The COSMIC Functional Size Measurement Method

Version 4.0

Introduction to the COSMIC method of measuring software

Version 1.0, May 2014

Acknowledgements

Version 1.0 Reviewers		
Alain Abran École de Technologie Supérieure, Université du Québec Canada	Diana Baklizky Métricas Brazil	Lindsay Davies Freelance Editor United KIngdom
Peter Fagg Pentad United Kingdom	Cigdem Gencel Free University of Bolzano-Bozen Italy	Arlan Lesterhuis* The Netherlands
Bernard Londeix Telmaco United Kingdom	Hassan Soubra Ecole Supérieure des Techniques Aéronautiques et de Construction Automobile France	Charles Symons* United Kingdom
Monica Villavicencio ESPOL Ecuador	Frank Vogelezang Ordina The Netherlands	Chris Woodward CW Associates United Kingdom

- Editors of this document.

Version Control

The following table summarizes the evolution of this document.

DATE	REVIEWER(S)	Modifications / Additions
2000	COSMIC Core Team	'Introduction & Overview' Slide Presentation & Supplementary Notes
September 2007	14 reviewers from 7 countries	First version of the 'Method Overview' document for v3.0 of the COSMIC method
May 2014	COSMIC Measurement Practices Committee	Version 1.0 of the 'Introduction to the COSMIC method of measuring software', for the COSMIC version 4.0.

Foreword

The COSMIC method is an internationally standardized method (ISO 19761, see [1]) for measuring a size of the functional requirements of most software domains, including business application (or ‘management information system’) software, real-time software, infrastructure software and some types of scientific/engineering software.

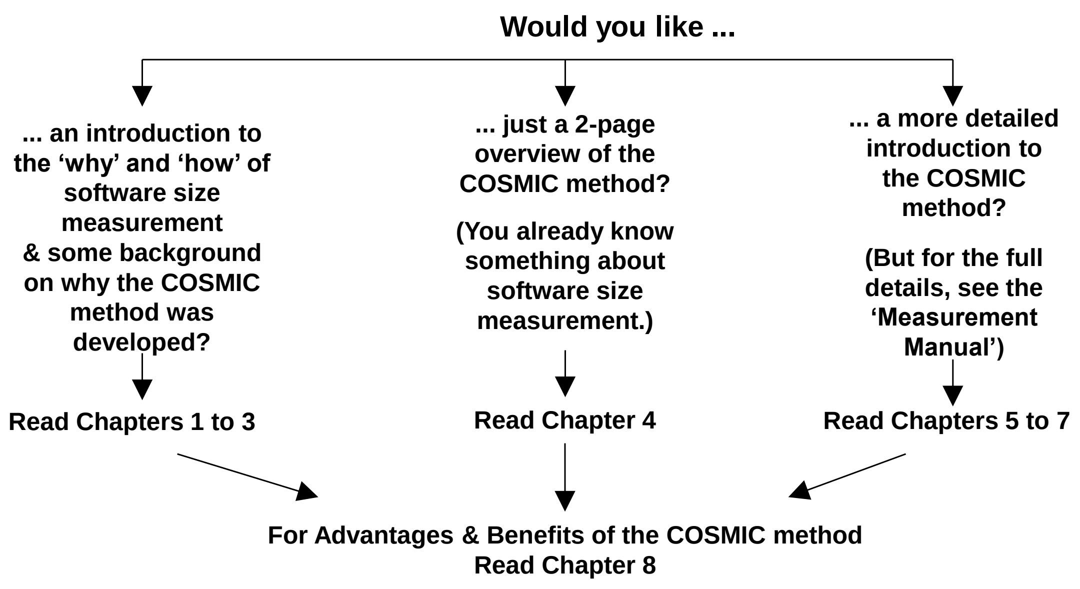
‘COSMIC’ stands for the ‘Common Software Measurement International Consortium’. It was formed in 1998 by a group of software measurement experts from Australia, Europe and North America, with the aim of developing a new method of measuring software size based on well-established software engineering principles and metrology criteria. Its publications are completely open and available for free download.

The method is now very widely used around the world, in all the domains for which it was designed, for purposes such as the measurement of sizes in software contracts, and is successfully applied for project performance measurement, benchmarking and estimating.

Aims of this ‘Introduction’ document

This document is aimed at people who need an introduction to software size measurement and its uses, and who want an overview of the COSMIC method, but not all of its details.[1]

Use the diagram below to decide which chapters to read.


Would you like ...
... an introduction to the ‘why’ and ‘how’ of software size measurement & some background on why the COSMIC method was developed?
Read Chapters 1 to 3
... just a 2-page overview of the COSMIC method?
(You already know something about software size measurement.)
Read Chapter 4
... a more detailed introduction to the COSMIC method?
(But for the full details, see the ‘Measurement Manual’)
Read Chapters 5 to 7
For Advantages & Benefits of the COSMIC method
Read Chapter 8

COSMIC method documentation

All COSMIC method documentation except the ISO 19761 standard can be downloaded from the portal of the COSMIC web-site www.cosmicon.com/dl_manager.asp.

The principal documents that define the method are:

- The ISO 19761 standard (‘Software Engineering – COSMIC – A functional size measurement method’), which contains the definitions and basic rules of the method. (At the time of writing, the 2012 version of this standard has not yet been updated to v4.0 of method.)

[1] The Introduction replaces the ‘Method Overview’ document and applies to version 4.0 of the method.

- The ‘COSMIC Method version 4.0: Measurement Manual’, which provides all the principles and rules and the glossary of terms. It also provides further explanation and many more examples in order to help measurers to understand and to apply the method. This is the main ‘working document’ that Measurers will need in practice.

The Figure below shows the structure of COSMIC documentation. At the time of publication of this Introduction, most of these documents are available; an asterisk indicates the document is still under development. All are being upgraded to bring them in line with version 4.0 of the method.

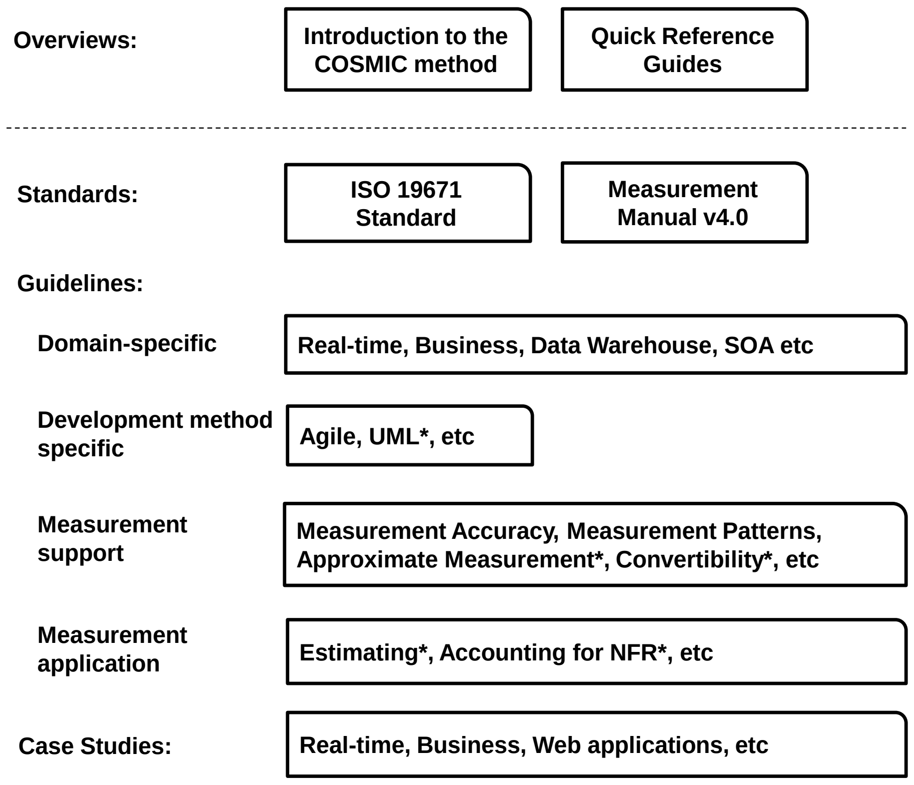

Overviews:
Introduction to the COSMIC method
Quick Reference Guides
Standards:
ISO 19671 Standard
Measurement Manual v4.0
Guidelines:
Domain-specific
Real-time, Business, Data Warehouse, SOA etc
Development method specific
Agile, UML*, etc
Measurement support
Measurement Accuracy, Measurement Patterns, Approximate Measurement*, Convertibility*, etc
Measurement application
Estimating*, Accounting for NFR*, etc
Case Studies:
Real-time, Business, Web applications, etc

Figure – COSMIC Documentation structure

Translations of the ‘Measurement Manual’ are also available in 12 other languages. All these can be found on the portal of the COSMIC web-site www.cosmicon.com/dl_manager.asp.

This same web-site has more general background information on functional size measurement and its uses, on the COSMIC organization and its activities, on suppliers of COSMIC-related services, COSMIC certification examinations, COSMIC Newsletters, how to contribute to and get COSMIC benchmark data, etc., as well as measurement support tools and many COSMIC-related research papers, all for free download.

The COSMIC Measurement Practices Committee

May 2014

Table of contents

-

1

WHY MEASURE SOFTWARE SIZE?

1.1 Why would anyone want to 'measure' software?

The most likely reason to measure a size of some software is if you need to estimate the effort for its development. Your first thought may then be 'how big is the software?' The software size is usually the main driver of the amount of the development work that must be done.

Analogy: If you ask a supplier to estimate the effort for a job such as tiling the walls of a bathroom, the supplier will first want to know the surface area of the walls (i.e. the size) to be tiled. Then, knowing the normal rate at which tiles of the required size can be fixed in, say, tiles per hour (which we call 'productivity') the supplier can give a first estimate of the effort. The starting point for such an estimate is always:

Estimated Effort = Estimated Size divided by Productivity.

This initial estimate might need to be refined due to unusual corners or windows in the bathroom, but the main 'cost driver' for the effort is the area (= estimated size) to be tiled.

The estimation process is similar when estimating effort to develop or change some software. We will need to measure or estimate the size of the software to be developed or changed. Productivity data of software development projects that used technology similar to what will be used for the new software can be obtained

- from measurements of productivity of completed projects within your own organization,
- from sources of benchmark data such as the publicly-available ISBSG industry database at www.isbsg.org.

(By the way, be careful to distinguish 'software size', the topic of this Introduction, from 'project size'. A project may include other activities than developing software; project size is measured in units of effort such as 'work-hours', or staffing level, or duration.)

1.2 Software size measurements have many other uses

Measuring software sizes can be very valuable for many other purposes than project estimating. For example:

- **Comparison**. An organization may wish to compare productivity using an Agile approach to project management versus its traditional 'waterfall' approach. For this you must use the same method of measuring the software produced by all types of projects.
- **Controlling scope**. Tracking the size of a new piece of software as its requirements evolve helps control 'scope creep' and hence the project budget. Tracking the size delivered helps project managers to control progress.
- **Controlling defect density.** When a project is completed, you may want to track defects found in the first month of operation and report, say, the 'defect density' in defects per unit size.

See chapter 8 for more uses of software size measurement.

1.3 Who typically benefits from these measurements?

Major commercial software suppliers routinely measure software sizes and use them for new project effort estimating and for project productivity measurement. Their measurements are vital for managing risk and maintaining profitability. Software customers should benefit even more from using these measurements to control scope creep and their suppliers' price/performance, delivered quality, etc.

2

HOW TO MEASURE SOFTWARE SIZE?

2.1 Like any other unit of measurement, you need standards

The size of software can be measured in many ways and at different points in a software project life-cycle.

If you want to use software size measurements for multiple purposes across multiple activities, it is evident that you must adopt a standard way of measuring software size. The COSMIC method is an example of an internationally standardized Functional Size Measurement method (FSM), having been accepted as the International Standard ISO 19761.

2.2 What are the most important ways of measuring software size?

There are three main ways of measuring software size.

- You can count the source lines of code (SLOC) written to implement the software requirements.
- You can measure the size of the requirements for software.
- You can use a method related to the software development method or stage.

2.3 Counting source lines of code

Counting SLOC was one of the earliest ways of measuring software size. The advantage of SLOC sizes is that SLOC can be counted automatically by programs that analyze the source code. But SLOC counts have significant disadvantages.

- There are no universally accepted standards for SLOC counting, counts may vary from one counting program to another program.
- When a given set of software requirements is programmed, the numbers of SLOC will depend on the programming language used and maybe the skill of the programmer. Comparisons of productivity across projects, especially when using different programming languages, are therefore inherently difficult.
- You only know a SLOC size precisely when the software programs are finished. So it's difficult to use SLOC counts for estimating effort early in the life of a project. To estimate a size in SLOC, the project must have progressed to a point where you have some knowledge of the design and program structure and then you will need some experienced guesswork or analogies for the SLOC estimate.
- Source lines of code may not be identifiable with some programming languages and tools, which are based on selecting and setting parameters and options.

Nevertheless, SLOC counts are still used when the physical size of the software is relevant and in some software domains which have built up years of experience of using these measures. Also, several well-known software project estimating methods, e.g. COCOMO II [2], have been calibrated using SLOC sizes.

2.4 Measuring software requirements

Methods for measuring software functional requirements, known as 'Functional Size Measurement' (or 'FSM') methods, one of which is COSMIC, have the obvious advantage for project estimating that they can be used as soon as the 'functional user requirements' (FUR) are known. Most FSM methods also have variants that can be used for approximate sizing even before the FUR are known in full detail.

The other big advantage of FSM methods is that the sizes they measure are independent of the technology used to implement the software. In addition, for some of the FSM methods, their units of measurement are internationally standardized. FSM method measurement units are the closest equivalent that the software industry has to standard units (such as the meter for measuring length).

2.5 Other ways of measuring software size

There are many other methods of measuring software size, but they are almost all related to specific development methods or to measuring size at a particular stage in the development.

Examples include the 'Use Case Points' of UML, 'User Story Points' of Agile methods, 'Object Points' of Object-Oriented methods, and so on. None of these methods are well defined and supported by international users groups, or are portable across different development methods. None have been internationally standardized. Some, such as User Story Points, are actually highly subjective.

2.6 A closer look at software requirements – FUR and NFR

A closer look at software requirements shows that there are two types of software requirements, namely 'functional user requirements' (or 'FUR') and 'non-functional requirements' (or 'NFR'). In very simple terms:

- FUR state what the software must do for its users, in terms of tasks and services (ISO 14143-1);
- NFR are typically constraints that apply to the whole hardware/software system and/or the project that develops it.

The following example illustrates both FUR and NFR

BUSINESS EXAMPLE: Assume a company's Personnel System.

- *The FUR would specify that it must enable the entry and maintenance of all data about the company's employees including their name and address, date of birth and start of employment, grade, job title, department, qualifications, dependents, career progression and appraisal record, etc. The software must also provide enquiries against the stored data.*
- *NFR for this same Personnel System might specify: security-access controls, system availability, the technology to be used for the software, a target implementation date and such-like.*

REAL-TIME EXAMPLE: Assume the embedded software that controls the functions of a simple copier.

- *Its FUR would specify that it must support all user commands, e.g. initializing the system after power-on, responding to the user entering the number of required copies, the selection of black or colored copying, magnification, etc., and then controlling all the steps to produce the copies after the user presses the 'start' button. The software must also respond to sensors signaling that there is a paper jam, paper or ink has run out, etc.*
- *NFR for the copier might specify: system response times, a zero-defect target for the software, system availability criteria, etc.*

Note that many systems requirements that initially appear as non-functional evolve into software functional requirements as a project evolves.

EXAMPLE: Systems requirements for auditability or usability may appear early in a project as non-functional but, as the project progresses, will be translated into requirements for software functionality that can be measured the same as any other FUR.

2.7 Skills needed for COSMIC Measurers

To use the COSMIC method to measure sizes accurately, the skills needed are those of any requirements engineer or systems analyst. To use the results of COSMIC size measurements for project performance comparisons, development of benchmarks and estimating, it is highly desirable to have a basic knowledge of statistical methods.

3

A BRIEF HISTORY OF FUNCTIONAL SIZE MEASUREMENT

This chapter describes why and how the COSMIC method was developed.

3.1 How did it all start?

In the mid 1970's, Allan Albrecht of IBM was tasked with measuring the productivity of software projects in a part of IBM that was starting to use multiple programming languages. Given the disadvantages of using SLOC as a measure of the size of delivered software, he had the clever idea of developing a size of the software requirements which would be independent of the technology used.

Albrecht's method was first published in 1979 (see reference [3]) and became known as 'Function Point Analysis - FPA'. Management of the development of the method was taken over by the International Function Points User Group and the method has become known as 'IFPUG FPA'.

Although the IFPUG method is probably still the most widely used FSM method in the domain of business application software, the method has several weaknesses, of which the following are the most important:

- It has become increasingly difficult to map Albrecht's function types to modern ways of modeling software requirements. This applies especially to areas where software is constructed as services, and in the domains of real-time and infrastructure software.
- The function types that it considers can be given only a very restricted range of sizes which means the method is insensitive to the extremes of size that exist in real software. A measurement scale should normally be linear and open-ended.

3.2 The International Organization for Standardization (ISO) steps in

By around 1990, there was demand for an ISO standard FSM method. But there was no agreement that any of the then-existing methods (the IFPUG method and others) were suitable candidates. ISO therefore established a working group[2] to study and define the principles of FSM. A first version of the resulting standard, ISO 14143/1 (see reference [4]), was published in 1998.

The new principles helped improve the understanding of FSM, but did not solve the problem of dissatisfaction with existing methods. The market needed a new FSM method.

3.3 COSMIC gets going

ISO procedures are designed to obtain agreement on standards from existing knowledge, but not for developing new ideas. An informal group of software measurement experts from Australia, Europe and North America, therefore decided in late 1998 to embark on developing a 'second generation' FSM method based on the principles of ISO 14143/1. The group called itself 'COSMIC', the Common Software Measurement International Consortium.

[2] ISO JTC1/SC7/WG12

COSMIC's objectives
The COSMIC group's objectives were to develop and gain market acceptance for a method of measuring the functional user requirements for software based on fundamental software engineering principles and conformant to measurement theory, to be applicable for measuring business, real-time and infrastructure software.

COSMIC is still an entirely voluntary, international group of software measurement experts, from industry and academia.

COSMIC continues to refine the definition and explanation of the method in light of practical experience, though it must be emphasized that ***the basic principles of the size measurement have not changed since the method was first published in 1999***.

3.4 ISO's final word: 'Let the market decide'

In the early 2000's at the ISO level, there was still no international standard FSM Method and no agreement that any existing method could be accepted as such. Finally therefore, ISO agreed a policy of 'let the market decide'. So there are now five ISO FSM standard methods (IFPUG, COSMIC and three others) for you to choose from.

The COSMIC method ISO standard (ISO 19761) was first published in early 2003. The latest 2011 version of this standard can be obtained from www.iso.org**.**

4

A VERY BRIEF OVERVIEW OF THE COSMIC METHOD

The aim of this chapter is to give a first, very high-level overview of the COSMIC method.

The first use of COSMIC method keywords in any chapter is given in **bold**. For the formal definition of keywords, see the glossary of the Measurement Manual [5].

4.1 Applicability of the method

The COSMIC method was designed to measure the **Functional User Requirements** (FUR) of business application (or 'management information system'), real-time and infrastructure software and some types of scientific/engineering software, in any layer of a software architecture, and at any level of decomposition.

4.2 The three phases of the COSMIC functional size measurement process

The COSMIC measurement process is shown in Figure 4.1. The three phases are explained in the next sections.

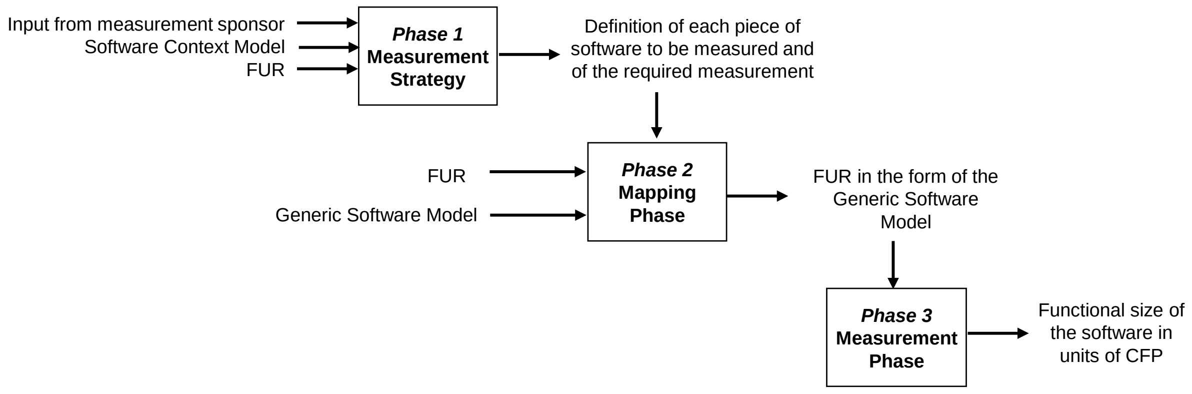

Input from measurement sponsor
Software Context Model
FUR
Phase 1
Measurement Strategy
Definition of each piece of software to be measured and of the required measurement
FUR
Generic Software Model
Phase 2
Mapping Phase
FUR in the form of the Generic Software Model
Phase 3
Measurement Phase
Functional size of the software in units of CFP

Figure 4.1 - The COSMIC measurement process

4.3 Phase 1: Measurement Strategy

We must first define what will be measured. The size of a piece of software depends on the viewpoint of who or what we define as its **functional users**, i.e. the humans, hardware devices or other software that interact with the software. In order to measure the size of the piece of software, we must therefore first agree the **purpose** of the measurement, which leads to defining its **scope** (the extent of the software's FUR to be measured) and its functional users**,** and then usually some other parameters[3].

It's essential to document the parameters of the measurement strategy so that the resulting measurement(s) will be correctly interpreted by all future users.

[3] The principles underlying the parameters needed for the measurement strategy are all defined in the COSMIC 'Software Context Model'. This model is not described in this Introduction. See the Measurement Manual.

4.4 Phase 2: Mapping

The task of the Mapping phase is to create the COSMIC model of the FUR, starting from whatever artefacts of the software are available, e.g. an outline or detailed statement of requirements, design models, the installed physical software, etc. To create the model, we apply the principles of the COSMIC **Generic Software Model** to the FUR to be measured.

This model of the FUR of software rests on four main principles:

1. Software functionality consists of **functional processes.** The task of each functional process is to respond to an **event** that has happened in the world of the software's **functional users**.
2. Functional processes consist of sub-processes. These do only two things: they move and they manipulate data. **Data movement** sub-processes that move data from functional users into a functional process and that move data out to them are called **Entries** and **Exits** respectively. Data movement sub-processes that move data to and from **persistent storage** are called **Writes** and **Reads** respectively. Figure 4.2 illustrates the four types of data movements.

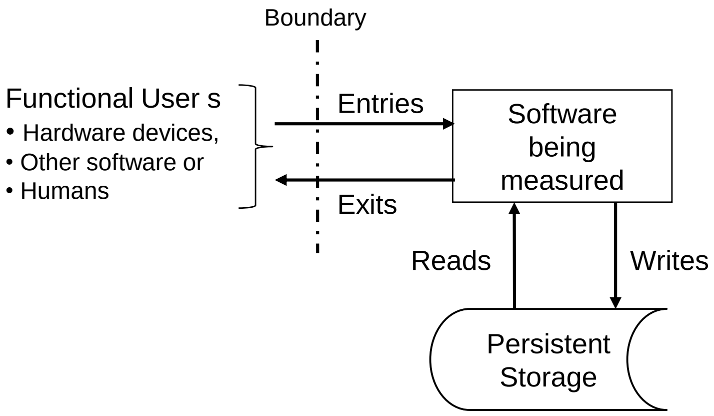


Figure 4.2 - The four types of data movements

3. Each **data movement** (Entry, Exit, Read or Write) moves a single **data Group** whose a**ttributes**[4] describe a single 'thing' (an o**bject of interest**).
4. Data manipulation sub-processes are assumed to be accounted for by the data movement with which they are associated. Data manipulation is not measured separately.

A functional process finishes executing when it has done all that it is required to do to respond to the data it received about the event.

4.5 Phase 3: Measurement

The COSMIC method measurement unit is the 'Cosmic Function Point' (CFP). Each data movement is measured as 1 CFP.

In the Measurement phase, we measure the size of a new piece of software by identifying all the data movements (Entries, Exits, Reads and Writes) of each functional process and sum these over all its functional processes.

A functional process must have at least two data movements (an Entry plus either an Exit or a Write) in order to provide a minimal but complete service. Hence the minimum size of a functional process is 2 CFP. There is no upper limit to the size of a functional process

To measure an enhancement to existing software, we identify all the data movements to be added, changed and deleted, and sum these over all its functional processes. The minimum size of any modification to a functional process is 1 CFP.

[4] Known as 'Data Element Types' or 'DETs' in some other FSM Methods.

5

COSMIC METHOD - THE MEASUREMENT STRATEGY PHASE

A COSMIC functional size measurement should follow a three phase process. In the first Measurement Strategy phase, the Measurer must agree with whoever needs the measurement (the 'sponsor') the purpose of the measurement and usually some other parameters that all depend on the purpose.

In this and subsequent chapters, the first use of a COSMIC keyword in each chapter is given in **bold**. Formal definitions of the keywords are given in the glossary of the Measurement Manual ('MM') which gives all the principles and rules of the COSMIC method, with many examples [5]. This Introduction only has informal definitions.

5.1 Why do we need a 'strategy'?

You need to agree and document the purpose of the measurement and various other parameters with the measurement sponsor so that in the future everyone will understand the measured size and how it may be used.

In practice, you will find that only a few recurrent 'patterns' of parameters will be needed for the different types of software you will have to measure in your organization. To help you, a 'COSMIC Guideline for Measurement Strategy Patterns' [6] defines some of the commonest patterns and their uses.

5.2 The five key strategy parameters to be determined

- The **purpose** of the measurement. The purpose helps determine all the following parameters.
- The **scope** of the piece(s) of software to be measured. A project might have to deliver several pieces of software, or the functionality to be measured might be restricted in some way. What's included in the functionality and what's excluded?
- The **level of decomposition** of the piece(s) of software to be measured. Different levels would be, for example, a 'whole application' ('level 0'), or one of the primary components of a distributed system ('level 1'), or a re-usable component in a SOA architecture ('level 2').
- The **functional users** of each piece of software to be measured. These are the humans or 'things' (hardware devices or other pieces of software) that are the intended senders or recipients of data to/from the software being measured. It is the functionality they 'see', that you will measure;
- The **layer**(s) of the software architecture in which the software resides. A piece of software to be measured must be confined to one layer.

By documenting these parameters for each measured size, you will help ensure that in the future the sizes will only be compared and used on a 'like-for-like' basis.

5.3 Software 'layers'

Most of the measurement strategy parameters are easy to understand. But the term 'layer' is used in various ways in the software industry. {Sometimes 'n-tier' is used instead of 'n-layer'.) Figure 5.1 shows a typical computer systems 'layered architecture' that supports business application software.

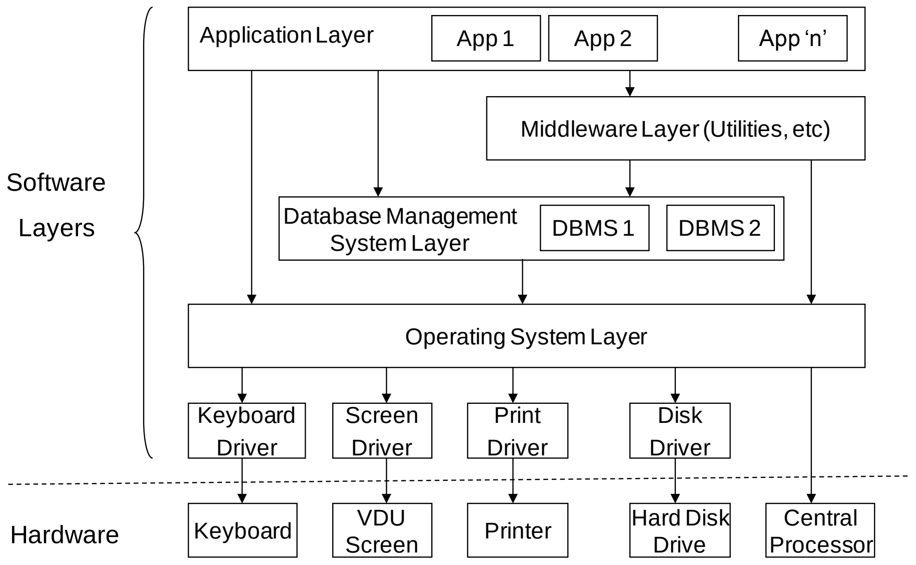


Figure 5.1 - Typical layered software architecture for a business/MIS computer system

Figure 5.2 shows that the Application Layer in Figure 5.1 may be sub-divided into other layers, dependent on the 'view' of the software architect (and consequently of the functional users of the software to be measured, as we shall see).

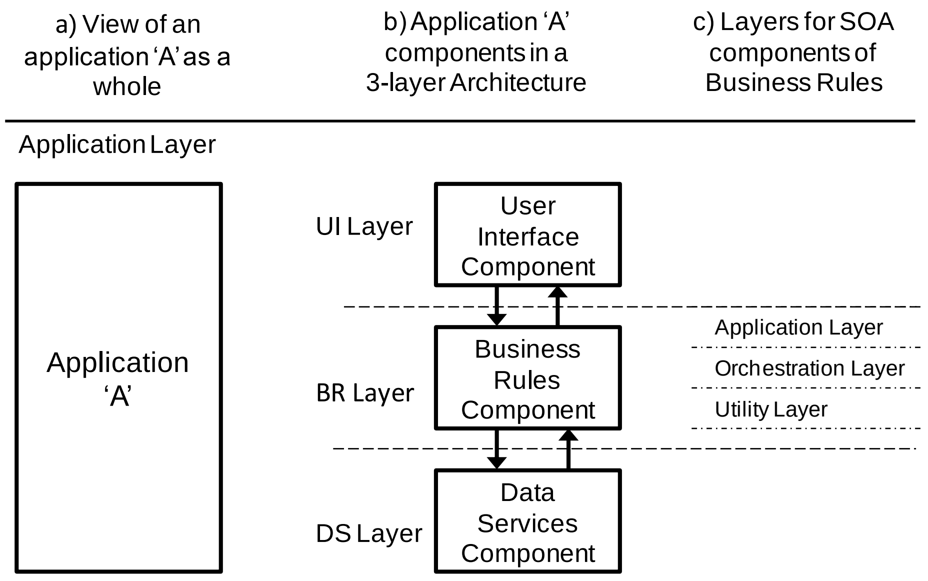


Figure 5.2 - Three views of a piece of application software

5.4 Examples of how the 'purpose' of a measurement affects the other measurement strategy parameters

BUSINESS EXAMPLE: Suppose the software to be developed and measured is a distributed 3-layer business application system. The context is a contract with a supplier that stipulates that for payment purposes, software sizes will be measured at the level of whole applications, ignoring any component structure.

Case 1.

Purpose: to measure the size of a delivered application for contract payment

Scope: The FUR of the one application

Functional users: Human users and any other interfacing applications

Level of decomposition: None ('level 0')

Layer: Application, i.e. view a) as in Figure 5.2

Case 2.

Purpose: to measure the size of each major component of the distributed application so that the supplier can estimate the project effort, because each component will be developed using a different technology.

Scope: Each component is measured separately (i.e. there are three measurement scopes).

Functional Users: See the three layers as in Figure 5.2 View b)

The User Interface component has the Human users and the Business Rules component as its functional users

The Business Rules component has the User Interface and the Data Services components as its functional users

The Data Services component has the Business Rules component and any other interfacing applications as its functional users.

Level of Decomposition: First-level decomposition of an application ('level 1')

Layers: See the three layers as in Figure 5.2 View b)

REAL-TIME EXAMPLE: The functionality of the software embedded in a hardware device used by humans, for example a combined computer printer/copier can be measured from the viewpoints of two types of functional users. (In both cases assume that we are not interested in any component structure of the software nor any firmware that the embedded software may use.)

Case 1.

Purpose: to measure the size of functionality available to the human user (the 'consumer offering' for Marketing), so as to compare against the offering of competitive products.

Scope: Functionality available to human operator users (i.e. excluding functionality needed by operators over which they have no control or cannot 'see', such as some functions needed to communicate with a computer)

Functional Users: Human operator users

Case 2.

Purpose: to measure the functionality that the embedded software developer must provide for the device to function

Scope: All the functionality of the embedded software

Functional users: All hardware devices with which the software must interact (e.g. keyboard, control buttons, screens, print drive mechanism, paper transport mechanism, etc and computer printer driver software that the printer must communicate with).

5.5 What else should you think about before starting to measure?

It's very important to agree what artefacts of the software are available that can be used to determine the FUR to be measured. In practice the available artefacts may not supply exactly the information needed for any FSM measurement, so the Measurer usually has to make some assumptions when deriving the FUR. It is best to consult an expert in the requirements of the software to be measured to help with understanding the software so that the measurement is as accurate as possible.

Some examples of the problems typically faced:

- If a size measurement is needed early in the life of a project, the requirements may not yet have been documented in the detail needed for an accurate COSMIC measurement. For these situations, we have a Guideline [7] that describes variants of the standard COSMIC method that can be used to measure an approximate size;
- Sometimes software requirements are defined at a 'high-level' and then defined at increasingly detailed 'lower' levels. We call these **levels of granularity**. To ensure comparability, sizes must be measured at the standard level of granularity of 'functional processes' (see further below). If necessary, a variant for approximate size measurement can be used to scale from a size measured at a higher level of granularity to the standard level;
- Sometimes a size must be measured of an installed system for which the requirements no longer exist. In these situations, the Measurer will need to 'reverse engineer' from the available artefacts, e.g. screens, user documentation, reports, user interfaces, etc. to determine the FUR.

Several COSMIC Guidelines describe how to derive or analyze FUR for different types of software or development methods. They are all available from www.cosmicon.com.

Finally, a Measurer may be asked to estimate how long it will take to measure a particular piece of software. The average speed of measurement using the COSMIC method is similar to that for other standard FSM methods. But the actual rate can vary about this average greatly. The effort needed for a measurement will tend to increase:

- the worse the quality of the artefacts available for measurement;
- the greater the accuracy of the measurement required by the measurement sponsor and the level of detail of the measurement to be documented;
- the less-experienced the Measurer is in the type of software to be measured and in the COSMIC method.

6

COSMIC METHOD - THE MAPPING PHASE

The task of the Mapping phase is to produce a model of the **Functional User Requirements** (**FUR**) of the software to be measured from its available artefacts using the principles of the COSMIC **'Generic Software Mode**l'. We first state these principles, then describe the elements of the model in more detail and finally deal with the mapping process.

6.1 The Generic Software Model

PRINCIPLES – The COSMIC Generic Software Model	
a)	A piece of software interacts with its functional users across a **boundary**, and with **persistent storage** within this boundary
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)	There are four data movement types, **Entry**, **Exit**, **Write** and **Read**. An Entry moves a data group into a functional process from a functional user. An Exit moves a data group out of a functional process to a functional user. A Write moves a data group from a functional process to persistent storage. A Read moves a data group from persistent storage to a functional process
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)	Each functional process is started by its **triggering Entry** data movement. The data group moved by the triggering Entry is generated by a functional user in response to a **triggering event**
i)	A functional process shall include at least one Entry data movement and either a Write or an Exit data movement, i.e. it shall include a minimum of two data movements. There is no upper limit to the number of data movements in a functional process
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

Important remark: ALL of the COSMIC keywords in the above principles (except 'persistent storage' should really end in '-type'. For example 'sub-processes' should really be written as 'sub-process types' and 'Entry' as 'Entry-type'. ALL FSM methods aim to identify 'types' and not 'occurrences' of functions or data. However, we omit '-type' from all these keywords for ease of reading, unless we need to distinguish 'types' and 'occurrences'.

Principle a) simply summarizes all that software does. The other principles are explained in the following sections of this chapter.

6.2 A key relationship: events / functional users / functional processes

Principles b) and h) tell us that the task of software is to respond to events that occur in the world of its functional users. A functional user informs software that an event has occurred and may send data about the event. The software must do something useful for the functional user(s) that have an interest in the response to that event. We call this 'something useful' a 'functional process'. All software FUR can be expressed in terms of functional processes.

The relationship between events in the world of the functional user and functional processes of the software is shown in Figure 6.1. (The **boundary** is the interface between the software being measured and its functional user(s).)

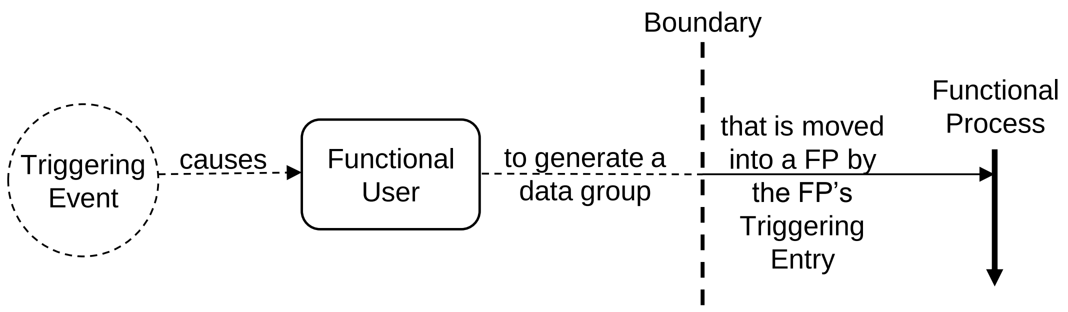


Figure 6.1 - The relationship between events, functional users and functional processes

The general interpretation of this diagram is that *an event causes a functional user to generate a message (a data group) that is moved by a 'triggering Entry' into its functional process, thus starting the functional process.* (Note, however, when a human functional user decides to make an enquiry on existing data, the human user effectively *generates* the event and then the message.)

An event is 'something that happens'. A triggering event has either happened or not happened; it cannot be sub-divided for the FUR of the software to be measured.

EXAMPLE: Suppose a soccer (football) match. The FUR of three different software applications could have quite different views of the events that happen at the match.

Application A allows reporters to enter the results of football matches for a newspaper. The only event that the FUR recognizes is 'match finished'.

Application B is a 'live reporting' system that enables a reporter to enter comments that are transmitted over the web to on-line users of the application about anything the reporter considers significant that happens during the match, e.g. kick-off, a goal scored, foul, injury, etc. The only event that the FUR recognizes is 'anything that happens about which the reporter enters a comment to Application B'.

Application C allows real-time monitoring of the performance of the players. Each player carries a GPS position-sensing device and a heart-beat monitor, which transmits data at regular very short intervals. The only event that the FUR recognizes is a 'tick' of the clock that controls the transmission of data on the current position and heart rate of each player at each 'tick' to the application C.

(Remember that for all these FUR we should really be writing 'event-type'. Each of the three FUR recognizes only one event-type but it is a different event-type for each FUR. But in terms of event-occurrences, App A expects one occurrence for each match, App B maybe several tens per match, and App C several tens of thousands per match.)

Note that Figure 6.1 says nothing about the *degree* (or 'cardinality') of the relationships between the various concepts. For example a single event might be detected by multiple functional users of the same or different pieces of software (e.g. an earthquake detected by multiple sensors); a functional user of one piece of software may detect many types of events (e.g. humans interacting with software).

The only rule that applies to the cardinality of the relationships along the chain of Figure 6.1 (and it is a very important rule) is that: '*any one triggering Entry of a piece of software being measured may initiate only one functional process in that software'.*

6.3 The structure of FUR and of Functional Processes

Principles b), c) and d), which describe the theoretical structure of FUR, i.e. their decomposition into functional processes and sub-processes, is illustrated in the left-hand part of Figure 6.2.

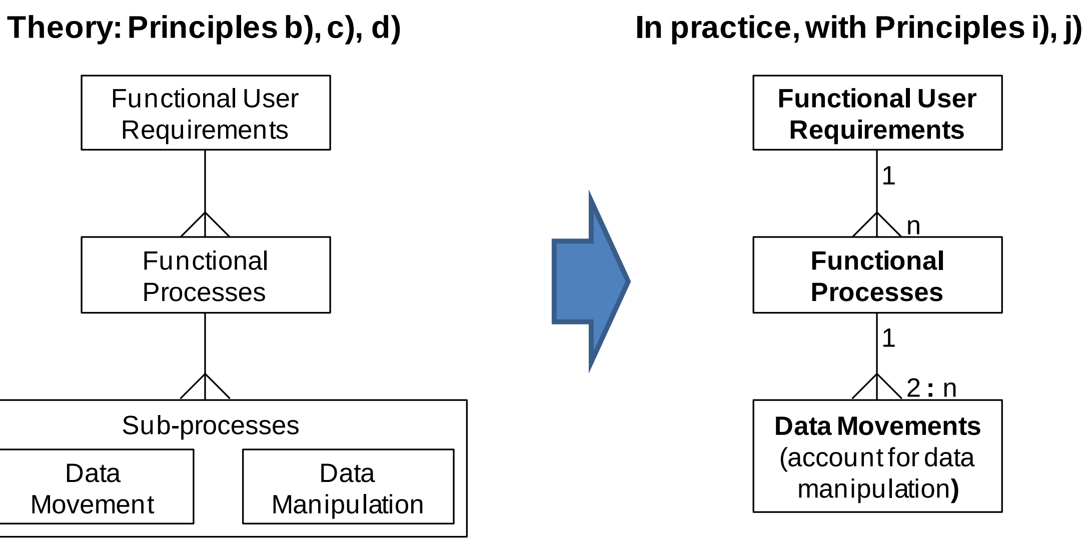

Theory: Principles b), c), d)
Functional User Requirements
Functional Processes
Sub-processes
Data Movement
Data Manipulation
In practice, with Principles i), j)
Functional User Requirements
1
n
Functional Processes
1
2 : n
Data Movements (account for data manipulation)

Figure 6.2 - The structure of Functional User Requirements

[The 'crow's foot' symbol shows the permitted degree of the relationship between two adjacent concepts. Principle i) is expressed by showing that one functional process can have from 2 (minimum) up to 'n' data movements.]

Correctly identifying functional processes is the most important step of the Mapping phase. So you really must understand its full definition.

DEFINITION – Functional process
a) A set of data movements representing an elementary part of the Functional User Requirements for the software being measured, that is unique within that FUR and that can be defined independently of any other functional process in that FUR. b) A functional process may have only one triggering Entry. Each functional process starts processing on receipt of a data group moved by the triggering Entry data movement of the functional process. c) The set of all data movements of a functional process is the set that is needed to meet its FUR for all the possible responses to its triggering Entry. NOTE 1: When implemented, it is an *occurrence* of a functional process that starts *executing* on receipt of an *occurrence* of a data group moved by an *occurrence* of a triggering Entry. NOTE 2: The FUR for a functional process may require one or more other Entries in addition to the triggering Entry. NOTE 3: If a functional user sends a data group with errors, e.g. because a sensor-user is mal-functioning or an order entered by a human has errors, it is usually the task of the functional process to determine if the event really occurred and/or if the entered data are really valid, and how to respond.

6.4 Accounting for data manipulation

The COSMIC method does not measure data manipulation explicitly because there is no generally-accepted way of measuring data manipulation so that it can be combined with a measure of data movements to produce a usable measure of functional size. We therefore invoke principle j) to assume that each data movement can account for any associated data manipulation, as shown in the right-hand part of Figure 6.2. This assumption has proven to be reasonable for all the practical purposes such as project performance measurement and estimating for which the method was designed and for the domains in which it is commonly used. Where the method cannot account adequately for data manipulation, the COSMIC measurement method has provision for local extensions to the method to overcome the limitation.

6.5 The four types of data movements

Principle e) is illustrated in Figure 6.3.

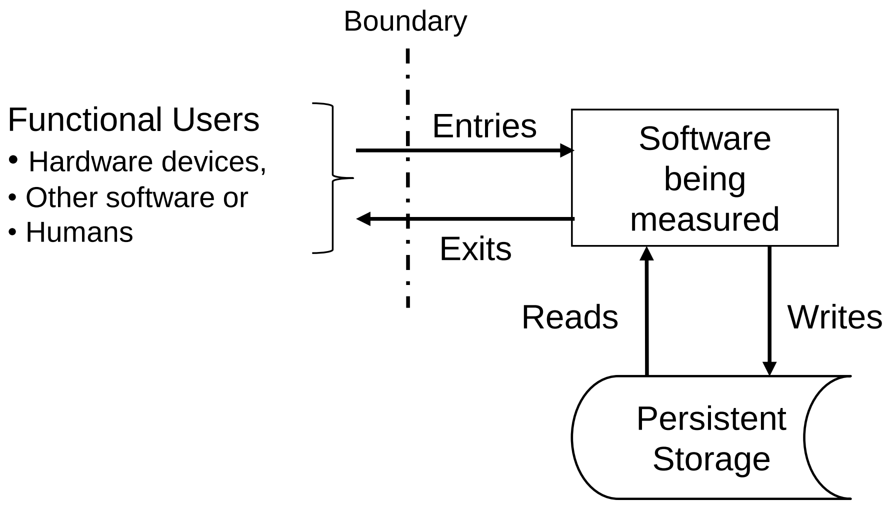


Figure 6.3 - The four types of data movements

Entries and **Exits** move data in and out of the software from/to functional users respectively.

Reads and **Writes** move data from **persistent storage** to the software, or vice versa, respectively.

6.6 Persistent storage

Persistent storage (shown in Figure 6.3) is an abstract concept of the Generic Software Model. In the model, such storage is accessible by any software in any layer if it needs to store data or to retrieve stored data. After a functional process has written some data on persistent storage, that ‘persistent data’, is available to other functional processes that need it or to another occurrence of the functional process that wrote it.

A consequence of this concept is that if you are measuring, say, an application that must store data or retrieve stored data, you do not have to think about how that data is physically processed by software in lower layers or by the hardware. Just represent FUR that require data to be stored or retrieved by Writes and Reads respectively.

You only need to think of persistent storage in terms of physical disks or memory if you must measure software for which physical hardware devices (storage or other) have been defined in the Measurement Strategy phase as functional users of the software. Functional users always interact with the software to be measured via Entries and Exits.

6.7 A data movement moves a single data group describing one object of Interest

So how do we distinguish one Entry data movement from another Entry data movement, and similarly for Exits, Reads and Writes?

An **object of interest** is any 'thing' (physical or conceptual) about which the software being measured must process or store data. A **data movement** moves a single **data group** that consists of one or more **data attributes** (known as 'Data Element Types' in other FSM methods). All attributes in a data group describe the same one object of interest.

It is not necessary to identify the data attributes for measurement purposes. The only reason to mention them is that it sometimes helps to distinguish different data groups and their objects of interest by examining the data attributes.

Figure 6.4 shows the relationships between these three concepts, with two examples.

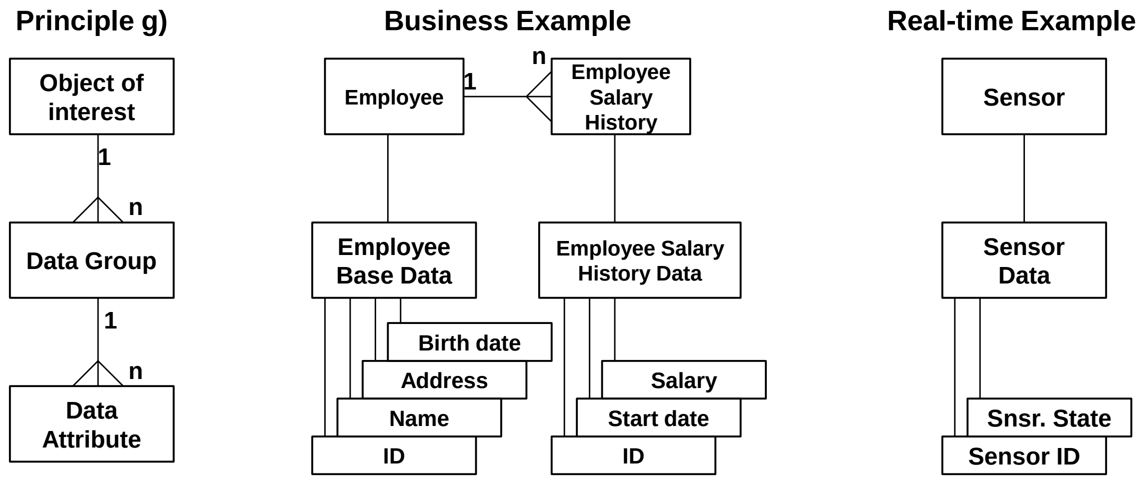


Figure 6.4 - Relationships of an object of interest, data groups and data attributes

To help you understand these concepts:

- If you are familiar with data analysis methods, often used in the domain of business applications, then entity-types found in Entity-Relationship Analysis, and the subjects of relations in 3rd normal form found in Relational Data Analysis will be objects of interest. But these analysis methods are usually only applied to the structure of *stored* (or 'persistent') data groups. For a COSMIC measurement, you will also need to apply these same analysis ideas to distinguish objects of interest and hence the data movements in the input and output of functional processes. Each Entry and Exit moves a *transient* data group describing one object of interest.
- There is a one-to-one relationship between objects of interest and the object classes resulting from UML analysis, though of course they are not the same concept.
- In the domain of real-time software, it is NOT usually necessary to think about objects of interest. Often, perhaps as shown in the real-time example of Figure 6.4, the functional user - the sensor - can be seen as sending data about itself, i.e. the functional user plays the part of object of interest, so it will have been identified earlier in the Measurement Strategy phase. (There is no point in making all these distinctions unless doing so helps the measurement process.)

6.8 The Mapping phase process

Assuming the available artefacts of the software to be measured are at the functional process level of granularity, we examine them to derive the FUR expressed as a COSMIC model. The steps of this process (remembering that we always refer to *types*) are:

- Identify the separate events in the world of the functional users that the software must respond to, i.e. the ‘triggering events’
- Identify which functional user(s) of the software must respond to each triggering event by generating a data group that is moved by a triggering Entry
- Identify one functional process for each triggering Entry
- Identify any other Entries and all the Exits, Reads and Writes of each functional process needed to meet the FUR for all possible responses to the triggering Entry.

For the last step you may need to identify the data groups that are moved in each data movement and the objects of interest that the data describe.

6.9 Some simple examples of Mapping

We can now analyze some simple examples to map from outline statements of requirements to the COSMIC functional processes and data movements using the Generic Software Model.

BUSINESS EXAMPLE: A simple Personnel System

Outline statement of requirements. A system is required to enable personnel staff to hold and maintain data about employees, including their salary and the history of their salary progression over time. [The statement also describes the data (attributes) to be recorded about each employee and their validation criteria, but most of this detail need not concern the Measurer in this simple example.] A report is required each month listing all employees by name and their current salary, the total number of employees, and the total current salary cost.

Measurement strategy parameters

Measurement purpose: An accurate functional size measurement of the Personnel System for project effort estimating.

Measurement scope: The whole system as specified in the statement of requirements.

Functional users: Personnel staff.

Layer: Application layer.

Level of decomposition: ‘level 0’, i.e. no decomposition.

Mapping phase

Some assumptions:

- *The data structure of the Business Example in figure 6.4 applies.*
- *Each employee will be allocated a unique ID by a member of personnel. The key of an ‘employee salary history’ record is [employee ID, salary start date].*
- *The word ‘maintain’ in the outline statement of requirements normally implies that there must be Create, Read, Update and Delete functional processes (remember the ‘CRUD’ acronym?) for each object of interest. ‘Update’ will enable a change of any attribute except the key attribute(s) of any data group.*
- *There are two objects of interest (‘employee’ and ‘employee salary history’) about which persistent data must be held. We will need the four ‘CRUD’ functional processes to maintain the employee base data.*
- *We also assume that an employee salary history record must be created when an employee first starts work. Subsequently, an employee’s salary may be updated either when the employee base data must be updated, or separately from an employee base data update. So there is no need for separate ‘create’, or ‘delete’ functional processes for the employee salary history, but there is a need for an ‘employee salary update’ functional process and for a separate ‘read’ functional process for enquiries on employee salary history. Adding in the*

process to produce the monthly report means the requirements can be satisfied by 7 functional processes. (We show below the analysis of four of these.)

- *In practical situations there also may be FUR for a ‘read’ functional process to display the employee’s base data separately from the enquiry to display the employee’s salary history. Also, when an employee leaves, the ‘delete’ functional process may be required to archive the employee base and salary history data, rather than actually delete it. We ignore these possible requirements for simplicity.*
- *Note also as a general rule when measuring on-line business applications, that ‘menus’ that only assist navigation and the selection of functional processes, and ‘blank’ data entry screens should be ignored. It is the movement of data by Entries, Exits, Reads and Writes that must be identified for measurement purposes.*

1. Analysis of functional process ‘Create employee’.

The FUR is to enter data for a new employee.

(The examples show the data movements and the data group that each of them moves).

Functional process ‘Create employee’. Triggering event: a new person is employed

Triggering Entry	:	*Employee base data*
Entry	:	*Initial salary and its start date*
Read	:	*Employee base data (to check that no employee already exists with the entered ID)*
Write	:	*Employee base data*
Write	:	*Employee salary history (a new record is created when the salary is first entered)*
Exit	:	*Error/confirmation messages (There must be error messages for various validation failures and also some form of confirmation on successful data entry. We include one Exit to account for all such messages.)*

2. Analysis of ‘Update employee data’ process, including possible salary update

We assume a user will first wish to retrieve and display the employee’s base data, before entering a change to one or more attributes, including maybe a new salary. This procedure will require two functional processes. The first is triggered by the event of the user deciding to display the existing data; it is the ‘read employee base data’ functional process. The second is triggered by the event that one or more attribute(s) of the employee have changed in the real world; it is the ‘update employee base data’ functional process. The two functional processes are:

Functional process ‘Read employee data’. Triggering event: Decision to display existing data

Triggering Entry	:	*Employee ID*
Read	:	*Employee base data*
Read	:	*Employee salary history*
Exit	:	*Employee base data*
Exit	:	*Employee salary history*
Exit	:	*Error/confirmation messages (in case a non-existent ID was entered)*

Functional process 'Update employee data'. Triggering event: Employee data changed has changed in some way

Triggering Entry	*: Updated employee base data (for the update of one or more attribute(s))*
Entry	*: Updated salary and its start date*
Write	*: Employee base data (the updated record)*
Write	*: Employee salary history (a new record is created if the salary has been updated)*
Exit	*: Error/confirmation messages (for entry of invalid data or the possible failure of the update)*

3. Analysis of the process to produce the monthly report for the payroll department

Functional process 'End-of-month employee' report. *Triggering event: The end of the month*

Triggering Entry	*: End of month time signal (every functional process must have a triggering Entry, even though this one conveys no variable data)*
Read	*: Employee base data (to get employee ID's and names)*
Read	*: Employee salary history (to obtain the current salary)*
Exit	*: Employee current salary (one line for each employee with their ID, name and salary*
Exit	*: End-of-month employee totals (of numbers of employees and of their total salary)*

NOTES: the final Exit moves a data group describing the object of interest 'All employees'. No data is stored about this object of interest, so the data group is transient; but the object of interest is a group of real people, i.e. a real 'thing' in the world of the functional user.

We have not counted an error message for this functional process as there does not seem to be any reason for the application to have to generate such a message. (The operating system might generate an error message if the data cannot be found, but this is not part of the application.)

REAL-TIME EXAMPLE: A simple domestic alarm system

Outline statement of requirements. (We deduce the functionality available to normal house occupants from knowing how to use the system and by examining it physically. We are not interested in the functionality provided for the alarm maintenance engineer).

The software supports the alarm's human interface via a keypad and red/green LED's. The software also accepts data from a device that senses whether the main front door of the house is open or not, and from several internal movement detectors. It also controls an internal and an external alarm. There is a battery to provide continuity if the electricity power supply fails, so there must be a power voltage detector. Finally, the house occupant must enter a PIN (Personal Identification Number) to activate and de-activate the alarm. This is stored by the software and can be changed, so there must be some persistent storage. After 20 minutes the external alarm must stop (a legal requirement).

There must be a clock mechanism, since certain functions must be completed within pre-set times. For example, if the system is activated before leaving the house, the front door must be closed within 30 seconds; if not, alarms will sound. We do not know how the clock is implemented but assume a software implementation for simplicity, which starts whenever needed. The functionality to keep track of elapsed times is then a form of data manipulation, which we can ignore.

Measurement strategy parameters

Purpose of the measurement: To list the functional processes of the embedded software, excluding those known only to the maintenance engineer.

Measurement scope: The alarm embedded application software. (We are not interested if there is an operating system)

Functional users: A context diagram shows how the functional users interact with the software. Note that the movement detectors are all functionally identical, so do not need to be distinguished.

Layer: Application.

Level of decomposition: 'level 0', i.e. no decomposition.

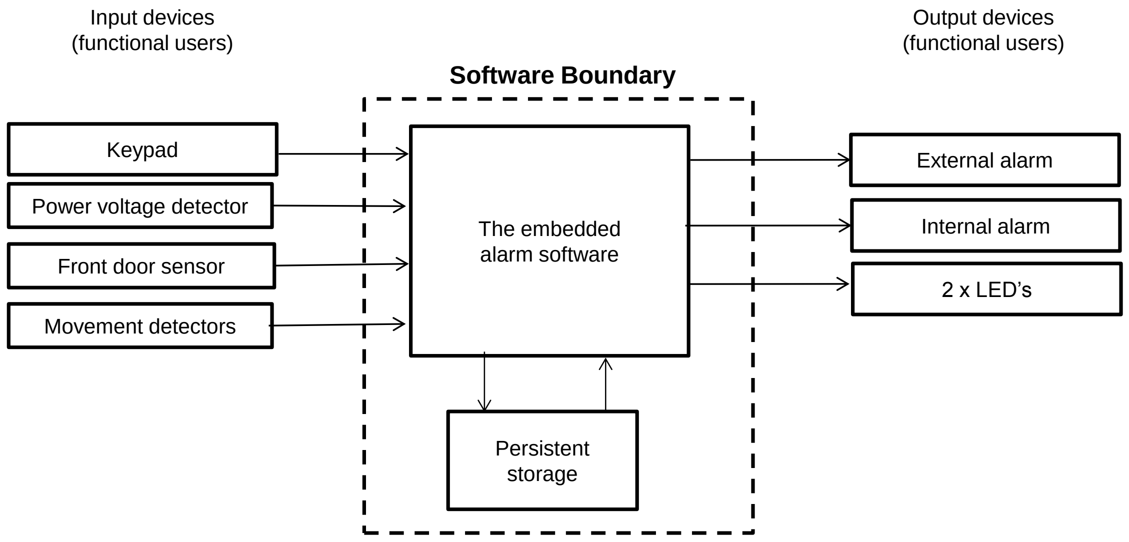


Figure 6.5 – The Intruder Alarm System

The Functional Processes*: The alarm application has nine functional processes. These can be easily identified by considering the events that the software must respond to.*

- *A new, or a changed, PIN is entered via the keypad (2 events)*
- *The user enters the current PIN via the keypad before leaving and closing the front door within a pre-set time, which activates the system*
- *The front door sensor detects that the door opens whilst the alarm system is activated.*
- *The system is activated, or de-activated, by the occupants whilst they are in the house (2 events), e.g. when retiring at night, out of range of the movement detectors*
- *A movement detector signals a movement whilst the alarm system is activated*
- *The power voltage detector signals electrical failure, or its restoration (2 events)*

Analysis of an example functional process:

We analyze the third event on the list above (the front door is opened whilst the alarm is activated). When the front door sensor detects this event, the internal alarm starts; the PIN code must then be entered within a pre-set time to de-activate the system and to stop the internal alarm. If the PIN code isn't entered before the pre-set time, or the wrong code is entered more than three times, the external alarm also starts. The functional process has the following 8 data movements.

Functional process: Possible intruder detected. Triggering event: Door opens whilst alarm system is activated.

Triggering Entry : 'Door open' message from the front door sensor when the system is activated

Read	:	*Obtain PIN from persistent storage*
Exit	:	*Message to start the internal alarm*
Entry	:	*PIN code entered (If the wrong code is entered, the user may enter the PIN two more times but the process is always the same. PIN validation is data manipulation, so is accounted for by this Entry)*
Exit	:	*Message to switch red LED from 'on' to 'off'*
Exit	:	*Message to switch green LED from 'off' to 'on'*
Exit	:	*Message to stop internal alarm (on successful entry of PIN)*
Exit	:	*Message to start external alarm (after three unsuccessful PIN entries, or if the PIN is not entered in time)*
Exit	:	*Message to stop the external alarm (after 20 minutes, a legal requirement)*

This functional process therefore has 2 x Entries, 1 x Read and 6 Exits. Its total size is 9 CFP.

6.10 Some general lessons from these examples

- Implementation details are generally irrelevant to the Mapping Process. For example the functional processes of the personnel system could be implemented in many ways, all leading to the same data movements in the COSMIC model. Similarly, we do not need to know how the 'door open' message triggers the process of the Intruder Alarm system. (Either the software, when activated, could poll the front door sensor and the movement detectors to ascertain their status, or these sensors could periodically send their status to the software.)
- It helps understanding to write out the data movements of a functional process in roughly the sequence they would be executed. But the actual sequence will be more complicated in practice. For example the validation of the data entered in the personnel system could be interspersed with the issue of error message occurrences.
- The examples illustrate the part of the definition of a functional process that states that 'The set of all data movements of a functional process (which includes the triggering Entry) is the set that is needed to meet its FUR for all the possible responses to its triggering Entry'. In the personnel system process that updates an employee data, a new salary history record is created only if the employee's salary is changed. In the intruder alarm functional process, messages sent to the LED's and/or to the alarms will depend in each case on whether the human operator entered the correct PIN or not. The Measurer's only task is to identify all the data movements that are needed by a functional process to meet the FUR for all of its possible responses to all the data it may receive in its Entries and Reads. The Measurer does not have to worry about the sequence of the data movements, nor whether they are needed or not in any particular occurrence of the functional process which will depend on the data values entered.
- The set of data movements of a functional process is the set of types, not of occurrences.
- The intruder alarm case is an example where the object of interest of each data group entering or exiting the software is also the functional user that sent the group or that receives it (i.e. the functional user is sending or receiving data about itself). In these cases, having identified the functional users in the Measurement Strategy phase, the objects of interest have been identified as well.

COSMIC METHOD - THE MEASUREMENT PHASE

By the end of the Mapping phase, the Measurer will have produced a COSMIC model of the FUR of the piece of software to be measured (an instance of the Generic Software Model). We can then measure the functional size of the FUR of the software by applying the rules of the Measurement phase to this model.

7.1 The COSMIC measurement principle

The COSMIC measurement principle reflects the model shown in the right-hand part of Figure 6.2.

The COSMIC measurement principle
The functional size of a piece of software is equal to the number of its data movements

A functional size is measured in units of 'COSMIC Function Points', abbreviated as 'CFP'. 1 CFP is defined by convention as the size of a single data movement (Entry, Exit, Read or Write).

7.2 Size aggregation

Sizes can be measured at various levels of aggregation.

- The size of a functional process is equal to the number of its data movements
- The size of a piece of software is equal to the sum of the sizes of its functional processes
- The size of a piece of software can be derived from the size of its components provided the aggregation rules given in the Measurement Manual are followed

The following table shows a way of recording the results of the analysis of the four functional processes of the Personnel System analyzed in section 6.9, using the matrix given in Appendix A of the Measurement Manual [5].

	Data Group Names							Nos. of Data Movements				
Personnel System Functional Processes	Employee Base Data	Employee ID	Employee Salary History	Error/Confirmation Message	End of month 'clock tick'	Employee current salary	Employee totals	Entries	Exits	Reads	Writes	Total
Create Employee	E, R, W		E, W	X				2	1	1	2	6
Read Employee data	R, X	E	R, X	X				1	3	2		6
Update Employee data	E, W		E, W	X				2	1		2	5
End of month report	R		R		E	X	X	1	2	2		5
				Totals for Personnel System:				6	7	5	4	22

7.3 Size of required changes

The size of some required changes to an existing piece of software, e.g. as handled by an ‘enhancement’ project, are measured as follows:

- The size of a required change to a data movement (i.e. that must be added, modified or deleted) is measured by convention as 1 CFP. (‘Modified’ could mean any changes to the data manipulation associated with the data movement and/or to any of the attributes of the data group moved.)
- The minimum size of a change to a functional process is therefore 1 CFP.
- The size of all the required changes to a piece of software is equal to the number of data movements that must be added, modified or deleted, summed over all functional processes.

8

ADVANTAGES AND BENEFITS OF THE COSMIC METHOD

The COSMIC method of measuring a functional size of software from its requirements is the first such method:

- designed according to basic software engineering principles,
- to be applicable to software from the business application, real-time and infrastructure domains and some types of scientific/engineering software, in any layer of a software architecture, at any level of decomposition from a whole application down to its smallest components,
- for software developments or for enhancements, independently of all technology used and of the development methods,
- designed and maintained by an international group of software metrics experts,
- to conform to the ISO 14143/1 standard on the principles of functional size measurement,
- that is completely 'open' with all documentation available for free download from its web-site www.cosmicon.com,
- and that has been accepted as an International Standard (ISO 19761).

Compared with '1st generation' functional size measurement methods (see chapter 3), the COSMIC method:

- is completely stable due to its basic design principles which have not changed since the method was first published. This means that an organization's investment in existing measurements is safeguarded and that the method will be applicable to future software paradigms,
- has an open-ended measurement scale that is conformant with measurement theory. This means that all mathematical manipulations of COSMIC size measurements are valid,
- has required no calibration against effort and is thus a pure measure of functional size.

The COSMIC method is supported by:

- comprehensive documentation; the Measurement Manual has been translated into more than ten languages,
- guidelines that describe how to apply the method to specific types of software, e.g. data warehouse or SOA software, or for specific project management approaches, e.g. Agile methods,
- case studies and tools for collecting and reporting measurement data,
- comprehensive benchmark data available via www.isbsg.org. Many measurements of software projects in different domains have shown an excellent correlation of COSMIC-measured sizes with project effort.
- vendor services including suppliers of training, consultancy, estimating tools, etc,
- an Entry-level certification exam,
- guidelines for assuring the accuracy and the comparability of measurements
- documented variants for approximate COSMIC sizing that can be used early in the life of a project when all the details for the requirements have not yet been established, or for quick size measurement,
- documented methods of converting sizes measured using 1st generation FSM methods to COSMIC sizes using statistical correlations,
- active user groups on Linkedin ('COSMIC Users') and Twitter (@COSMIC_FSM),

- its web-site www.cosmicon.com and occasional newsletters.

The COSMIC method is being used successfully around the world for project performance measurement, for estimating, project scope control etc. The following mind map shows a range of possible uses of functional size measurements.

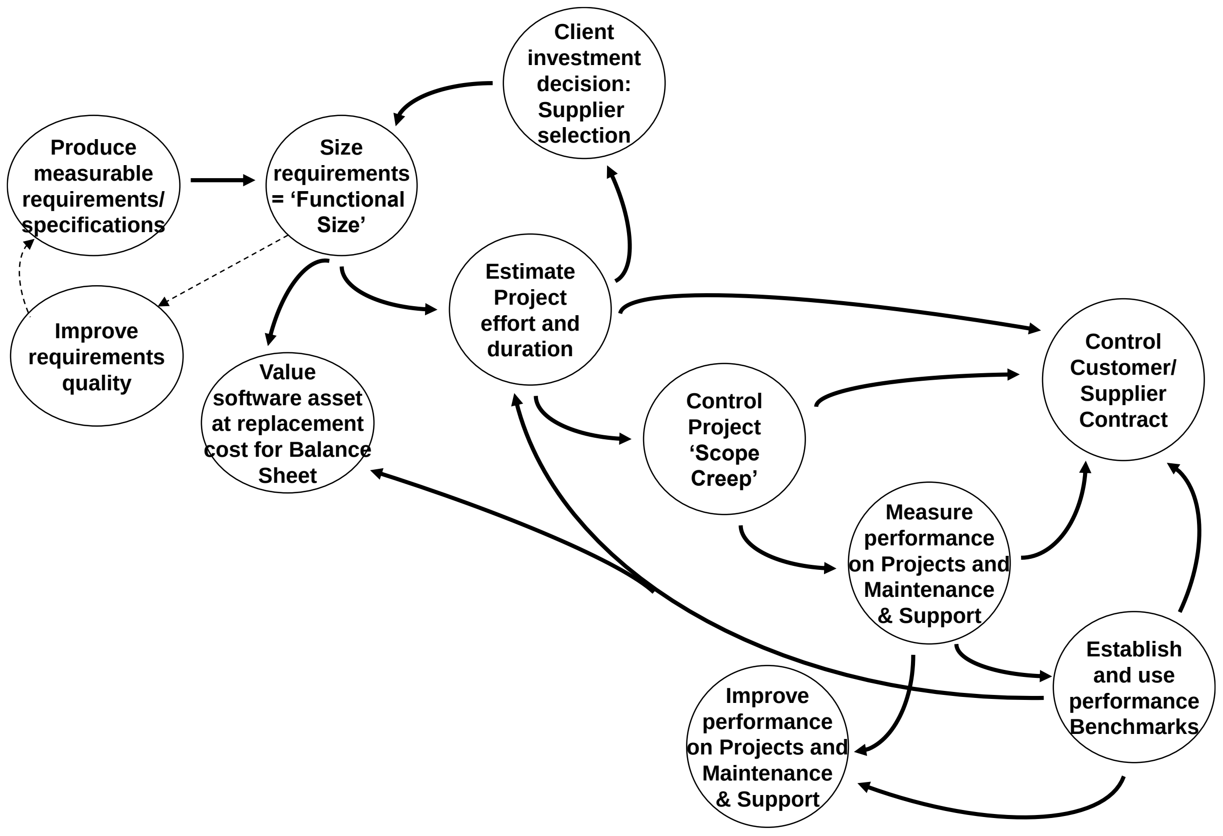

Client investment decision: Supplier selection
Produce measurable requirements/ specifications
Size requirements = 'Functional Size'
Estimate Project effort and duration
Improve requirements quality
Value software asset at replacement cost for Balance Sheet
Control Project 'Scope Creep'
Control Customer/ Supplier Contract
Measure performance on Projects and Maintenance & Support
Establish and use performance Benchmarks
Improve performance on Projects and Maintenance & Support

Figure 8.1 - Mind map of possible uses of functional size measurements

The COSMIC method is also being extensively studied by the academic research community. The www.cosmicon.com web-site has a large library of research and conference papers

REFERENCES

[1] ISO 19761:2011 - Software Engineering – COSMIC: a functional size measurement method' obtainable from www.iso.org.
[2] The COCOMO II Estimating Model', www.csse.usc.edu/csse/research/COCOMOII
[3] Albrecht, A.J., 'Measuring Application Development Productivity', IBM Applications Development Symposium, Monterey, October 1979
[4] ISO 14143-1:2012 - Software and Systems Engineering - Software measurement - Functional size measurement - Definition of concepts
[5] The 'COSMIC Functional Size Measurement Method, version 4.0: Measurement Manual'. (The COSMIC Implementation Guide for ISO 19761: 2011)
[6] Guideline for 'Measurement Strategy Patterns': Ensuring that COSMIC size measurements may be compared, version 1.0, March 2013
[7] Guideline for approximate COSMIC size measurement (In preparation. *Ad interim*, see chapter 1 of 'Advanced and Related Topics', December 2007)
[8] Guideline for sizing Business Application software, version 1.1, May 2008
[9] Guideline for sizing Real-time software, version 1.0, June 2012
[10] Guideline for sizing Data Warehousing application software, version 1.0, May 2009
[11] Guideline for sizing Service-Oriented Architecture software, version 1.0, April 2010
[12] Guideline for the use of COSMIC FSM to manage Agile projects, version 1.0, September 2011
[13] Guideline for mapping from the Unified Modeling Language to COSMIC FSM (in preparation)

APPENDIX - 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.