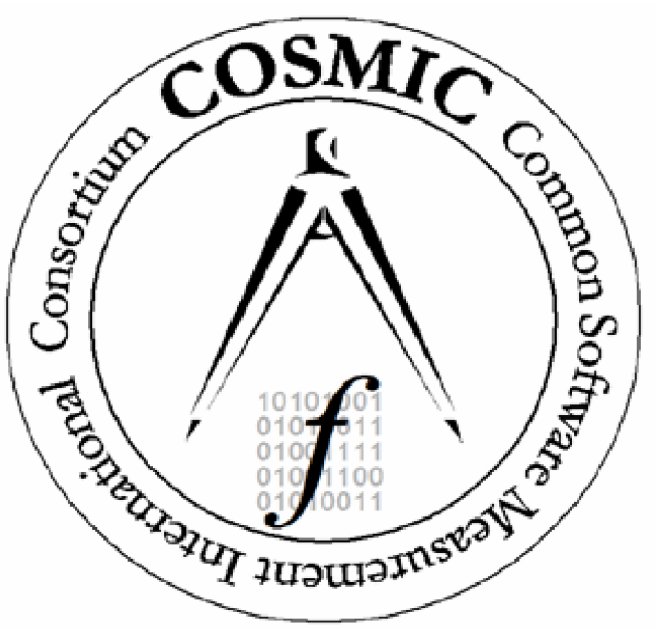

The COSMIC Functional Size Measurement Method

Version 3.0

Method Overview

September 2007

ACKNOWLEDGEMENTS

'COSMIC-FFP' METHOD[1] VERSION 2.0 AUTHORS (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,

Charles Symons, Software Measurement Services, UK

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

* Founding members of the COSMIC Core Team, along with the COSMIC-FFP Method v2.0 authors

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

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

* Editors of version 3.0 of the COSMIC method

VERSION CONTROL

The following table summarizes the changes to this 'Method Overview' document

DATE	REVIEWER(S)	Modifications / Additions
September 2007	COSMIC Measurement Practices Committee	First version for public release. The content is partly based on chapter 2 of the Measurement Manual v2.2, but has been largely extended.

FOREWORD

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

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

Purpose of this document

The purpose of this 'Method Overview' is to give a summary of the COSMIC functional size measurement method, version 3.0. We envisage that this 'Method Overview' document will be of interest to readers who

- need an overview of the method, but who do not need to know all its details
- are new to the idea of measuring functional sizes of software, and who need an introduction to the subject
- are familiar with an existing '1st generation' functional size measurement method (such as the 'IFPUG', 'MkII' or 'NESMA' methods) and who are considering advancing to the COSMIC method.

COSMIC Method Documentation

For a full account of the COSMIC Method documentation, please refer to 'COSMIC Method v3.0: Documentation Overview and Glossary of Terms'. The glossary of this document contains the definition of all terms that are common to all COSMIC documents. This document and all other COSMIC documents mentioned below may be down-loaded free-of-charge from www.gelog.etsmtl.ca/cosmic-ffp

For clarity, and for the intended readers of this document, the other principal documents of interest will be as follows.

- The ISO/IEC 19761 standard, which contains the fundamental normative definitions and rules of the method.
- The 'COSMIC Method version 3.0: Measurement Manual', which provides these rules and definitions, and also aims to provide further explanation and many more examples in order to help measurers to fully understand and to apply the method. This should be the main 'working document' that measurers will need in practice.
- The 'COSMIC Method version 3.0: Advanced and Related Topics', which contains material beyond the basic method such as on approximate size measurement early in a project's life and on convertibility of functional sizes measured with other functional size measurement methods to COSMIC sizes

Other COSMIC documentation available includes Guidelines for the method's application in specific software domains, various Case Studies illustrating the method's use, research papers, benchmark data, etc. Translations of the Measurement Manual into other languages are also available. All these can be found on www.gelog.etsmtl.ca/cosmic-ffp.

More general background information on functional size measurement and its uses, on the advantages of the COSMIC method, on the COSMIC organization and its activities, on suppliers of COSMIC-related services, COSMIC Newsletters, etc., can be found on www.cosmicon.com

The COSMIC Measurement Practices Committee

TABLE OF CONTENTS

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. Hence 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 example, for efficiency analysis or to improve the performance of designs, etc.

On the other hand, functional measures are needed, for example by developers to estimate or measure software size from requirements early in a project's life as the main input to estimating project effort, or to quantify the performance of products or services from a user's or owner's perspective for productivity analysis. Functional measures must be independent of technical development and of implementation decisions. They can then be used to compare the productivity obtained via different techniques and technologies.

Function Point Analysis (FPA)[2] is an example of a functional size measurement method. 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 can successfully capture 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]. In particular, FPA has not been well accepted in the real-time software community.

The 'Full Function Point' method (version 1.0) was proposed in 1997[3] with the aim of extending FPA to capture the functional size of real-time software and of technical and system software. Field tests showed that FFP is also suited to measuring the functional size of MIS software, leading, in such applications, to similar results[4].

In 1998, the FFP group merged their efforts with the work of the COSMIC group[5] which proposed the principles for a second generation of functional size measurement methods. These efforts resulted in the first, publicly-available 'field trials' version 2.0 of the COSMIC-FFP measurement method, published in October 1999.

---

[2] 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. 'FPA' s now known as the 'IFPUG' method.

[3] 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

[4] 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.

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

From version 2.0 onwards, the COSMIC measurement method has also been designed to ensure its full compliance with the ISO/IEC 14143-1: 1998 standard (and subsequently ISO/IEC 14143-1: 2007) as well as with the COSMIC principles.

From version 3.0, the method's name has been simplified from 'COSMIC-FFP' to 'COSMIC'

About the COSMIC initiative

Given the explosive growth and diversity of software contracting and outsourcing, suppliers and customers need more accurate methods of estimating and of measuring performance. These methods must work equally reliably across all types of software. 'First generation' 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 aims to meet these needs of, firstly, 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, secondly, of 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 original aims of the COSMIC project were to develop, to test, to bring to market and to seek acceptance of new software sizing methods to support estimating and performance measurement. These aims have now been achieved and the method is being accepted in a growing number of organizations in the public and private sectors around the world.

After the principles of the COSMIC method were first laid down in 1999, field trials were successfully conducted in 2000/01 with several international companies and academic institutions. Papers describing these trial results and many other research findings are listed on the www.gelog.etsmtl.ca/cosmic-ffp site. The process of developing an International Standard for the COSMIC method was started in 2001. The standard was approved in December 2002 and was published by ISO in early 2003 as ISO/IEC 19761.

COSMIC continues to refine the definition and explanation of the method in light of practical experience, though **it must be emphasized that the Generic Software Model, which is the basis for size measurement, has not changed since it was first published in 1999**. Version 3.0 of the Measurement Manual is the latest step in this process of refinement, which continues whilst always remaining compatible with the ISO/IEC 19761 standard. The designation of 'version 3.0' compared with the previous 'version 2.2' indicates that v3.0 represents an important step forward in the refinement of the method. For a full account of the changes made in progressing from v2.2 to v3.0, see 'The COSMIC Method v3.0: Measurement Manual'

The Common Software Measurement International Consortium (COSMIC) envisages that these additions and refinements will be submitted to ISO for inclusion in ISO/IEC 19761 when it is due for revision in 2007/8.

In 2006, COSMIC introduced the first 'Entry-level' certification examinations for practitioners of the method. Users of the COSMIC method are encouraged to submit performance data on their projects to the database of the International Software Benchmarking Standards Group ('ISBSG'), to enhance the existing benchmark data related measured using the COSMIC method.

For further information about COSMIC, its publications, activities and examinations, please visit www.cosmicon.com, or www.gelog.etsmtl.ca/cosmic-ffp. For further information about the ISBSG, visit www.isbsg.org.

2

OVERVIEW OF THE COSMIC MEASUREMENT METHOD

The COSMIC measurement method defines a standardized measure of software functional size. This chapter presents and discusses:

- the types of software for which the method has been designed to measure functional size (otherwise known as 'the domain of applicability' of the method) in section 2.1
- an overview of the software models used for measurement, in section 2.2. These models introduce all the basic concepts of the COSMIC method. Understanding these concepts is important because to measure the functional size of a real piece of software the measurer must map from the actual artifacts of the piece of software (e.g. its statement of requirements or its physical implementation) onto the concepts of the COSMIC models
- an overview of the general COSMIC measurement process, which consists of three phases:
  - the Measurement Strategy, performed before starting a measurement (subsection 2.3.1)
  - the Mapping Phase (subsection 2.3.2)
  - the Measurement Phase (subsection 2.3.3)

The result of the measurement process is a size measure expressed in 'COSMIC Function Points' (or 'CFP'). These phases are illustrated in Fig. 2.0 below.

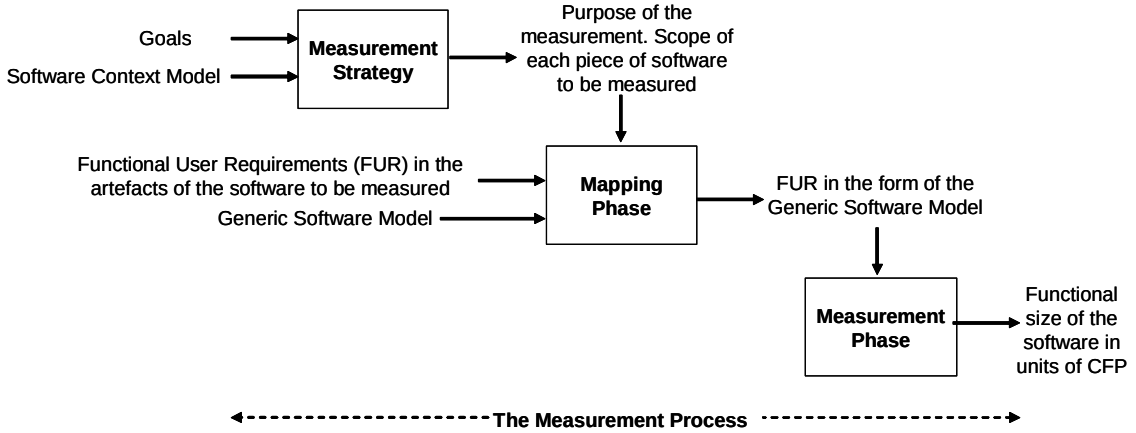

Figure 2.0 – Structure of the COSMIC method

The overviews of subsections 2.3.1, 2.3.2 and 2.3.3 are each expanded in a chapter of the Measurement Manual, where the complete and detailed definitions, principles and rules of the method are given with illustrative examples.

2.1 Applicability of the COSMIC method

2.1.1 Applicable domains

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

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

2.1.2 Non-applicability

The COSMIC measurement method has not yet been designed to take into account the functionality of mathematically-intensive software, that is, software which is characterized by complex mathematical algorithms or other specialized and complex rules, such as in expert systems, simulation software, self-learning software, weather forecasting systems, etc., or which processes continuous variables such as audio sounds or video images, such as, for instance, in computer games, musical instruments, etc. Nor does the COSMIC method attempt to measure aspects of functionality such as 'complexity' (however defined) that might be considered to contribute to software 'size'.

For software with such functionality it is possible, however, to define local extensions to the COSMIC measurement method. The Measurement Manual explains in what contexts such local extensions should be used and provides examples of a local extension.

2.2 The COSMIC software models

This section provides an overview of the COSMIC method and all of its basic concepts. The definitions of all these concepts are given in the document 'COSMIC Method v3.0: Documentation Overview and Glossary of Terms'. In this section, the first time the term for one of these concepts is used, it is given in **bold**.

It is essential that measurers understand ALL of the COSMIC models and basic concepts described below and are able to apply ALL of the principles and rules described in the chapters of the Measurement Manual when carrying out a measurement. Only by this discipline can the measurer be sure that measurements are meaningful and will be repeatable by other measurers on the same software and/or may be compared with measurements made by other measurers in different software environments.

2.2.1 Functional User Requirements

The COSMIC measurement method involves applying a set of models, principles, rules and processes to the **Functional User Requirements** (or **'FUR'**) of a given piece of software. The result is a

numerical 'value of a quantity'[6] representing the **functional size** of the piece of software according to the COSMIC method in units of 'COSMIC Function Points' (or 'CFP').

The functional size produced by the COSMIC measurement method is designed to be independent of any implementation decisions embedded in the operational artifacts of the software to be measured. 'Functionality' is concerned with 'the information processing that the software must perform for its **users'**.

More specifically, a statement of FUR describes 'what' the software must do for the **functional users.** These are 'the senders and intended recipients of data to and from the required functionality'. A statement of FUR excludes any technical or quality requirements that say 'how' the software must perform. Only the FUR are taken into account when measuring a functional size.

Extracting the functional user requirements from software artifacts in practice

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

Functional User Requirements can be derived from software engineering artifacts that are produced before the software exists, such as 'Requirements Definition' documents, the results of data or functional analysis of the requirements, etc, Hence the functional size of software can be measured prior to its implementation in a computer system.

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

Extracting or deriving the functional user requirements from software artifacts

The process to extract the FUR from different types of software engineering artifacts or to derive them from installed software and to express them in the form of the COSMIC software models will obviously vary depending on the types of artifacts. Such processes are domain-dependent and vary so much that they cannot be dealt with in the COSMIC method. The latter assumes that the functional user requirements of the software to be measured either exist or can be extracted or derived from its artifacts. However, COSMIC also publishes domain-dependent 'Guidelines' which describe some aspects of deriving FUR.[7]

The COSMIC method therefore limits itself to describing and defining the concepts of the COSMIC software models, i.e. the targets of the extraction or derivation process. These concepts are embodied in two COSMIC software models – the 'Software Context Model' and the 'Generic Software Model'.

2.2.2 The COSMIC Software Context Model

A piece of software to be measured must be carefully defined (in the measurement **scope**) and this definition must take into account its context of any other software and/or hardware with which it

[6] As defined by ISO, see 'The COSMIC Method v3.0: Documentation Overview and Glossary of Terms'

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

interacts. This Software Context Model introduces the principles and concepts needed for this definition.

PRINCIPLES – The COSMIC Software Context Model
a) Software is bounded by hardware
b) Software is typically structured into **layers**
c) A layer may contain one or more separate 'peer' pieces of software and any one piece of software may further consist of separate **peer components**
d) Any piece of software to be measured, shall be defined by its measurement scope, which shall be confined wholly within a single layer
e) The scope of a piece of software to be measured shall depend on the **purpose** of the measurement
f) The functional users of a piece of software shall be identified from the functional user requirements of the piece of software to be measured as the senders and/or intended recipients of data
g) A piece of software interacts with its functional users via **data movements** across a **boundary** and the piece of software may move data to and from **persistent storage** within the boundary
h) The FUR of software may be expressed at different **levels of granularity**
i) The level of granularity at which measurements should normally be made is that of the **functional processes** (see section 2.2.3)
j) If it is not possible to measure at the level of granularity of the functional processes, then the FUR of the software should be measured by an approximation approach and scaled to the level of granularity of the functional processes[8]

These principles and concepts will now be elaborated and illustrated with some simple examples.

To do this, we need to distinguish two views of a computer hardware/software system, that is, of the context of a piece of software to be measured, namely

- The 'physical' view, which shows how in practice the software is typically structured into a hierarchy of layers, each with its own specialist function. This view shows that in reality, all communication with any piece of software takes place via hardware devices and (maybe) other intermediate software layers

- The 'logical' view, which is an abstraction of the physical view used for functional size measurement purposes. This view shows that the functional users of a piece of software to be measured ('the senders and/or intended recipients of data') interact with the software across a boundary and that the software moves data to and from persistent storage. In this abstraction all hardware and software that enables these interactions is ignored.

Whilst only the second view is needed for functional size measurement purposes, it is helpful to explain both views, since measurers must be able to distinguish the two views. Furthermore, the COSMIC method uses terms such as 'layer' and 'peer component' in very specific ways which need to be understood. (These terms are used with many different meanings in the software industry.)

Fig. 2.2.2.1 illustrates the physical view for a typical piece of business application software in its context of a layered software architecture comprising the operating system, device drivers, etc, and of

[8] The subject of scaling between different levels of granularity is dealt with in the COSMIC method v3.0 document 'Advanced and Related Topics'

the hardware. Fig. 2.2.2.2 illustrates the same physical view for a simple example of real-time embedded software.

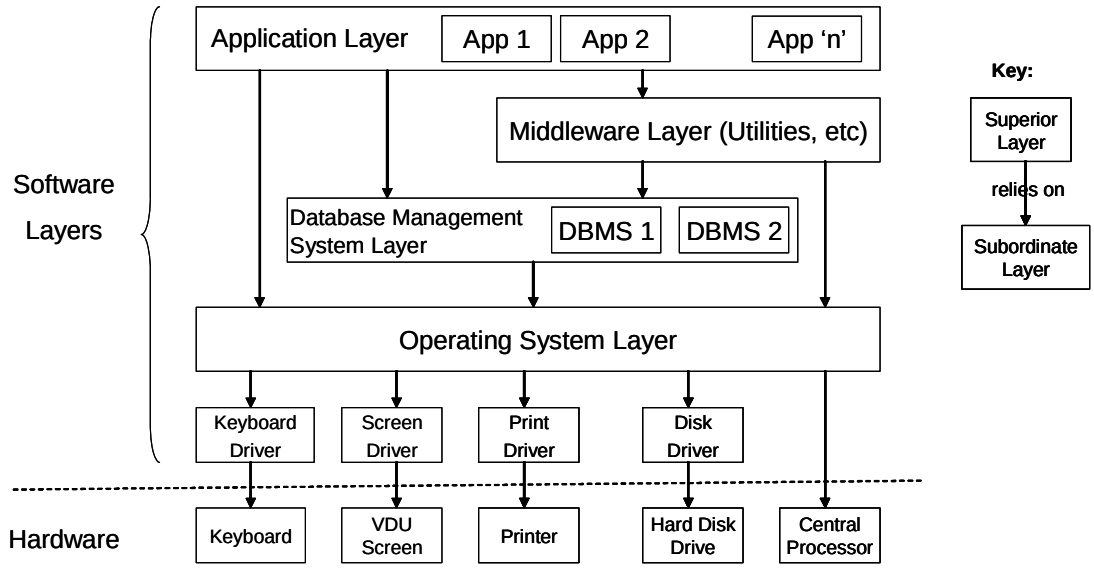

Fig. 2.2.2.1 Typical layered software architecture for a Business/MIS computer system

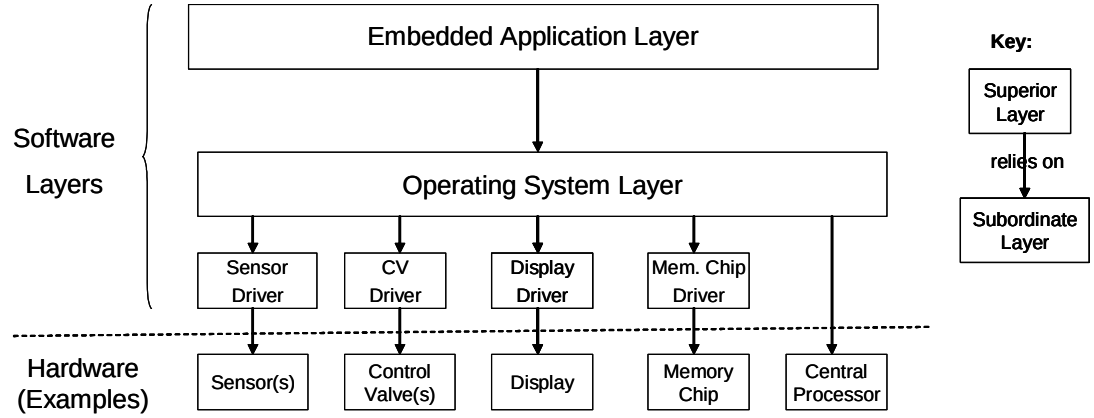

Fig. 2.2.2.2 Typical layered architecture for a real-time embedded-software computer system

Principle (a).

Software used by a human user is bounded by I/O hardware such as a mouse, a keyboard, a printer or a display; real-time embedded software is typically bounded by engineered devices such as sensors or relays. Software is also bounded by 'persistent storage' hardware such as a hard disk, or other types of memory that can be used to retain data.

Principle (b)

If the piece of software to be measured is part of a designed, layered architecture, it should be easy to decide on the layer to which the piece belongs. However, if a software environment has grown and

evolved over time, the layers (if any) may not be clearly distinguishable. For these circumstances, the COSMIC method includes some rules for distinguishing layers.

Principle (c)

For example, the separate main components of a business application (e.g. a 'three-peer' architecture of a 'front end / user interface' component, a 'business rules' component and a 'data services 'component) are peer components. Such main peer components may be sized separately and the COSMIC method gives rules for such sizing. This ability to measure separately the sizes of the main components of a piece of software when they execute on different technical platforms is very important in practice for the purposes of performance measurement and estimating.

Any piece of software in any layer can of course be decomposed into its components at various levels (e.g. down to individual modules or object-classes) and the COSMIC method can be used to measure a functional size at any such level. Measurers will, however, need to define their own local standard levels of decomposition (in liaison with the software or system architect) below the level of the main peer components in any one layer if they wish to ensure comparability of measurements from different sources.

Principle (d)

Principle (d) requires that the scope of any piece of software to be measured, shall be confined wholly within a single layer. The reason for this is that each layer has a specialized function and may be developed using different technology from other layers. So it may or may not make sense to measure the sizes of some pieces of software residing in two or more layers and then to aggregate those sizes as if the result represented the size of a single entity. The resulting size measure might, like the sum of the sizes of some apples and oranges, be very difficult to interpret and/or to compare with other functional size measurements. For rules on aggregating the sizes of pieces of software in different layers, see the section on aggregating measurement results in the Measurement Manual.

Principle (e)

As an example, assuming the pieces of application software in Figs. 2.2.2.1 and 2.2.2.2 are each separate pieces of software, developed by their own project teams, then in each case, for most normal measurement purposes it would make sense to define the measurement scope as the 'whole application'. However, if an application is developed as three main peer components (as mentioned in relation to principle (c) above) each using different technologies and/or by different project teams, then if the purpose is project effort estimating, it would make sense to define three separate measurement scopes, one for each peer component. The size measurements of each of the three separate components could then be used as input to an estimating formula that is able to account for the different technologies and/or project team characteristics used for each component.

Principle (f)

To illustrate this principle and the next principle (g), we need to examine the logical view of the software to be measured. Figures 2.2.2.3 and 2.2.2.4 illustrate this logical view showing the interaction of 'functional users' with a piece of business application software and with a piece of real-time embedded application software respectively.

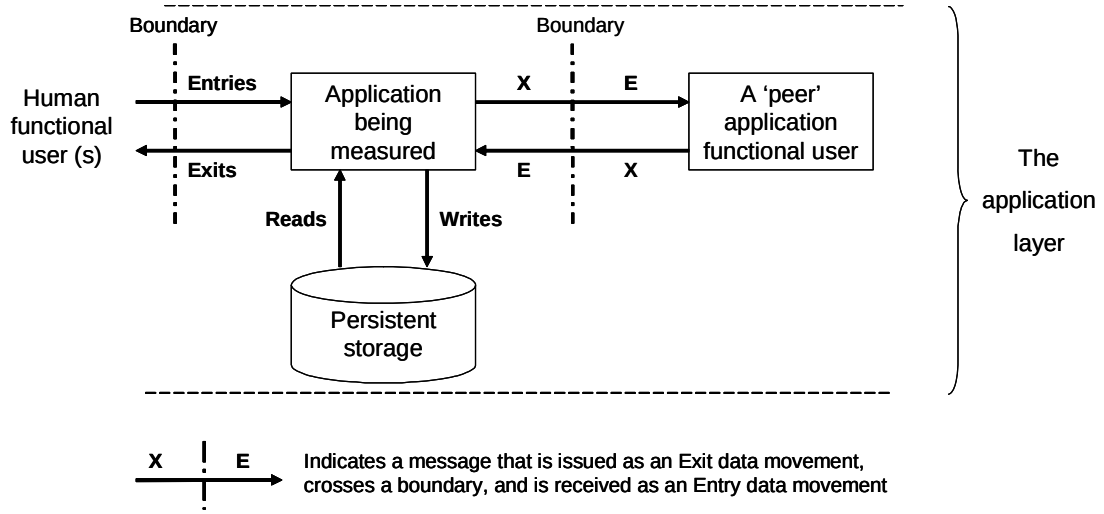

Figure 2.2.2.3 - A business application with both humans and another 'peer' application as its functional users (logical view)

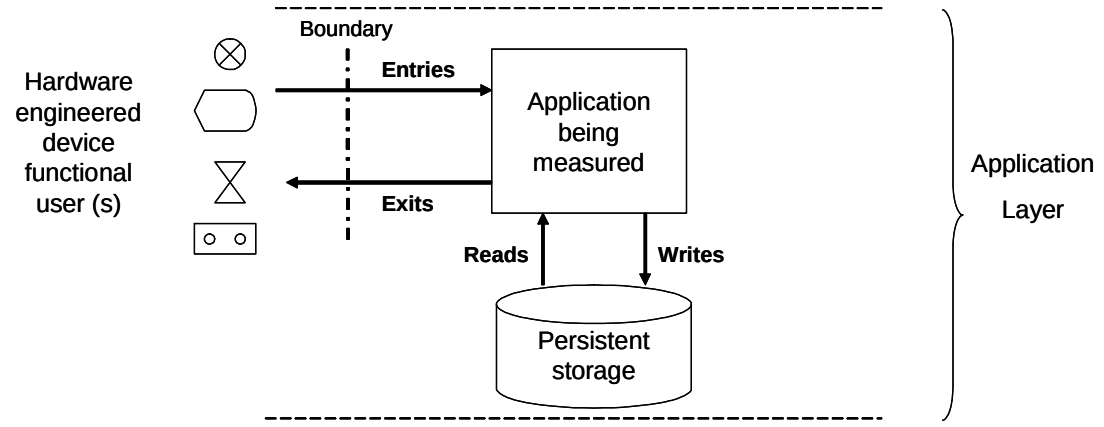

Figure 2.2.2.4 - A real-time embedded software application with various hardware engineered devices as its functional users (logical view)

Consider the example of a piece of business application software to be measured. Fig. 2.2.2.1 shows that the 'users' of the application could be considered to include the operating system, any of the hardware devices (e.g. the keyboard, the printer, etc.) and the human users because they can all be said to 'interact' with the application, directly or indirectly. But not all of these (types of) users will be specified in the FUR as the senders and intended recipients of data to/from the application. The operating system and the hardware devices are 'enablers' of these data exchanges, rather than senders or intended recipients.

For a piece of business application software, the FUR will normally only ever describe the required functionality from the viewpoint of the human users of the application, and maybe other peer applications that send or receive data to/from the application. These humans and peer applications will therefore be the 'functional users' of the application, as how in the logical view of Fig. 2.2.2.3.

Because of the strict separation of functionality into layers as in Fig. 2.2.2.1, the FUR of a business application can normally ignore any software layers or hardware devices such as the operating system

or screens that enable the interaction of the functional users with the application. There should never normally be any doubt about the identity of the functional users.

For the example of the embedded real-time software, its FUR would normally describe the functionality required from the viewpoint of the hardware devices (the sensors, valves, etc) that the software must support. These devices will therefore be the 'functional' users of the embedded software, as shown in Fig. 2.2.2.4. But as we shall see in the Measurement Manual, though uncommon, the FUR of some types of software may sometimes be written where there is more than one type of functional user, thus giving rise to different functionality and hence of functional size.[9]

Principle (g)

Figs. 2.2.2.3 and 2.2.2.4 also illustrate that functional users interact with software across a boundary via two types of movement of data (**Entries** and **Exits**). Software also exchanges data with persistent storage hardware via two types of data movement (**Reads** and **Writes**). These data movements are defined further in section 2.2.3

The 'boundary' is defined as 'a conceptual interface between the software under study and its functional users'. This boundary should not be confused with any line that might be drawn on a diagram to depict the scope of a piece of software to be measured, or around a software layer.

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

Fig. 2.2.2.5 now illustrates the logical view of some business application software that has been developed as three main peer components as described in principles (c) and (e) above. Supposing that the peer components have been developed using different technologies, it is likely that the purpose of the measurement would dictate that a separate measurement scope should be defined for each peer component.

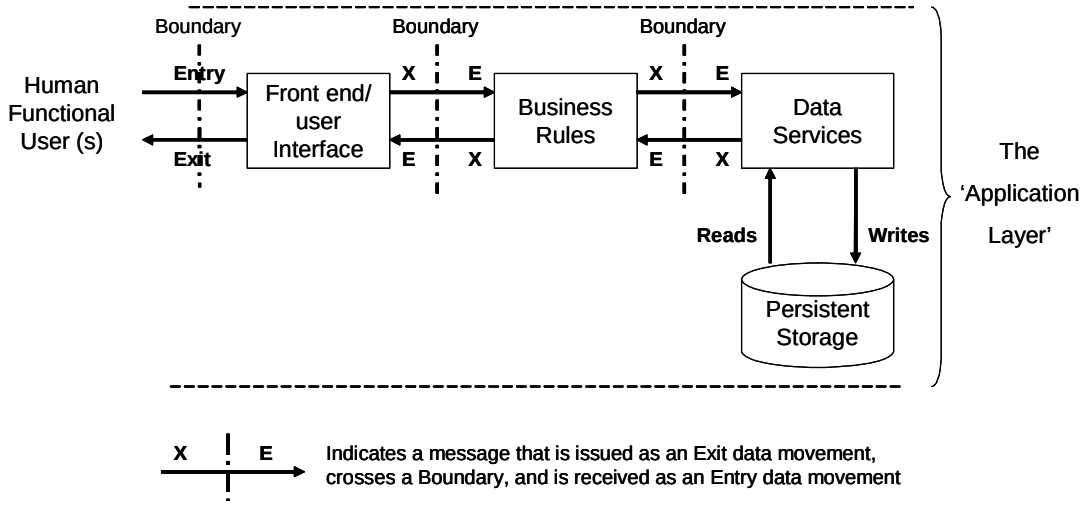

Figure 2.2.2.5 – A business application when its main 'peer' components must be measured separately (logical view)

[9] An exception when the operating system can be a functional user of an application is when the operating system is required by the application FUR to supply, for example, a 'clock tick' to start a functional process in the application.

In the logical view of Fig. 2.2.2.5, we see that the 'front end / user interface' component has humans and the 'business rules' components as its functional users, each interacting with the component via Entries and Exits across a boundary. This Figure also shows that only the 'data services' component interacts with the persistent storage, and that its functional user is the 'business rules' component. With these logical views, the FUR of each component can be measured separately.

Principle (h) and (i)

The FUR of software may be expressed at different 'levels of granularity'. (Note that the concept of 'level of granularity' is concerned with the level of detail of the description of a piece of software. This must be distinguished from 'level of decomposition, which is concerned with the breakdown of software into its component parts.) The level of granularity at which measurements should normally be made is that of the functional processes (see section 2.2.3). When starting a new software development, the process of determining the functional user requirements ('FUR') of software typically starts with defining and agreeing a 'high-level' statement of requirements, which is then refined and worked out in more detail. The FUR of a piece of software of a given scope may therefore exist at different levels of granularity. A typical example of using a 'functional analysis' technique to determine the FUR of a piece of software might result in the following hierarchy of levels of the FUR.

A 'level 1' main function', when analysed in more detail is shown to consist of a number of 'level 2 functions', Each of these consists of 'level 3 sub-functions', each consisting of 'level 4 sub-sub-functions', etc. At some point in this hierarchy, the analysis will reveal individual functional processes (See section 2.2.3 for more on these; for now we need think of these only as standard chunks of functionality that we can measure.)

As the analysis 'zooms in' on more and more detail, the measured functional size may well appear to increase because more details must be taken into account. (Note: this phenomenon is different to that of 'scope creep', in which size increases because the scope of the software increases.)

Consequently, in order to be able to compare measurements sensibly from different sources or to use measurements in some other process, all measurements must be made at (or scaled to) a standard level of granularity, which we call the 'functional process level of granularity'. In most cases, when the purpose is to measure the functional size of a fully-specified or of an existing piece of software, the functional process level of granularity at which to measure is self-evident.

Everyone is familiar with the idea of measuring distances on maps using different scales, for example, where 1 km is represented on a map by 1 cm or 1 mm, and converting distances from one scale to another. But when measuring the functional size of a piece of software using COSMIC, we have only one standard 'functional process level of granularity' and only one unit of measure. So if we need to compare measurements made at different levels of granularity, we must invent our own local scaling factors to convert sizes to the units at the standard functional process level of granularity. These concepts are explored further in the section on the standard level of granularity in the Measurement Manual.

Principle (j)

The problem of needing to measure a piece of software at a level of granularity higher than that of its functional processes normally arises only in the early stages of new software developments whilst the requirements are still evolving. In such circumstances when it is not possible to measure at the level of granularity of the functional processes, then the FUR of the software should be measured by an approximation approach and scaled to the level of granularity of the functional processes (see the chapter on early sizing in the document 'COSMIC Method v3.0: Advanced and Related Topics').

In summary, the Software Context Model of the COSMIC measurement method provides a set of concepts and principles, namely software layers and peer components, the scope of a piece of software to be measured, its functional users, data movements and a boundary to help measure functional user requirements, which may be drawn up at different levels of granularity. During the

Measurement Strategy process (as described in section 2.2.4) we apply these concepts and principles to the FUR of the software to be measured to answer questions such as 'what measurement is required?' or 'how do we interpret this measurement?'

2.2.3 The COSMIC Generic Software Model

Having interpreted the FUR of the software to be measured in terms of the Software Context Model, we now apply the Generic Software Model to the FUR to identify the components of the functionality that will be measured. This Generic Software Model assumes that the following general principles hold true for any software that can be measured with the method. See the Glossary for the definition of all terms[10].

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

These principles arise from a common understanding of software as follows.

Principles (a) to (e)

The task of software is to respond to events that occur *on the functional users' side of* its boundary, i.e. in the world of its functional users. A functional user notifies software of the occurrence of an event and may send data about the event. The software must do something useful for the functional user in response to that event. We call this 'something useful' a 'functional process'. All software FUR can therefore be expressed as a list of types of events and of the corresponding functional processes that carry out the response of the software to each event.

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

Principles (c) and (d) tell us that functional processes can be regarded as consisting of two types of sub-processes, namely data movements and data manipulation. Software can only move and/or manipulate data.

Fig. 2.2.3.1, below, illustrates principles (b) to (d) of the Generic Software Model.

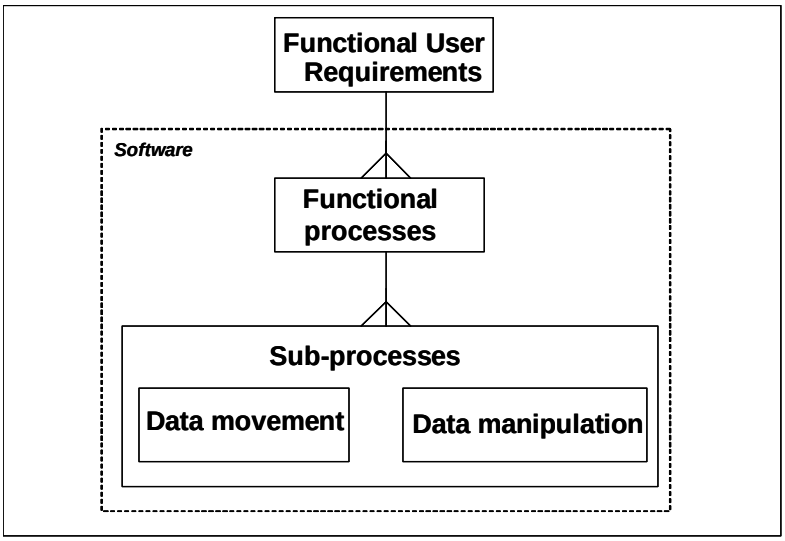

Figure 2.2.3.1 – The structure of Functional User Requirements

Principles f) and g)

Each data movement carries only one data group, that is, data about a single object of interest, i.e. a thing 'of interest' to a functional user. As an example in the domain of business application software, a relatively simple functional process to enter an order might typically involve the following data movements (objects of interest are all given in inverted commas in the following):

- Two Entries of data groups about the 'order' and 'order-item' (assuming a multi-item order). The first of these Entries of data describing the 'order' object of interest is the one that triggers (or starts) the functional process
- Two Reads of data groups about 'customer' and 'product' to validate that the customer is allowed to order and that the required products are valid and available
- Two Writes of data groups about the 'order' and the 'order-item' to move the entered data to persistent storage
- One or more Exits of data groups containing for instance an 'order-confirmation' message including the total order value, an instruction to the warehouse to pick each 'order-item', etc.

All of these objects of interest are real or conceptual things in the real world of the functional users (human, in this case), about which the piece of software is required to process data. They must be identified and distinguished in order to identify the data movements.

When measuring real-time or embedded software exactly the same principles apply, though very often the 'functional user' and the 'object of interest' are in practice virtually indistinguishable. For example, suppose a functional process needs to obtain the current temperature from a sensor, and assume the sensor is equipped such that it can communicate directly with software; the sensor is thus a functional user of the piece of software. In this case, the sensor sends an Entry data movement that has probably only two data attributes (the sensor ID and the temperature). These two attributes convey data about the sensor (as object of interest) – or though it could equally be argued that the object of interest is the 'thing' whose temperature is measured by the sensor.

Suppose then a very simple real-time functional process to measure temperature via a sensor and to control it against a target temperature. This functional process is triggered at regular intervals by a signal from a clock, and would consist of the following data movements.

- An Entry from the clock that triggers (starts) the functional process
- An Entry from the temperature sensor containing the sensor ID and the current temperature
- A Read of the target temperature from persistent storage (assuming that the target temperature can be set and varied by another functional process)
- An Exit to the heater containing a signal to switch it on or off, if the heater's state needs to be changed

Note that in this very simple example, all but one of the data groups consists of only one data attribute.

Principle (h)

This principle tells us that the four types of data movements are distinguished by their source and destination. Data movements either cross the boundary between the software being measured and its functional users (Entries and Exits) or move between the software and persistent storage (Reads and Writes). These relationships are shown in fig. 2.2.3.2 below.

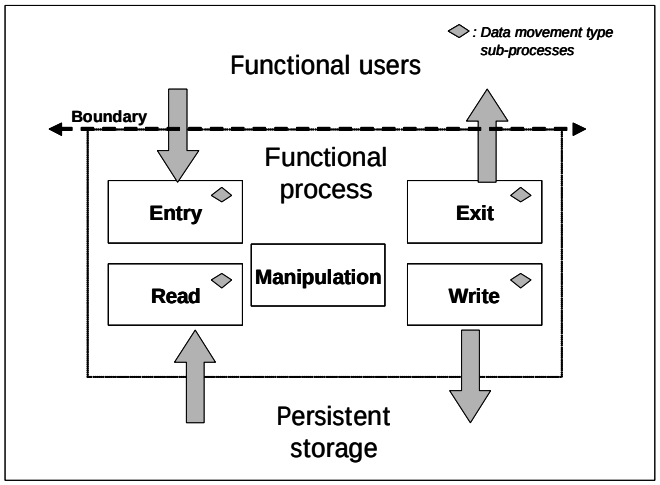

Figure 2.2.3.2 – The components of a functional process and some of their relationships

Principle (i)

This principle states that a functional process must have at least two data movements. This follows from the preceding principles. A functional process that receives only one data movement and does nothing with it would be practically useless. Therefore, all functional processes must have at least one data movement informing it about the occurrence of an event (an Entry), and at least one other data movement as a response (or useful outcome), either to a functional user (an Exit) or to persistent storage (a Write).

Principle (j)

For measurement purposes, and given the software domain for which the method has been designed, the COSMIC method assumes a simplification of the Generic Software Model.

As a first approximation in this version of the measurement method, data manipulation-type sub-processes, illustrated in figure 2.2.3.2, are not recognized separately but are considered to be associated with or part of specific data movement sub-processes. (From now on therefore, for brevity, the expression 'data movement' will be used, rather than 'data movement sub-process'.). The reason for this approximation 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 the subsection on the data manipulations associated with the data movement, in the Measurement Manual.

Given this approximation we can see why the standard COSMIC 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. The Measurement Manual also points out the need for caution when measuring very small pieces of software, and especially small changes to software, where the assumption of principle (j) may no longer be valid.

The Measurement Manual also provides a mechanism for defining a local extension to the COSMIC method that enables an organization to account explicitly for data manipulation functionality, if it should so desire. For such cases, special reporting conventions are also defined.

By using the concepts and their definitions, and the principles and rules of the COSMIC measurement method, the functional user requirements extracted from the artifacts of a piece of software can be mapped onto the 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 measurement rules and processes are then applied to this instantiated Generic Software Model in order to produce a value of a quantity representing the functional size of the piece of software – see 2.3.3 below for the measurement rules.

2.3 Overview of the COSMIC measurement process

Three distinct and related phases are necessary to measure the functional size of a piece of software:

1. Setting the measurement strategy using the principles of the Software Context Model
2. Mapping the artifacts of the software to be measured onto the Generic Software Model
3. Measuring the specific elements of this model.

2.3.1 The Measurement Strategy Phase

Before starting a measurement, the measurer must agree with the sponsors of the measurement and must document (a) the purpose of the measurement, (b) the scope of each piece of software to be measured, (c) the functional users of each piece and hence the boundary of each piece, and (d) the level of granularity at which the measurements are required. Establishing a clear statement of the purpose (a) of the measurement is critically important because it determines the other three parameters (b), (c), and (d). Determining these three parameters will very often be an iterative process.

(a) The purpose of the measurement

The purpose of the measurement establishes why the measurement is being carried out and what the results will be used for. This in turn will help determine not only the other three parameters of the Measurement Strategy, but also, for example, the required accuracy of the measurement. (As will be seen in the chapter on early sizing in the 'Advanced and Related Topics' document, it is possible to estimate a functional size with an approximation variant of the basic COSMIC method. This variant can be applied early in a project life-cycle before all the requirements have been established in sufficient detail to measure a size according to the exact rules of the method).

(b) The scope of the software to be measured

The overall scope of the software to be measured follows from the purpose. The overall scope establishes what software functionality will be included in a measurement (and what will be excluded).

Depending on the purpose, the overall scope may need to be sub-divided into a number of separate pieces of software functionality, each to be measured separately with its own scope.

Such a sub-division of the overall scope would first be necessary if the overall scope includes software in more than one layer, because any piece of software to be measured must reside wholly within one layer. Second, a sub-division might be necessary, for example, if the measurement purpose is concerned with estimating, and the total software to be measured comprises separate components that will be developed with different techniques and/or technologies and/or execute on separate technical platforms and/or be developed by separate teams. Each component would then have its own measurement scope.

(c) The functional users and the boundary of each piece of the software to be measured

The functional users of each piece of software can be identified by examining the data flows in and out of the software as stated or implied in its functional user requirements and taking into account the purpose of the measurement. The functional users will be the senders or intended recipients of the data.

In most circumstances, identifying the functional users is obvious from the purpose of the measurement and from the FUR. Exceptionally, the functional users may vary depending on the purpose of the measurement. An example will be given in the subsection on functional users in the Measurement Manual that illustrates a case where there may be a choice.

When the functional users are known, the boundary - the conceptual interface between the functional users and the piece of software to be measured - can be easily established.

(d) The level of granularity of the measurements

The level of granularity of the FUR of a piece of software at which the measurements should normally be made is the level at which the functional processes are identified and their breakdown into data movements is defined.

Where the purpose is to measure the FUR of some fully-specified or existing piece(s) of software, the level of granularity is normally self-evident when the functional processes have been identified.

On the other hand, in the early stages of software development where the purpose is to measure the FUR of some piece(s) of software as they evolve, the FUR may have to be measured before the point where any or all of the individual functional processes have been revealed and their data movements defined. To complicate matters further, the FUR of the different pieces may exist at different levels of granularity at the time the measurement is needed. In these circumstances, some ‘functional units‘ will need to be defined locally that can be identified and counted in the available software artifacts and a method will be needed to scale from the level of granularity where the size is measured by the counts of the functional units to the level of granularity at which functional processes can be identified and sized (i.e. to COSMIC Function Points). Such scaling methods are discussed in the section on identifying a standard level of granularity of the Measurement Manual and in the document ‘Advanced and Related Topics’.

Having completed these steps (a) to (d), some iteration may be needed. For example, as some requirements are revealed in more detail, this could lead to a need to refine the scope of the piece(s) of software to be measured.

A full discussion with definitions and more examples of purpose, scope, functional users and of the level of granularity is given in the chapter on the Measurement Strategy of the Measurement Manual.

2.3.2 The Mapping Phase

The Mapping Phase takes as input the functional user requirements of each piece of software to be measured extracted from its artifacts (as they are found or documented within the organization or as they are inferred from the existing physical software), taking into account the Software Context Model. The output from the Mapping Phase is an instance of the Generic Software Model.

The steps of instantiating a Generic Software Model in the Mapping Phase are then:

- identify the events in the world of the functional users that the software must respond to and hence identify the functional processes
- identify the data movements (Entries, Exits, Reads and Writes) of each functional process, which in turn depends on identifying the data groups that are moved.

A full discussion with definitions and examples of the concepts and steps of the Mapping Phase is given in the related chapter of the Measurement Manual.

2.3.3 The Measurement Phase

The Measurement Phase takes as input an instance of the Generic Software Model and, using a defined set of rules and processes, produces a numerical value, the magnitude of which is directly proportional to the functional size of the model, based on the following principle:

PRINCIPLE – The COSMIC measurement principle
The functional size of a piece of software is directly proportional to the number of its data movements.

The characteristics of the set of rules and processes governing the production of this numerical value are as follows:

Characteristic 1 – Unit of measure

The measurement standard, namely 1 CFP (COSMIC Function Point), is defined by convention as equivalent to a single data movement.

Characteristic 2 – Additivity of sizes within a given measurement scope

The functional size of a functional process is defined as the arithmetic sum of the number of its constituent data movements. By extension, the functional size of any piece of software with a given measurement scope[11] in any layer in the software model is the arithmetic sum of the functional sizes of the functional processes of that piece of software.

Characteristic 3 – Size of change(s) to a piece of software

The functional size of any required functional *change(s)* to a piece of software is by convention the arithmetic sum of the number of its data movements that must be added, modified and deleted as a consequence of the required change(s).

Characteristic 4 – The minimum and maximum size of a functional process

According to these characteristics, as already established in principle (g) of the Generic Software Model, the minimum functional size for a single functional process is 2 CFP, because the smallest

[11] For rules on the aggregation of sizes of pieces of software in different measurement scopes, see the chapter of the Measurement Manual on the measurement phase.

functional process must have at least one Entry (as input), and either one Exit (as output) or one Write (as an alternative useful outcome).

As a change may affect only one data movement, it follows that the minimum size of a change to a functional process is 1 CFP.

Further, according to these characteristics, there is no upper limit to the functional size of any one functional process and hence no upper limit to the functional size of any piece of software.

Additional principles and detailed rules and processes of the Measurement Phase for determining the functional size from the FUR of a piece of software expressed in the Generic Software Model are presented in the related chapters of the Measurement Manual and are summarized in its appendices B and C.

APPENDIX A - COSMIC CHANGE REQUEST AND COMMENT PROCEDURE

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

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

mpc-chair@cosmicon.com

Informal General Feedback and Comments

Informal comments and/or feedback concerning the Measurement Manual, such as any difficulties of understanding or applying the COSMIC method, suggestions for general improvement, etc should be sent by e-mail to the above address. Messages will be logged and will generally be acknowledged within two weeks of receipt. The MPC cannot guarantee to action such general comments.

Formal Change Requests

Where the reader of the Measurement Manual believes there is an error in the text, a need for clarification, or that some text needs enhancing, a formal Change Request ('CR') may be submitted. Formal CR's will be logged and acknowledged within two weeks of receipt. Each CR will then be allocated a serial number and it will be circulated to members of the COSMIC MPC, a world wide group of experts in the COSMIC method. Their normal review cycle takes a minimum of one month and may take longer if the CR proves difficult to resolve. The outcome of the review may be that the CR will be accepted, or rejected, or 'held pending further discussion' (in the latter case, for example if there is a dependency on another CR), and the outcome will be communicated back to the Submitter as soon as practicable.

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

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

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

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

Questions on the application of the COSMIC method

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