calibrated approximate version of the COSMIC-FFP method to obtain the early size estimate. Any approximate COSMIC-FFP method relies on finding a concept at a higher level of abstraction than the Data Movement, that can be assigned a size in Cfsu.

The first higher level concept above the Data Movement is the Functional Process. The simplest process for obtaining an approximate size of a new piece of software is therefore as follows. For the new piece of software:

1. Identify a sample of other pieces of software with similar characteristics to the new piece
2. Identify their functional processes
3. Measure the sizes of the functional processes of these other pieces with COSMIC-FFP
4. Determine the average size, in Cfsu, of the functional processes of these other pieces (e.g. = 8 Cfsu)
5. Identify all the functional processes of the new piece of software (e.g. = 40)
6. Based on the sample the early estimated size of the new piece of software is 8 x 40 = 320 Cfsu.

Such a process can be refined to give a more accurate result if instead, before step 4 above, the functional processes are sorted into categories giving equal contributions to the total size. In a real example for one component of a major real-time avionics system, it was decided to divide the functional processes into four quartiles of equal contribution to size. The average size of the functional processes in each quartile (and the names given to these quartiles) was:

'Small'	3.9 Cfsu
'Medium'	6.9 Cfsu
'Large'	10.5 Cfsu
'Very Large'	23.7 Cfsu

(To interpret these figures, for example, 25% of the total size of the component was accounted for by 'Small' functional processes whose average size was 3.9 Cfsu, another 25% of the total size by 'Medium' functional processes of average size 6.9 Cfsu, etc.)

In step 5 of the above process, the functional processes of the new piece of software are identified and also classified as 'Small', 'Medium', 'Large' or 'Very Large'.

In step 6, the average sizes listed above are then used to multiply the number of functional processes of the new piece of software, in each quartile respectively to get the total early estimated size.

N.B. Anyone wishing to adopt this process is strongly advised to calibrate their average sizes with local data relevant to the piece of software to be sized (and NOT to use the above averages). Much further research and measurement is needed before it will be possible to give 'industry-average' sizes of functional processes for various circumstances.

This process can then be repeated, in principle, to any concept at a higher level of abstraction which can be assigned a size in Cfsu. The difficulty with continuing in this way is that whereas we have defined the concepts of 'data movement' and of 'functional process' precisely, if we are to go to higher levels, we need additional concepts that can be defined precisely, so that their average sizes can be calibrated. Again, we need further research to identify and define such higher level concepts that could be used in a *universal* way.

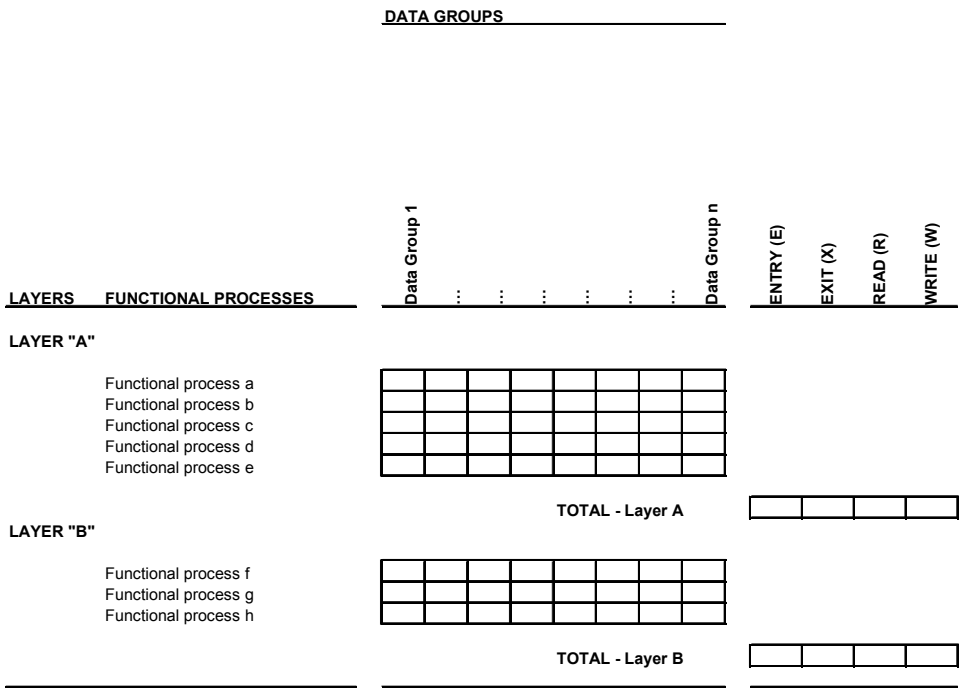
Meanwhile, an organisation wishing to continue the process to higher levels can do so if it has locally well-defined higher-level concepts. Examples might be 'Use Case', or perhaps 'process-group', that everyone in the organisation interprets in a common way. If that is the case, then the calibration process described above can be continued *locally* to the higher level.

Appendices

Appendix A

COSMIC-FFP GENERIC SOFTWARE MODEL

The matrix below can be used as a repository to hold each identified component of the generic software model to be measured. It is designed to facilitate the use of the measurement process.



MAPPING PHASE

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

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 layer, on the "TOTAL" line.

Appendix B

COSMIC-FFP PRINCIPLES IDENTIFICATION

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

ID	NAME	PRINCIPLE DESCRIPTION
P-01	Layer	a) Software in each layer delivers functionality to its 'own' Users (a User may be a human, a physical device or other software, e.g. software in another layer) b) Software in a subordinate layer provides functional services to software in another layer using its services. c) Software in a subordinate layer could perform without assistance from software in the layer using its services. d) Software in one layer might not perform properly if software in a subordinate layer on which it depends is not performing properly. e) Software in one layer does not necessarily use all the functionality supplied by software in a subordinate layer. f) In a hierarchy of layers, software in any one layer can be a subordinate to a higher layer for which it provides services. g) Software in one layer and in a subordinate layer can physically share and exchange data. However, the software in each layer will interpret the data attributes differently and/or group them in different data groups. h) Software that shares data with other software shall not be considered to be in different layers if they identically interpret the data attributes that they share.
P-02	Boundary	By definition, there is a boundary between each identified pair of layers where one layer is the user of another, and the latter is to be measured. Similarly, there is a boundary between any two distinct pieces of software in the same layer if they exchange data in 'peer-to-peer' communications; in this case each piece can be a user of its peer.
P-03	Functional process	a) A functional process is derived from at least one identifiable Functional User Requirement, b) A functional process is performed when an identifiable triggering event occurs, c) A functional process comprises at least two data movements, an entry plus either an exit or a write, d) A functional process belongs to one, and only one, layer e) A functional process terminates when a point of asynchronous timing is reached. A point of asynchronous timing is reached when the final (terminating) data movement in a sequence of data movements is not synchronized with any other data movement.
P-04	Data Group	a) A data group must be materialized within the computer system supporting the software. b) Each identified data group must be unique and distinguishable through its unique collection of data attributes. c) Each data group must be directly related to one Object of interest described in the software's Functional User Requirements. (Attention is drawn to 4.1.3 Principle e) for a Read, and to 4.1.4, principle f) for a Write.)

ID	NAME	PRINCIPLE DESCRIPTION
P-05	Data Attribute	a) A data attribute must represent a valid type of data. b) A data attribute must represent the smallest piece of data referenced by the measured software from the perspective of Functional User Requirements.
P-06	ENTRY	a) The data movement receives data attributes lying outside the software boundary, from the user side. b) The data movement receives data attributes from only one data group, that is data about a single Object of interest. If input from more than one data group is received, identify one ENTRY for each group. c) The data movement does not exit data across the boundary, read or write data. d) Within the scope of the functional process where it is identified, the ENTRY is unique, that is, the processing and data attributes identified are different from those of other ENTRIES included in the same functional process.
P-07	EXIT	a) The data movement sends data attributes outside the boundary, to the user side. b) The data movement sends data attributes belonging to only one data group, that is, data about a single Object of interest. If data belonging to more than one data group are sent outside the software boundary, identify one EXIT for each referenced data group. c) The data movement does not receive data from outside the boundary, or read or write data. d) Within the scope of the functional process where it is identified, the sub-process is unique; that is, the processing and data attributes identified are different from those of other EXITS included in the same functional process.
P-08	READ	a) The data movement retrieves data attributes from a data group in persistent storage. b) The data movement retrieves data attributes belonging to only ONE data group, that is, data about a single Object of interest. Identify one READ for each Object of interest for which data attributes are retrieved in any one functional process (see also the 'De-duplication' Rule which may over-ride this rule). c) The data movement does not receive or exit data across the boundary or write data. d) Within the scope of the functional process where it is identified, the data movement is unique, that is, the processing and data attributes identified are different from those of any other READ included in the same functional process. e) During a functional process, the Read (or a Write) of a data group can only be performed on the data describing an Object of Interest to the User. Constants or variables which are internal to the functional process, or intermediate results in a calculation, or data stored by a functional process resulting only from the implementation, rather than from the FUR, are not data groups and are not taken into account in the functional size.
P-09	WRITE	a) The data movement moves data attributes to a data group on the persistent storage side of the software. b) The data movement moves the values of data attributes belonging to only ONE data group, that is data about a single Object of interest. Identify one WRITE for each Object of interest for which data attributes are referenced in any one functional process (see also the 'De-duplication' Rule which may over-ride this Rule). c) The data movement does not receive or exit data across the boundary, or

ID	NAME	PRINCIPLE DESCRIPTION
		read data. d) Within the scope of the functional process where it is identified, the data movement is unique, that is, the processing and data attributes identified are different from those of any other WRITE included in the same functional process. e) A requirement to delete a data group from persistent storage is measured as a single Write data movement. f) During a functional process, the step of storing a data group that does not persist when the functional process is complete is not a Write; examples are updating variables which are internal to the functional process or producing intermediate results in a calculation.
P-10	Aggregation of measurement results	a) For each functional process, the functional sizes of individual data movements are aggregated into a single functional size value by arithmetically adding them together. b) $Size_{Cfsu}$ (functional $process_i$) = $\sum$ size($entries_i$) + $\sum$ size($exits_i$) + $\sum$ size($reads_i$)+ $\sum$ size($writes_i$) c) For any functional process, the functional size of changes to the Functional User Requirements is aggregated from the sizes of the corresponding modified data movements according to the following formula. $Size_{Cfsu}$ (Change(functional $process_i$)) = $\sum$ size(added data movements) + $\sum$ size(modified data movements) + $\sum$ size(deleted data movements) d) The size of each piece of software to be measured within a layer shall be obtained by aggregating the size of the new and any changed functional processes within the identified FUR for each piece e) Sizes of layers or of pieces of software within layers may be added together only if measured from the same Measurement Viewpoint. f) Furthermore, sizes of pieces of software within any one layer or from different layers may be added together only if it makes sense to do so, for the Purpose of the measurement. (For example, if various major components are developed using different technologies, by different project sub-teams, then there may be no practical value in adding their sizes together.)

Appendix C

COSMIC-FFP RULES IDENTIFICATION

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

ID	NAME	RULE DESCRIPTION
R-01	Layer	a) Functional service software packages such as database management systems, operating systems or device drivers, are generally considered as distinct layers. b) If software is conceived using an established architectural paradigm of layers as meant here, then use that paradigm to identify the layers. c) The software application level is generally considered to occupy the highest layer level. d) When in doubt, use the concepts of coupling and cohesion (see Appendix D) to distinguish between interacting layers.
R-02	Boundary	a) Identify triggering events, then identify the functional processes enabled by those events. The boundary lies between the triggering events and those functional processes. b) For real-time or technical software, use the concept of layers to assist in the identification of the boundary (section 3.1).
R-03	Functional process	a) By definition, a triggering event-type gives rise to a triggering Entry-type, that is, the movement of a data group-type defined as comprising a certain number of data attribute-types. If it can happen that certain occurrences of the triggering Entry data group contain only sub-sets of the values of all the data attribute-types, then the possible existence of such occurrences does not imply the existence of a different event-type or functional process-type. For instance, if an occurrence of a specific event-type triggers the entry of a data group comprising data attributes A, B and C, and then another occurrence of the same event-type triggers an entry of a data group which has values for attributes A and B only, this is not considered to be a different triggering event-type. It is considered to be the same for the purpose of identifying COSMIC-FFP functional processes. Consequently, only one entry and one functional process are identified, manipulating data attributes A, B and C. b) In the context of real-time software, a functional process is also triggered by an event. It terminates when a point of asynchronous timing is reached. A point of asynchronous timing is equivalent to a self-induced wait state.

ID	NAME	RULE DESCRIPTION
R-04	Data Group	a) APPLICATION TO BUSINESS APPLICATION SOFTWARE. Measurement practice has established that, in business application software, a data group is identified for each 'entity-type' (or 'Third Normal Form' relation) found in the normalized data model of the measured software. These are usually data groups showing indefinite persistence and the software is required to store data about the entity-types concerned. In COSMIC-FFP, we use the term 'Object of interest' instead of 'entity-type' or 'TNF relation' in order to avoid using terms related to specific software engineering methods. Examples: in the domain of management information software, an Object of interest could be 'employee' (physical) or 'order' (conceptual) – the software is required to store data about employees or orders. Furthermore, data groups showing transient persistence (i.e. data groups about transient Objects of interest) are formed whenever there is an ad hoc enquiry which asks for data about some 'thing' about which data is not stored with indefinite persistence, but which can be derived from data stored with indefinite persistence. In such cases the transient Object of interest is the subject of the entry data movement in the ad hoc enquiry (the selection parameters to derive the required data) and of the exit data movement containing the desired attributes of the transient Object of interest. Example: we form an ad hoc enquiry against a personnel database to extract a list of names of all employees aged over 35. This group is a transient Object of interest. The entry is a data group containing the selection parameters. The exit is a data group containing the list of names. b) APPLICATION TO REAL-TIME SOFTWARE. Real-time software measurement practice has established that data groups for this type of software often take the following forms: • 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 cause an interrupt. • A message-switch may receive a message data group as an Entry and route it forward unchanged as an Exit. The attributes of the message data group could be, for example, 'sender, recipient, route_code and message_content', and its Object of interest is 'Message'. • Common data structure, representing Objects of interest that are mentioned in the Functional User Requirements, which are held in volatile memory and accessible to most of the functional processes found in the measured software, • 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, • Files, commonly designated as "flat files", representing Objects of interest mentioned in the Functional User Requirements, which are held in a persistent storage device.

ID	NAME	RULE DESCRIPTION
R-05	Data Group 'De-duplication'	(The term 'de-duplication' means 'the process of eliminating duplicate copies from a list'.) A data movement of a given type (E, X, R or W) moving any one Data Group (that is, data about a single Object of interest), is *in general*, identified only once in any one Functional Process. Examples: 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. For sizing purposes, we identify only one Read. 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, we identify only one Read, because we identify data movement types, not occurrences. However, there may be legitimate exceptions to this rule. Example: In real-time software, we can envisage a Functional Process where there is a *Functional User Requirement* to repeat a Read before the process ceases in order to check that the data have not changed since the first Read in the same process. In this case, if there is additional and/or different data manipulation associated with the second Read following discovery that the data have changed, then the Read should be identified twice in the one functional process.
R-06	Read and Write Data Movements in 'Update' Functional Processes	Most commonly in on-line business application software, a FUR to update persistent data results in *two* separate functional processes. The first is a 'retrieve-before-update' in which the data about the Object(s) of interest to be updated is/are first Read and displayed. This is followed by the 'update' functional process in which the changed data is made persistent by one or more Write data movements. The retrieve-before-update functional process containing only the Read(s) and no Write(s) allows the human user to verify that the correct record(s) is/are being addressed. The subsequent 'update' functional process allows the user to enter the data that should be changed, added or deleted and to complete the updating via the Write(s). However, there are also various circumstances which can lead to a FUR for *one* functional process in which a Read is followed by a Write of the 'same data', without human intervention. By 'same data' we mean either: • the Data Group that is written is identical to the Data Group that was read but its values have been updated, or • the Data Group that is written is for the same Object of interest as the Data Group that was read, but the Data Group that is written differs from the one that was read due, for example, to the addition of data attributes. In such cases a Read and a Write of the 'same data' should be identified in the one functional process.

ID	NAME	RULE DESCRIPTION
R-07	Correspond-ence of Data Movements across Boundaries	In the Developer Measurement Viewpoint, when one item of software is the user of another software item: a) An Entry to one item of software from a peer item in the same layer corresponds to an Exit in the peer item b) An Exit from one item of software to a peer item in the same layer corresponds to an Entry in the peer item c) A Write is a data movement to persistent storage. It does not cross the boundary of the functional process to which it belongs. But if the Write is delegated to software in another 'lower' layer, this will correspond to an Entry across the boundary of the lower layer d) A Read is a data movement from persistent storage. It does not cross the boundary of the functional process to which it belongs. But if the Read is delegated to software in another 'lower' layer, this will correspond to an Entry/Exit pair across the boundary of the lower layer e) In all four of the above cases, include the data movements on each side of the boundary in the size of their respective components
R-08	ENTRY	a) Clock-triggering events are always external to the software being measured. Therefore, an event occurring every 3 seconds is associated with an ENTRY moving one data attribute, for instance. Note that even if such a triggering event is generated periodically not by hardware, but by a software functional process, the latter can be ignored in the measurement since it occurs, by definition, outside of the boundary of the software being measured. b) Unless a specific functional process is necessary, obtaining the time from the system's clock is not considered as an ENTRY. For instance, when a functional process writes a time stamp, no ENTRY is identified for obtaining the system's clock value.
R-09	EXIT	When measuring size from the End User Measurement Viewpoint, by convention all software messages generated without user data (e.g. confirmation and error messages) are considered to be separate occurrences of one message-type. Therefore, a single EXIT is identified to represent all these messages within the scope of the functional process where these messages are identified. For instance, consider functional processes A and B identified within the same layer. "A" can potentially issue 2 distinct confirmation messages and 5 error messages to its users and "B" can potentially issue 8 error messages to its users. In this example, one EXIT would be identified within functional process "A" (handling 5+2=7 messages) and a separate EXIT would be identified within functional process "B" (handling 8 messages).
R-10	Aggregation of measure-ment results	When using the measurement matrix found in Appendix A, aggregation can be performed by adding the number of data movements found in each row to obtain the size of the functional process and then by adding the sizes of the functional processes to obtain the size of the layer.
R-11	COSMIC-FFP Measure-ment Labelling	A COSMIC-FFP measurement result is noted as "***x*** Cfsu (v. ***y***) ", where: • "**x**" represents the numerical value of the functional size, • "v.**y**" represents the identification of the standard version of the COSMIC-FFP method used to obtain the numerical functional size value "x". Example: a result obtained using the rules of this Measurement Manual is noted as '**x** Cfsu (COSMIC-FFP v2.2)' 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-FFP version, use the above labeling convention, but note the use of the approximation method elsewhere – see section 5.2.
R-12	COSMIC-	A COSMIC-FFP measurement result using local extensions is noted as:

ID	NAME	RULE DESCRIPTION
	FFP local extensions labelling	" ***x*** Cfsu (v. ***y***) + z Local FSU", where: • "**x**" represents the numerical value obtained by aggregating all individual measurement results according to the standard COSMIC-FFP method, version v.y, • "v.**y**" represents the identification of the standard version of the COSMIC-FFP method used to obtain the numerical functional size value "x". • "z" represents the numerical value obtained by aggregating all individual measurement

Appendix D

FURTHER INFORMATION ON SOFTWARE LAYERS

The concept of layers evolved from difficulties encountered by practitioners[36] in applying the "boundary rules" found in the IFPUG Counting Practices Manual (up to v. 4.0). From traditional Functional Size Measurement practices, the software boundary is defined as the border between the software being measured and the users. Furthermore, the definition states that boundary positioning is dependent on the user's external view of the application and independent of technical and/or implementation considerations.

Some questions arise from this definition:

- **Who is the user ?**

  The COSMIC-FFP measurement method defines users as "human beings, software or engineered devices which interact with the measured software".

- **How can we determine what the user's view is ?**

  The user's view, as defined above, implies many levels of abstraction. At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment. At lower levels of abstraction, a more procedural orientation is used. Each step in the software engineering process is a refinement in the level of abstraction of the software solution (Pressman97).

- **When do we consider a function as technical ?**

  Software that is usually included within the scope of an organization can be categorized based on the types of service provided, as illustrated in Figure D.1 below.

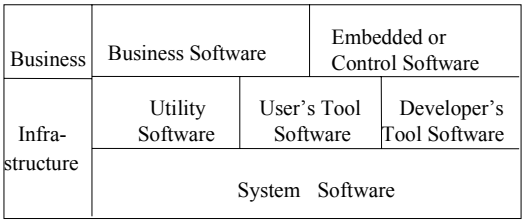


Figure D.1 – Categories of software according to the service provided

The types of service provided are further defined below.

EXAMPLE – Functionality provided by different types of software

Business Software
This type of software delivers functionality which supports the organization's core business. The users are primarily humans , however a small proportion of the functionality may also be delivered to, or triggered by, other business applications. This type of software is typically business or commercial (MIS) software and would include payroll applications, accounts receivable or fleet management systems.

Embedded or Control Software
This type of software also delivers functionality which supports the organization's core business

[36] See for instance: "Function Point Counting Practices for Highly Constrained Systems", Release 1.0, March 1993 by the European Function Point Users Group or "Comments on ISO/IEC 14143-5", release V1b by Grant Rule.

or products. The users are primarily other software applications embedded in equipment. This type of software typically operates under strict timing conditions and is often referred to as real-time software. Examples would include equipment monitoring systems and telephone switching systems.

Utility Software
This type of software delivers functionality which provides the infrastructure to support business software. The users are primarily the business software itself, which initiates the operation of the utilities, but may include the developers or business administrators as the administrative users. Examples would include backup utilities (to ensure the data reliability of the business application) or archiving utilities (to optimize the performance of the business application). Other examples are installation and conversion software.

User's Tool Software
This type of software delivers tooling functionality used by administrative users to create the functionality delivered by business software. The users are primarily the business software itself, which uses functionality delivered by the tools to enable them to deliver functionality to the business. Administrative human users of these tools may be either from the business or from IT. Examples would include report generators, spreadsheets and word processors.

Developer's Tool Software
This type of software delivers tooling functionality used by developers to create the functionality delivered by business software. The users are primarily other applications, which are either generated by, or used as input to, tool operation. Human users may also include IT developers as administrative users. Examples would include code generators, testing software, new product generators, etc.

System Software
This type of software enables all the other types of software to operate and deliver their own functionality. The users are primarily other applications with a limited interface to human IT operational staff. Examples would include operating systems, printer drivers, protocol converters and presentation software.

This descriptive approach is convenient for distinguishing different types of functionality, but there are some grey zones when it is used to measure software functional size. The concept of layers is proposed as a way to formalize the different types of functionality for the purpose of measuring the functional size of software.

At a given service level, software delivers functionality to a specific category of user. When items in a set of software deliver functionality hierarchically relative to one another, they are identified as belonging to different layers. Each layer is thus the 'user' for the layer below it in this hierarchy. Software layers represent different levels of abstraction. At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment. At lower levels of abstraction, a more procedural orientation is offered. A higher-level layer is then at a higher level of abstraction than a lower layer.

In Figure D.2, below, the top layer (L_1) is the layer delivering functionality to the software end-users. Software within any layer may also deliver functionality to peer systems within the same layer Comment: I spent all of yesterday examining a massive distributed system which is primarily all about peer-to-peer communications between separate processors.

- Layer 1 (L_1) delivers functionality directly to end-users. End-users are either humans in the case of business applications, or equipment in the case of embedded (real-time) software.
- Layer 2 (L_2) groups together infrastructure software delivering functionality to support its users: the software in L_1.
- Layer n groups together infrastructure software delivering functionality to support its users: the software in L_{n-1}.

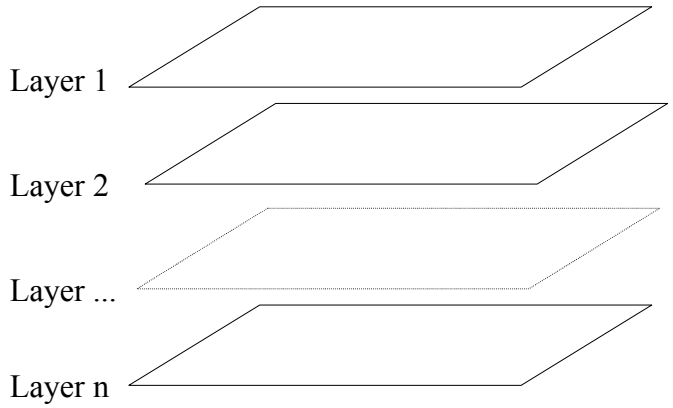


Figure D.2 – Generic representation of layers within a software product

- **How can we make a distinction between different layers ?**

The concepts of software architecture are used in Software Engineering to understand the overall structure of software and the ways in which that structure provides conceptual integrity to software (Pressman). A software architect organizes a software solution to obtain the best set of modules. Effective modularity can be achieved by defining a set of independent modules which only communicate with one another the information that is necessary to achieve a required function (information-hiding principle).

The principle of information-hiding suggests that modules be characterized by design decisions that hide each one from all others. In other words, modules should be specified and designed so that information (procedures and data) contained within a module is inaccessible to other modules that have no need for such information (Pressman).

The program structure should be partitioned both horizontally and vertically. The simplest approach to horizontal partitioning defines three partitions – input, data transformation and output. Vertical partitioning suggests that control (decision-making) and work should be distributed top-down in the program architecture (Pressman).

COSMIC-FFP already provides rules to measure horizontal partitioning (measure of a process in terms of input, data transformation and output). Vertical partitioning suggests that a process can access different software levels or layers. From one layer to another, pieces of software are independent processes, inaccessible to other processes which have no need for such information.

The concept of functional independence is the direct outgrowth of modularity and the concept of information-hiding. Independence is measured using two qualitative criteria: cohesion and coupling. Cohesion is a measure of the relative functional strength of a module. Coupling is a measure of the relative interdependence among software modules.

Appendix E

COSMIC-FFP SUB-RELEASES

This appendix contains a summary of the principal changes made in deriving this version 2.2 from version 2.1 of the Measurement Manual. Its purpose is to enable the reader to trace the changes that have been made and to understand their justification. (For the changes made in deriving version 2.1 from version 2.0, please see version 2.1 of the Measurement Manual, Appendix E.)

The reasons for the most important of the changes made in producing this version 2.2 of the Measurement Manual are discussed in the paper by Abran et al given at the IWSM meeting in October 2002.

Reference in version 2.1 and 2.2	Nature of change	Comment
GENERAL		All changes made in this version 2.2 of the Measurement Manual ('MM') have been made so as a) to bring the MM into line with the International Standard ISO/IEC 19761 for COSMIC-FFP, and b) to clarify and improve the explanations in various places. As regards Principles and Rules, the following changes have been made • one new principle has been introduced, namely concerning the need for the 'measurement viewpoint'; this results in the need for additional conditions for the aggregation of functional sizes of different pieces of software • an addition to an existing principle requiring the aggregation of data movements at the functional process level before aggregating to the layer level • new rules concerning the 'de-duplication' of data movements and the identification of Reads and Writes of the 'same data' in a functional process, and the correspondence of data movements across boundaries Otherwise, no changes have been made to any of the existing Principles and Rules of the method. The chapter and section numbering of this version 2.2 is almost identical to that of version 2.1. The exceptions are noted below. In previous versions of the MM, some terms have not been used consistently. In this version 2.2 the following changes have been made throughout the MM to ensure consistency. • 'COSMIC-FFP context model' and 'COSMIC-FFP generic software model' are used consistently. ('COSMIC-FFP software model' has been eliminated.) • In consequence of introducing the term 'measurement viewpoint', phrases such as 'from the user perspective' have been eliminated where there is a risk of confusion • 'Sub-process' has been changed to 'data movement', wherever

		appropriate • 'Object of interest is used consistently instead of 'object' • 'persistent storage' is used consistently; 'permanent storage' is eliminated • 'Boundary is used consistently, replacing 'storage boundary', 'I/O boundary', software boundary', notably in several Figures • 'Business application' is used consistently, replacing 'MIS software' and such-like expressions. The term 'application' by itself is eliminated • The term 'client' is eliminated from the description of layers, in order to remove possible confusion with 'client/server' concepts. 'Client/server' is used only where that type of relationship is explicitly intended.
Acknowledgments Version Control Foreword	Editorial	Modified for the issuance of Version 2.2. Foreword re-written to explain the changes made in this MM V2.2 to bring it into line with the International Standard ISO/IEC 19761 for the COSMIC-FFP method
Glossary	Technical	Updated to be consistent with terms and definitions in ISO/IEC 19761. Notably the following terms and/or definitions have changed or added: • Abstraction (added) • Boundary • Data movement • Developer viewpoint (added) • End User viewpoint (added) • Entry • Exit • Functional Process • Functional Size (added) • Measurement Method (added) • Measurement Procedure (added) • Measurement Process (added) • Measurement Viewpoint (added) • Persistent Storage (added) • Purpose of a Measurement (added) • Read • Triggering Event • User • Viewpoint (added) • Write The definitions of some other terms have had minor editorial changes. All these changes have been carried forward into the later chapters where they are used
Introduction	Editorial	Brought up to date as regards the COSMIC initiative
2. Overview of the COSMIC-FFP Measurement method	Editorial Editorial Editorial Editorial Editorial	• 'Applicability of the COSMIC-FFP method' made into a separate section 2.1. Consequently all subsequent sections re-numbered • Section 2.2 updated to make clear the distinction between the FUR as found by the measurer in the artifacts of the software to be measured, and the FUR as expressed in the COSMIC-FFP generic software model • Fig. 2.2.1 updated to give greater emphasis to considering the 'measurement context' before starting a measurement • Figs. 2.3.1 and 2.3.2 updated to be consistent with Fig. 2.2.1 and section 2.2 • Fig. 2.4.1.1 changed to make clear that persistent 'storage' is within the boundary and to show more clearly that 'engineered devices' can be direct users of the software

	Editorial Technical Editorial Technical Editorial	• Figs. 2.4.1.2 and 2.4.1.3 modified to show more clearly the boundaries. Fig. 2.4.1.3 modified to show a more typical 'multi-tier' application layer architecture • A sub-section 'On Measurement Scaling' of section 2.5 in v2.1 has been separated into a new section 2.6 'Sizing early in a project life-cycle: Measurement Scaling', and considerably expanded with an example. • Fig. 2.6.1 updated to make clearer the relationship between a local approximation method for early project life-cycle sizing, and the full COSMIC-FFP measurement method • Section 2.7 'Functional Size Measurement context: Purpose, Scope and Measurement Viewpoint' (previously in v2.1 part of section 2.5) is now expanded considerably to elaborate on the meaning and importance of deciding the Purpose and Scope of a measurement and to introduce the concept of the Measurement Viewpoint, which should be determined before starting a measurement. The 'End User' and 'Developer' Measurement Viewpoints are defined • The text of section 2.6 'Other aspects of size' is moved to section 2.5 on 'The COSMIC-FFP measurement phase
3. The Mapping Phase	 Editorial Editorial Editorial Technical Technical Technical Editorial Editorial	Section 3.2 'Identifying software boundaries' • Fig. 3.1 updated to give greater emphasis to considering the Purpose, Scope and Measurement Viewpoint before starting a measurement • Rule 3.2 (b) of v2.1 for Boundary was confusing and has been deleted • Figs. 3.2.1 and 3.2.2 of v2.1 illustrating Case 1 are combined (as Fig. 3.2.1) to present one model of the users, boundary, data movement types and persistent storage of application software from the end-user measurement viewpoint • Figs. 3.2.2 a) and b) replaces the previous Fig. 3.2.3 for Case 2 which illustrates the analysis in multi-layer software from the developer measurement viewpoint. The associated text is changed to explain correctly the relationship between a Read in an application layer and an 'Entry/Exit pair' in a device driver layer. These Figures are shown in the conventions of message sequence diagrams • A new Case 3 is introduced to show how the solution to Case 2 changes when data has to be read via client/server software. The solution is in Fig. 3.2.3 Section 3.3 'Identifying functional processes' • The definition of 'Functional Process now includes a definition of 'Actor' that applies in this context. Section 3.4 'Identifying Data Groups' • The rules for data groups b) for real-time software include more examples and eliminate possible confusion between a 'message data group', and its Object of interest, a 'message' • In 'Examples of data group identification', a third example is added for an ad hoc enquiry with multi-Exit output
4. The Measurement Phase	4.1 'Identifying Data Movements' Technical Technical	• A new rule is introduced (the 'de-duplication rule') which states that *in general* a data movement of a given type (E, X, W, R) moving any one data group should be identified only once in any one functional process. The rule gives examples of possible exceptions to the general case • A new rule is introduced concerning the reading and writing of the

		'same data' in functional processes that must update persistent data
	Technical	• A new rule is introduced that defines the correspondence of data movements across boundaries
	Editorial	• The explanation of the data manipulation associated with a Read was previously misleading and is improved
	Technical	• Additions to the explanation of the data manipulation associated with a Read and with an Entry to make clear that they are considered to include any 'request to' (read or enter) functionality – from ISO/IEC 19761
	Technical	• Additions to the principles for a Read and for a Write to make clear that they should not be identified for the reading or creation of intermediate results, reading of data pre-stored in the software, etc. – from ISO/IEC 19761
	Technical	• Addition to the principles for Writes that a requirement to delete a data group from persistent storage is identified as a Write
		4.2 'Applying the Measurement Function'
	Technical	• Additions of new principles for aggregation of functional sizes to make clear that they can only be added if measured from the same Measurement Viewpoint. Examples added to clarify these principles
5. COSMIC-FFP Measurement Reporting	Editorial	• The 'Conventions' for labelling COSMIC-FFP measurements and measurements made with a local extension to the method are re-designated as 'Rules', since these are mandatory requirements of ISO standards
	Technical	• A Note is added to the rule for labeling COSMIC-FFP measurements when made using local approximations to the standard method
	Technical	• Several additions made to the conventions for reporting of COSMIC-FFP measurement results to ensure consistent interpretation, notably in the area of Purpose, Scope and Measurement Viewpoint
6. COSMIC-FFP Convertibility	Technical	A new chapter describes the factors to be considered in converting measurements obtained using three '1st generation' Functional Size Measurement methods (IFPUG, MkII FPA and FFP V1.0) to sizes according to the COSMIC-FFP method
7. Early COSMIC-FFP	Technical	A new chapter describes the factors to be considered in determining an estimate of functional size according to the COSMIC-FFP method early in the life of a project when the FUR have not yet been defined in detail. A process is described for calibrating a local approximate sizing method
Appendix B Principles	Editorial	Updated in line with this v2.2 of the Measurement Manual
Appendix C Rules	Editorial	Updated in line with this v2.2 of the Measurement Manual. The addition of three new rules results in re-numbering of some rules from v2.1
Appendix D Further Information on Software Layers	Editorial	Figure references corrected, otherwise unchanged
Appendix E	Editorial	Updated to summarize the changes from v2.1 to v2.2
Appendix F	Editorial	A new procedure is introduced to enable users of this COSMIC-FFP Measurement Manual to provide comments and to submit Change Requests to the COSMIC Measurement Practices Committee
References	Editorial	Several additions, notably of papers describing practical experience of using the COSMIC-FFP method

Appendix F

COSMIC-FFP 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-FFP 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-FFP 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-FFP 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-FFP method

The COSMIC MPC regrets that it is unable to answer questions related to the use or application of the COSMIC-FFP 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.

References

Abran, A., Symons, C., Oligny, S., 'An Overview of COSMIC-FFP Field Trial Results', 12th *European Software Control and Metrics Conference – ESCOM 2001*, April 2-4, London (England), 2001.

Abran, A., Symons, C., Desharnais, J.M., Fagg, P., Morris, P., Oligny, S., Onvlee, J., Meli, R., Nevalainen, R., Rule, G. and St-Pierre, D., "COSMIC-FFP Field Trials Aims, Progress and Interim Findings," *11th European Software Control and Metric Conference - ESCOM SCOPE 2000*, Munich, Germany, 2000.

Abran, A., Desharnais,J.M., Oligny, S., St-Pierre, D. and Symons, C., "COSMIC-FFP – Measurement Manual Version 2,0 - Field Trials Version" Université du Québec à Montréal, Montréal, October 1999.

Abran, A., Fagg, P., Meli, R., and Symons, C., 'ISO Transposition and Clarifications of the COSMIC-FFP Method of Functional Sizing', International Workshop on Software Metrics, Magdeburg, October 2002.

Abran, A. "FFP Release 2,0: An Implementation of COSMIC Functional Size Measurement Concepts," *Federation of European Software Measurement Associations - FESMA'99*, Amsterdam, 1999.

Abran, A., "Functional Size of Real-Time Software: Challenges & Solution," Tokyo, Japan: *Japanese Function Point Users Group - JFPUG*, 1998.

Abran, A., St-Pierre, D., Maya, M. and Desharnais, J.M., "Full Function Points for Embedded and Real-Time Software," *United Kingdom Software Metrics Association - UKSMA*, London, UK, 1998.

Abran, A., Desharnais, J.M., Maya, M., St-Pierre, D., and Bourque, P., "Design of Functional Size Measurement for Real-Time Software," Université du Québec à Montréal, Technical Report no. 13, November 23, Montréal (Canada),1998.

Abran, A., Ho, V.T., Oligny, S. Fetcke, T. 'Convertibility Study of Functional Size Measurement Methods: COSMIC-FFP and IFPUG methods', obtainable from www.lrgl.uqam.ca

Abran, A., Maya, M., Desharnais, J.M., and St-Pierre, D., "Adapting Function Points to Real-Time Software," *American Programmer*, Vol. 10, no 11, p. 32-43, 1997.

Albrecht A.J., Gaffney J.E. Jr., "*Software function, source lines of code and development effort prediction: a software science validation*", IEEE Transactions on Software Engineering, Vol. SE-9, pp. 639-648, November 1983.

Albrecht, A.J., *AD/M Productivity Measurement and Estimate Validation*, IBM Corporate Information Systems, IBM Corp., Purchase, N.Y., May 1984.

Bevo, V., Lévesque, G., and Abran, A., "Application de la méthode FFP à partir d'une spécification selon la notation UML: compte rendu des premiers essais d'application et questions," *International Workshop on Software Measurement - IWSM'99*, Lac Supérieur, Canada, 1999.

Bootsma, F., 'Applying Full Function Points to Drive Strategic Business Improvement within the Real-time Software Environment', *IFPUG Annual Conference*, New Orleans, Oct. 18-22, 1999.

Büren, G. , Koll, I., 'First Experiences with the Introduction of an Effort Estimation Process', *ICSSEQ 99*, Dec. 8-10, Paris, France, 1999.

Büren, G., Foltin, E., Weber, E. and Schweikl, S., 'Applicability of FFP at Siemens AT', *International Workshop on Software Measurement – IWSM 99*, Lac Supérieur Québec, Canada, 1999.

Cooling, J. E., *Software Design for Real-Time Systems,* Chapman and Hall, 1991.

Desharnais, J.M., Abran, A., Oligny, S., St-Pierre, D. and Symons, C., "COSMIC-FFP – Manuel de mesures Version 2.0 - Essais sur le terrain", Université du Québec à Montréal, Montréal, October 2000.

Desharnais, J.M., St-Pierre, D., Oligny, S. and Abran, A., "Software Layers and Measurement," *International Workshop on Software Measurement – IWSM,* Lac Supérieur, Québec, 1999.

Desharnais, J.M., Abran, A. and St-Pierre, D., "Functional Size of Real-Time Software," *11th International Conference - Software Engineering and its Applications*, Paris, France, 1998.

Fetcke, T. , The Warehouse Software Portfolio: a case study in functional size measurement'. Report No. 1999-20, obtainable from www.lrgl.uqam.ca

Foltin, E. 'Die Full-Function-Point-Methode', *Ein Ansatz zur Bestimmung des funktionalen Umfangs von Real-Time-Software*, Austria, 2000.

Ho, V.T., Abran, A. and Oligny, S., "Using COSMIC-FFP to Quantify Functional Reuse in Software Development," *11th European Software Control and Metric Conference - ESCOM SCOPE 2000,*. Munich, Germany, 2000.

Ho, V.T., Abran, A. and Fournier, B., "On the Reproducibility of a Functional Size Measurement Method and its Use for Determining the Range of Errors in the Measurement of a Particular Software Portfolio," Université du Québec à Montréal - UQAM, January 27, 2000.

Ho, V.T., Abran, A. and Oligny, S., "Case Study - Rice Cooker, Version 1.0," Université du Québec à Montréal, Montréal, Québec - UQAM, December 1999.

Ho, V.T., Abran, A., Oligny, S., Bevo, B., Ndiaye, I., Lakrib, C., Hoyos, M., Boudaa, A., Dion, D., Mattalah, M. and Dinh, H.P., "Case Study - ISND Loop Back Tester, Version 1.0," Université du Québec à Montréal, Montréal - UQAM, December 1999.

Horgan, C., 'Measuring Software Size for Continuous Improvement', ASQ Software Division Conference, 2002

ISO, International Vocabulary of Basic and General Terms in Metrology", International Organization for Standardization, Switzerland, 2nd edition, 1993, ISBN 92-67-01075-1.

ISO, ISO/IEC 14143-1: 1998 – Software Engineering – Software measurement – Functional size measurement – Part 1 : Definition of concepts.

ISO/IEC 10746-2. 'Information technology -- Open Distributed Processing -- Reference Model: Foundations'

Jenner, M.S., 'COSMIC-FFP 2.0 and UML: Estimation of the Size of a System Specified in UML – Problems of Granularity', FESMA Meeting, Heidelberg, Germany, June 2001

Jenner, M.S., 'Automation of Counting of Functional Size Using COSMIC-FFP in UML', International Workshop on Software Engineering 2002, Magdeburg, Germany, October 2002

Kececi, N., LI, M. and Smidts, C., 'Function Point Analysis: An Application to a Nuclear Reactor Protection System', *Probabilistic Safety Assessment – PSA'99*, August 22-25, Washington, 1999.

Kececi N., Abran A., "An Integrated Measure for Functional Requirements Correctness". IWSM2001, 11th International Workshop on Software Measurement, August 28-29, 2001 Montréal (QC) Canada

Kececi N., Abran A. "Analyzing, Measuring and Assessing Software Quality in a Logic Based Graphical Model" 4th International Conference on Quality and Dependability-QUALITA 2001, 22-23 March 2001 Annecy France.

Kececi N., Halang, W., Abran A. A Semi-formal Method to Verify Correctness of Functional Requirement Specifications of Complex System, DIPES Workshop, World Computer Congress, IFIPS, Montreal, August 2002.

Laplante, P., *Real-Time Systems Design and Analysis: An Engineer's Handbook*, The Institute of Electrical and Electrical Engineers Inc., New York, NY, 1993, 339 pages.

Lam M., Couzens G., Goodman P., Laing B., Rule G., *Function Point Counting Practices for Highly Constrained Systems*, European Function Point Users Group, 1993, 36 pages.

Le Petit Larousse Illustré, 1996 Edition.

Lokan, C. and Abran, A., "Multiple viewpoints in functional size measurement," *International Workshop on Software Measurement – IWSM 99*, Lac Supérieur, Québec, 1999.

Maya, M., Abran, A., Oligny, S., St-Pierre, D. and Desharnais, J.M., "Measuring the Functional Size of Real-Time Software," *9th European Software Control and Metrics Conference and 5th Conference for the European Network of Clubs for Reliability and Safety of Software - ESCOM-ENCRESS-98*, Rome, Italy, 1998.

Meli, R., Abran, A., Ho, V.T. and Oligny, S., "On the Applicability of COSMIC-FFP for Measuring Software Throughout its Life Cycle", *11th European Software Control and Metric Conference - ESCOM SCOPE 2000*, Munich, Germany, 2000.

Molinié, L.c Miranda, M., Abran, A., Desharnais, J.M., Oligny, S.,. St-Pierre, D. and Symons, C., "COSMIC-FFP - Manual de Medición Versión 2.0 – "Versión de pruebas de campo"," Université du Québec à Montréal - UQAM, October 1999.

Morris, P. and Desharnais, J.M. 'Measuring All the Software not Just What the Business Uses', *IFPUG Fall Conference*, Orlando, Florida, Sept. 21-25, 1998.

Nagano, S., Mase, K., Watanabe, Y., Watahiki, T. and Nishiyama, S., 'Validation of Application Results of COSMIC-FFP to Switching Systems', Australian Conference on Software Metrics, November 2001.

Nevalainen, R., "COSMIC - taivaan lahja ohjelmiston koon laskentaan," *Swengineering - Project Special*, Finland, vol. 1, pp. 6-7, 2001.

Oligny S. and Abran, A., "On the Compatibility Between Full Function Points and IFPUG Function Points," *10th European Software Control and Metric Conference - ESCOM SCOPE 99,* Herstmonceux Castle, England, 1999.

Oligny, S., Abran, A. and St-Pierre, D., "Improving Software Functional Size Measurement," *COCOMO and Software Cost Modeling International Forum 14*, Los Angeles, USA, 1999.

Oligny, S., Abran, A., St-Pierre, D. and Desharnais, J.M., "Innovations dans la mesure de la taille fonctionnelle du logiciel," *12th International Conference on Software and Systems Engineering and their Applications - ICSSEA*, Paris, France, 1999.

Oligny, S., Abran, A., Desharnais, J.M., St-Pierre, D. and Symons, C., "COSMIC-FFP Technology Transitioning Data Collection Protocol," Université du Québec à Montréal, Montréal – UQAM, Version 1c, December 1999.

Oligny, S., Desharnais, J.M. and Abran, A., "A Method for Measuring the Functional Size of Embedded Software," *3rd International Conference on Industrial Automation*. Montréal, 1999.

Oligny S. and Abran, A., "Field Testing Full Function Points: Recent Results," presented at *NESMA Autumn Congress 1998*, Amsterdam, Netherlands, 1998.

Oligny S. and Meli, R. "COSMIC-FFP - Aims, Design Principles and Progress", GUFPI – ISMA (Gruppo Utenti Function Point Italia – Italian Software Metrics Association), Rome, November 1999

Pressman, R., *Software Engineering – A practitioner's approach*, 4th Edition, McGraw Hill, 1997.

Rule P.G., *Comments on ISO/IEC 14143-5*, Release V1b.

St-Pierre, D., Maya, M., Abran, A., Desharnais, J.M., and Bourque, P., "Full Function Points: Counting Practices Manual," Université du Québec à Montréal, Montréal, Technical 1997-04, September 1997.

St-Pierre, D., Abran, A., Araki, M. and Desharnais, J.M., "Adapting Function Points to Real-Time Software," *IFPUG 1997 Fall Conference*, Scottsdale, Arizona,1997.

St-Pierre, D., Desharnais, J.M., Abran, A. and Gardner, B., "Definition of When Requirements Should be Used to Count Function Points in a Project," in *The Voice 1996 - A publication of the International Function Point Users Group*, Vol. 1, no 1, 1996, p. 6-7, 22.

Symons, C., Abran, A., Desharnais, J.M., Fagg, P., Goodman, P., Morris, P., Oligny, S., Onvlee, J., Nevalainen, R., Rule, G. and St-Pierre, D., "COSMIC-FFP - Buts, approche conceptuelle et progrès," presented at *ASSEMI*, Paris, France, 1999.

Symons, C., Abran, A., Dekkers, C., Desharnais, M.M., Fagg, P. Morris, P., Onvlee, J., Nevalainen, R., Rule, G. and St Pierre, D., "Introduction and Overview of Principles," *Japanese Function Point User Group - JFPUG*, Tokyo, Japan, 1999.

Toivonen, H., 'Defining Measures for Memory Efficiency of the Software in Mobile Terminals', International Workshop on Software Measurement, Magdeburg, Germany, October 2002