The COSMIC Functional Size Measurement Method

Version 4.0

Guideline for Sizing Business Application Software

VERSION 1.2

September 2014

Acknowledgements

Authors and reviewers of this version 1.2		
Diana Baklizky TI Metricas Brazil	Jean-Marc Desharnais École de Technologie Supérieure – Université du Québec Canada	Cigdem Gencel Free University of Bolzano-Bozen Italy
Arlan Lesterhuis*, The Netherlands	Bernard Londeix Telmaco United Kingdom	Andreas Schuderer APG The Netherlands
Charles Symons*, United Kingdom	Monica Villavicencio ESPOL Ecuador	Chris Woodward Chris Woodward Associates Ltd. United Kingdom

*Authors of this document. For authors and reviewers of previous versions of this Guideline, please see the version 1.1 of May 2008.

Version Control

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

DATE	REVIEWER(S)	Modifications / Additions
05-12-01	COSMIC Measurement Practices Committee	First public version 1.0 issued
26-05-08	COSMIC Measurement Practices Committee	Revised to v1.1 to bring in line with the COSMIC method v3.0
26-09-14	COSMIC Measurement Practices Committee	Revised to v1.2 to bring in line with the COSMIC method v4.0

Foreword

Purpose of the guideline and relationship to the Measurement Manual

The purpose of this Guideline is to provide additional advice beyond that given in the 'Measurement Manual' [1] on how to apply the COSMIC method v4.0 of Functional Size Measurement (FSM) to size software from the domain generally referred to as 'business application' software[1]. This is the domain for which '1st generation' FSM methods, such as the IFPUG, MkII and NESMA methods, were designed to be applicable. [2]

The Measurement Manual contains the concept definitions, principles, rules, and measurement processes which expand on the basic definition of the method as given in the ISO/IEC 19761:2011 standard [2]. It also contains much explanatory text on the concepts, plus examples of application of the method to software from various domains.

This Guideline expands on the explanatory text and provides additional detailed guidance and more examples for sizing business application software than can be provided in the Measurement Manual.

Intended readership of the guideline

This guideline is intended to be read by those who will have the task of measuring functional sizes of business application software according to the COSMIC method at any point in a software life-cycle. These individuals will be referred to as 'Measurers' throughout the guideline. It should also be of interest to those who have to interpret and use the results of such measurements in the context of project performance measurement, software contract control, estimating, etc. The guideline is not tied to any particular development methodology or life-cycle.

To apply the COSMIC method to business application software, either at the requirements stage or any time later in the software life-cycle, requires a good understanding of certain systems analysis methods, especially data analysis methods. One difficulty is that the systems analysis methods are not always used or interpreted in the same way by different practitioners. For example, these methods may legitimately be used at different levels of granularity of software requirements. But if we are to have reliable and consistent functional size measurements, we must have rules that help us to agree on only one interpretation of any given piece of functionality. Therefore this guideline describes the mapping of some of the concepts of some data analysis methods to concepts of the COSMIC model.

Readers of this guideline are assumed to be familiar with the COSMIC Measurement Manual, version 4.0. For ease of maintenance, there is little duplication of material between the Measurement Manual and this guideline.

Readers who are new to the COSMIC method are strongly advised to first read the 'Introduction to the COSMIC method of measuring software' [3] before reading the Measurement Manual. This document explains the 'why' and 'how' of measuring software, gives a brief history of FSM, and contains a 2-page summary and a more extensive overview of the method. There is also a 4-page 'Quick reference guide for sizing business applications' [4] which supports this Guideline.

Measurers who have used a '1st generation' functional size measurement method and who wish to upgrade to the COSMIC method may carry a lot of experience specific to the earlier method. Some of

[1] For a full description of what characterizes 'business application software,' see section 1.1. An ISO Technical Report [5] gives two formal models that aim to distinguish the business application software domain from other software domains.

[2] As a 2nd generation FSM method, COSMIC is the first method designed entirely on basic software engineering principles. As a result, it may be used to measure business application software, as well as real-time, scientific/engineering and infrastructure software (such as operating system software) and hybrids of all these types, in any layer of a multi-layer software architecture, and components of these at any level of decomposition.

this experience is relevant to COSMIC, and some is not. Much has to be re-learned. Someone converting from an existing method may be familiar with a detailed 'recipe book' with examples of how to measure functional size in many different situations, whereas the COSMIC Measurement Manual concentrates on defining general principles and rules, and does not have so many domain-specific examples.

Scope of applicability of this guideline

The content of this guideline is primarily intended to apply to the measurement of the functional user requirements of business application software domain. This domain includes software often referred to as 'business data processing applications', 'business transaction processing', 'management information systems' and 'decision support systems', all 'data movement-rich' software.

The content of this guideline *may* be applicable to a wider range of types of software than is commonly understood by 'business applications'. This 'wider range' could be described as 'any application software designed for use by human users for professional business use, except general-purpose software tools such as word processors, spreadsheets and such-like'. Examples for which the guideline may be applicable would include the software used by human operators to set the main control parameters and to monitor the performance of real-time systems, e.g. for process control or for telecommunications systems. Moreover, the method has been successfully applied to size some software that might be considered as 'data manipulation-rich'. As an example the sizing of an expert system has been added (see section 4.2.3, example 7).

However, until more experience has been obtained, the Common Software Measurement International Consortium claims only that the detailed rules of this guideline are applicable to the domain of business application software. Feedback of practical experience from this and wider domains would be most welcome (see Appendix C for the comment procedure).

Introduction to the contents of the guideline

For definitions of the terms of the COSMIC method in general, please refer to the glossary in the Measurement Manual [1]. Terms specific to the business application software domain can be found in the glossary at the end of this guideline.

Chapters 1 and 2 of the guideline provide background material designed to assist Measurers to extract Functional User Requirements (FUR), especially the data requirements, that are needed for measurement from software artefacts that are encountered in practice.

Chapter 1 discusses what characterizes the functionality of business application software and discusses some aspects of how FUR are developed that are relevant for measurement.

Chapter 2 defines different conceptual 'levels' of data analysis and then describes how three of the most widely-used data analysis methods map to the COSMIC concepts, namely Entity-Relationship Analysis (E/RA), Class diagrams of the Unified Modeling Language (UML) and Relational Data Analysis (RDA). Additional rules for identifying objects of interest are introduced.

Measurers are strongly advised to ensure they understand these first two chapters before proceeding to chapters 3 and 4. The latter provide extensive practical guidance and many examples on applying the COSMIC method's measurement process to business application software for, respectively, the Measurement Strategy phase, and the Mapping and Measurement phases.

Introduction to the new version 1.2 of this guideline

The new version 1.2 brings this guideline into line with version 4.0 of the COSMIC functional size measurement method, as defined in the 'COSMIC Method v4.0: Measurement Manual', [1].

Changes and other suggestions for editorial improvements made it necessary to publish an updated version of this guideline. For a summary of the main changes made in producing version 1.2 from version 1.1 of the Business Application Guideline, see Appendix B.

The main areas where readers will notice changes from version 1.1 are as follows.

- The more highly-developed discussion of Non-functional Requirements and how they may evolve as a project progresses into requirements for functionality;
- The changed definition of a ‘layer’ and the consequent elimination of the need to define a ‘peer component’’
- The improved explanation for sizing batch-processed applications.

It must be emphasized that in updating to version 4.0 of the COSMIC method, no changes were made to the basic functional sizing rules; some re-wording has been necessary for improved understanding.

In this version 1.2 of the Business Application Guideline, all the answers to the examples are identical to those given in version 1.1, except for five which have been corrected due to analysis errors (i.e. not due to changes in the rules). Two of these errors were first reported in April 2009 in Method Update Bulletin 6. The description of some other examples has been changed slightly to make them clearer, as a result of comments received from COSMIC practitioners.

The COSMIC Measurement Practices Committee

Table of Contents

1

THE FUNCTIONALITY OF BUSINESS APPLICATION SOFTWARE

1.1 Characterization of business application software

'Business application software' is distinguished by the following characteristics[3]:

- The primary purpose of business application software is to capture, store and make available data about assets and transactions in the business world (both from the private and public sectors) so as to support such business by record-keeping, by enabling enquiries and by providing information for decision-making.
- The functionality tends to be dominated by the need to store business data of varying structural complexity, and to ensure the integrity and availability of such data over long periods of time.
- The functional users of business application software are mostly human, interacting mostly on-line with the software via data entry/display devices; this means that much functionality is devoted to handling human user errors and to helping them to use the software efficiently. Apart from humans, any other peer applications, or major components of other applications that interact with the application being measured will also be functional users of the application. See section 3.2.1 for a further characterization of the meaning of 'functional user'.
- Distinct business applications (or major components of applications) may interact with each other (i.e. exchange data) either on-line or in batch mode.
- Most data are stored historically, i.e. after events happened in the real world, though some data concerns future events, e.g. plans and diaries. On-line response times must be suitable for human interaction, though some data may also be processed in batch mode. This domain does not include software that is used to control events in the real-world in real-time. However, business applications may receive data in real-time, e.g. prices in a market, and may be constrained to respond quickly (but note that the contents of this guideline do not include any discussion of the measurement of specifically real-time aspects of business applications).
- Although some business rules governing data manipulations may be logically complex, business application software rarely involves large amounts of complex mathematics.
- Business application software is present in one 'layer' of a software architecture, the 'application layer'. Invariably, application layer software depends on software in other layers e.g. the operating system, device drivers etc. for it to perform. However, a business application may itself be structured into layers (e.g. a three-tier architecture) or may be developed using components from a Service-Oriented Architecture (SOA) which itself is layered. For more on this see the Measurement Manual.
- Following on from the previous point, a piece of software that is regarded by its human functional users as a single 'business application' may be composed of several major 'components' which may be distributed over different computers (where each component resides in the application layer of its respective computer). Although this underlying structure will not be visible to the human functional users, if the purpose of the measurements is to provide sizes as input to an estimating process, it may be necessary to define separate measurement scopes for each component, for example, when different development or processor technologies are used for each component.

[3] An ISO Technical Report [5] gives two formal models that aim to distinguish the business application software domain from other software domains.

1.2 Functional User Requirements

All Functional Size Measurement (FSM) methods aim to measure a size of the 'Functional User Requirements' (FUR) of software. Yet in spite of ISO definitions, there is often uncertainty in practice on what is meant by FUR. Also, Measurers often need guidance on questions such as 'who are the functional users that should be considered', 'how to extract FUR from real-world software artefacts' and 'what to do if you cannot determine them accurately enough for a given FSM task?'

In this chapter, we aim to give general guidance to help answer questions that may arise in practice when applying the COSMIC method in the business application software domain.

1.2.1 The meaning of 'functional'

'Functional user requirements' are defined by ISO in [5] as follows:

> 'A sub-set of the User Requirements. Requirements that describe what the software shall do, in terms of tasks and services.
>
> NOTE: Functional User Requirements include but are not limited to:
>
> - data transfer (for example Input customer data, Send control signal);
> - data transformation (for example Calculate bank interest, Derive average temperature);
> - data storage (for example Store customer order, Record ambient temperature over time);
> - data retrieval (for example List current employees, Retrieve aircraft position).
>
> Examples of User Requirements that are not Functional User Requirements include but are not limited to:
>
> - quality constraints (for example usability, reliability, efficiency and portability);
> - organizational constraints (for example locations for operation, target hardware and compliance to standards);
> - environmental constraints (for example interoperability, security, privacy and safety);
> - implementation constraints (for example development language, delivery schedule).'

There are two difficulties with this definition if taken literally in the context of using the COSMIC method.

- First, requirements can exist at different levels of granularity (i.e. levels of detail). For clarity, in the COSMIC method we now limit the term 'FUR' to mean 'the functional user requirements that are completely defined so that a precise COSMIC functional size measurement is possible'. For situations where requirements have been specified at a 'high-level', i.e. they have not yet evolved to a level of detail where a precise COSMIC size measurement is possible, we will use 'requirements' or 'functional requirements', as appropriate. See the Measurement Manual, section 1.2.
- Second, when first expressed, some requirements may appear as 'non-functional' according to the ISO definition of FUR, but as the requirements are worked out in more detail, they may evolve wholly or partly into functional requirements for software that must be developed and whose FUR can be measured. This is true of many types of quality requirements. When using the COSMIC method these 'derived' FUR should be included in a measurement of functional size, even if they were first expressed as non-functional requirements.

In the COSMIC method, any uncertainty on whether a clearly-stated requirement is 'functional' or not must and always can be resolved by applying the definitions of the method's basic concepts and its principles and rules to the FUR to be measured.

1.2.2 The evolution of FUR in a typical software development project life-cycle

As implied in the definition of FUR in section 1.2.1, a statement or derivation of FUR should be expressed at the 'logical' level, i.e. FUR should be completely divorced from any consideration of physical implementation factors.

In the real-world, it is very uncommon to find an existing, 'pure' statement of FUR that can be used directly for measuring a functional size. Typically in a software development project the first document to be produced might be a high-level statement of requirements, containing functional requirements mixed with technical and quality requirements, and perhaps supported by a data model at the 'conceptual' level (see section 2.1). This might be seen as the first statement of the 'problem' to be solved.

As the project progresses, new facts come to light, new details of the requirements are found, economic choices are made on what to include or exclude and how to meet some of the requirements, etc. There may be several iterations and at each one, the latest agreed statement of what has to be done may be seen as the latest 'solution' to the preceding statement of the 'problem'; this terminology is particularly common in Agile development methods. The level of expansion of the description of a single piece of software is called its 'level of granularity'; see the Measurement Manual for the details.

The task of FSM practitioners is not to seek to measure either the 'problem' or the 'solution' as there is nothing absolute about this way of distinguishing the stages of a software project; the task is to identify and measure the FUR of the software as they evolve at any stage of the project. FSM methods measure only the FUR, not any physical design or technical implementation of the FUR. We can always infer the FUR of the software at any stage, even long after the software has been installed, implemented and is in regular use[4].

This means that at any point in the life-cycle, the Measurer must be able to derive the FUR from whatever artefacts of the software are available, whether they are an outline statement of requirements, an agreed 'specification' or 'design document', or just the installed system plus a user guide. It does not matter in principle to FSM what software artefacts are available for measurement. Whatever the state of the software and its artefacts, it is always possible to derive a corresponding statement of the *implied* FUR, though the difficulty of doing this may be very variable in practice. To do this, the Measurer must have a thorough understanding of the underlying concepts of the FSM method and of a process to analyze the artefacts available and to map them to the concepts of the FSM method.

When applied to some existing software, this process usually involves a form of 'reverse-engineering'. A typical example when only the installed system is available for the measurement is that a single physical screen layout may be found to serve two or more separate COSMIC functional processes, e.g. create and update. FSM measures the separate logical functions of the software, not its physically implemented artefacts.

Since software artefacts can exist in so many forms, it is impossible to prescribe a single reverse-engineering process from the artefacts to the COSMIC model. In this guideline, therefore, our approach is to give many examples of how to interpret real software requirements and artefacts for each of the concepts of the COSMIC model.

1.2.3 Non-functional requirements that evolve into software FUR

In the Measurement Manual, a 'non-functional requirement' (NFR) is defined as

> 'Any constraint on, including any requirement for, a hardware/software system or software product, or on a project to develop or maintain such a system or product, except a functional user requirement for software.
>
> NOTE: System or software requirements that are initially expressed as non-functional often evolve as a project progresses wholly or partly into FUR for software.'

[4] In the early stages of a project, it will be necessary to use a way of sizing functional requirements approximately – see section 1.2.4.

As stated above, a typical statement of requirements for a hardware/software system early in its life contains both functional requirements for the software, and other requirements that may appear as 'non-functional'. Further, as a software development project progresses, some system requirements will remain unchanged, but some may evolve into software FUR. Examples will illustrate both cases.

Requirements to write the software in a given programming language or to execute it at a particular data center will always remain non-functional requirements. A requirement that the software should have zero major defects in the first month of operation will always remain a non-functional requirement. Other requirements may evolve partly or wholly into software FUR.

EXAMPLE 1: A statement of FUR may define (in a lot of detail) that a new system must enable a stockbrokers' clients to enquire on their portfolio of investments and their current value over the Internet. Early in the project, a target response time is specified as a NFR. Further study, and after agreement with the sponsoring user, it is agreed that the response time requirement can only be met by developing the system to run on a certain hardware platform and also by providing continuous feeds of stock market prices to the portfolio enquiry system. The original target response time requirement is still valid, but it now results in an additional 'constraint' (specified hardware) plus the FUR of some new application software to receive the feeds of stock market prices. When such a change occurs, the number of software functions to be built will increase and hence the functional size of the software that must be built will inevitably increase.[5]

EXAMPLE 2: A statement of FUR defines a new system and includes the NFR that it must be 'easy to use'. As the project progresses, the development team assumes the company standard graphical user interface (or 'GUI') must be built. (A specific statement of the FUR for the GUI may never actually be produced, since the development team knows how to interpret such a requirement. But that is immaterial; if the GUI is provided, then the corresponding FUR can be inferred.) GUI features, such as drop-down lists, are functions of the software available to a user and they are measurable if they involve movements of data about an object of interest. Software with a GUI may have more functionality, and therefore a bigger functional size, than software lacking such functions. See sections 4.1.8 and 4.4.5 for additional guidance on sizing GUI functionality.

So requirements evolve as projects progress and some original non-functional requirements may lead to additional FUR. However, *at any time when a measurement is needed* we must aim to identify both the explicit FUR of the software and any FUR that are expected to evolve from its NFR at that point in time. It does not matter to FSM that some software FUR happened to start their life as apparently 'non-functional' system requirements.

Where a measurement is being undertaken in the context of a software contract, and when a requirement evolves into additional, agreed, measurable software functionality within the defined scope of the measurement, the cause and any assumptions leading to the increase in functional size should be documented as part of the measurement.

See [14] for more on non-functional requirements.

1.2.4 When the requirements lack sufficient detail to apply the COSMIC method

Early in the life-cycle of a software development project, requirements typically exist only at a 'conceptual' or 'high' level of granularity, i.e. not in much detail. Whilst this lack of detail may make it impossible to apply the COSMIC method as it is defined in the Measurement Manual, an estimate of functional size may still be needed, for example for an investment decision. To deal with this situation, Measurers can use one or more approximation variants of the COSMIC method as described in the 'Guideline for approximate COSMIC functional size measurement'. [6].

When using an approximate size measurement variant of the COSMIC method, it is strongly recommended to quote the best estimate of size, together with probable upper and lower limits to the

[5].Where functionality appears to have arisen as a result of a developer's implementation choice rather than as a direct result of a stated or implied FUR, the Measurer must determine from the agreed purpose and scope of the measurement whether it is reasonable to include the size of such functionality in the measurement. If in doubt, the Measurer should always try to determine 'the real FUR of the users'. This requires judgement and is not a topic on which any FSM method can give precise rules.

size. It is bad practice and can give a false sense of confidence to report a single number for the size when there is actually significant uncertainty.

1.2.5 Measurement as a quality control on FUR

All-too-often, statements of requirements or specifications are ambiguous, there may be inconsistencies or omissions and there may be plain errors. In spite of this, software eventually gets delivered to the customer's satisfaction due to informal verbal agreements between developers and the customer - but the documentation will probably never be properly corrected to reflect reality. (For more on ensuring the accuracy and repeatability of COSMIC measurements, see [15]).

This state of affairs makes the job of the Measurer more of a challenge but also potentially more valuable. One of the great benefits of applying the COSMIC method is that it helps identify inconsistencies, ambiguities, errors and omissions. In other words the measurement process is also a very good quality-control process. If a specification cannot be measured it is almost certainly unreliable in some way. The Measurer can then add great value pointing out where the defects occur and why.

Frequently, uncertainties in the documentation, or in how to interpret the physical, installed software, can be resolved by talking to an analyst, a user or some other expert in the software. Where this is impossible, the reported size should be quoted with an upper and lower limit of uncertainty, just as described above for approximate sizing.[6]

1.2.6 Who can specify FUR?

Any of the following stakeholders can potentially generate FUR for a business application. They may all need to be checked to ensure completeness of the FUR to be measured.

- Business users, either sponsors or people who will actually use the software, who may specify FUR for business and 'help' functionality and to provide for future business flexibility;
- Accountants or auditors, who may specify FUR for validation criteria, logging functionality, controls and traceability;
- Application managers, who may specify FUR for logging, security access authorization, to provide for future ease of maintenance, e.g. by requiring some data to be variable, maintainable parameters, and user (error/confirmation) messages;
- Non-business users. These may specify FUR about functional processes storing and/or processing data about objects of interest to the non-business staff, for instance functional processes to maintain data for media control of interest to operations staff. The physical media are the objects about which data is to be maintained. Other examples would be FUR about functional processes for backup or conversion and FUR about technical (error/confirmation) messages that are relevant to business users.

[6] Measurers and authors of software artefacts should cooperate in order to obtain good quality artifacts, see the recommendations in the 'Guideline for assuring the accuracy of measurements'.

2

INTRODUCTION TO DATA ANALYSIS

The reader is assumed to be familiar with the definitions of ‘object of interest’, ‘data group’, ‘data movement’ and ‘data attribute’ (or its synonym ‘data element’) as found in the Measurement Manual. When using these terms, remember that we really mean *types* of these, not occurrences. We omit ‘type’ for convenience. Only where it is necessary to distinguish types from occurrences will the text make the distinction explicit.

The COSMIC method is based on the identification of data movements in each functional process where each data movement moves a group of attributes (a ‘data group’)[7] that all describe a single object of interest. Measurers will therefore usually have to perform some data analysis of the functional requirements to be measured to identify the objects of interest and hence the data movements.

Given its importance, we remind readers of the definition of an ‘object of interest’[8].

> “Any ‘thing’ in the world of the functional user that is identified in the Functional User Requirements, about which data must be processed and/or stored by the software. It may be any physical thing, as well as any conceptual object or part of a conceptual object.
>
> NOTE 1: In the above definition, ‘process’ can mean any operation to move or manipulate data.
>
> NOTE 2: In the COSMIC method, the term ‘object of interest’ is used in order to avoid terms related to specific software engineering methods. The term does not imply ‘objects’ in the sense used in Object Oriented methods.”

The key to identifying the objects of interest of any data is therefore to ask ‘what ‘thing’ do these data describe?’ (Or, ‘what are these data <u>about</u>?’)

EXAMPLE: In the database of a company’s Personnel Information System, ‘employee’ will be an object of interest. The data group which holds data <u>about</u> each employee will have attributes such as employee ID, name, address, date of birth, date of employment, etc. Conversely, these data attributes are not objects of interest in this system: we do not want to know any data <u>about</u> employee ID, nor about name, address, etc.

2.1 Data modeling ‘levels’

Data analysis methods usually define and distinguish ‘data models’ at different ‘levels’[9] [7]. For our purposes it is sufficient to define the three modeling levels, as follows:

- ‘Conceptual’. This level shows the main things[10] in the real-world and the relationships between them that are important to think about when deciding on the possible scope of a piece of software;

---

[7] In normal language, when discussing objects of interest for which ‘data’ are held in computer systems, we usually think of the data attributes as both ‘identifying’, e.g. codes, names and descriptions, reference numbers, etc., and as quantitative, i.e. counts, quantities, amounts of money, ratios, indices, etc. But we must also consider objects of interest that have attributes containing only text, e.g. standard contract or insurance policy clauses, job descriptions, help text for applications, etc.

[8] This definition has been re-phrased from that given in the Measurement Manual v4 to improve understanding. It will be included in the proposed v4.0.1 to be issued in 2015.

[9] These are not different levels of abstraction because they are not different views of the same ‘thing’; they are models of three different, but related ‘things’

- 'Logical'. This level usually shows all the things, i.e. the objects of interest, in the real world and the relationships between them that are relevant to the software to be built, and the data groups describing these things and their relationships;
- 'Physical'. This level shows the actual data records and their relationships of the files or databases that will be managed by the actual software that will hold data about the things in the real world.

In software development, conceptual models are useful when eliciting requirements in order to establish the main things (or 'entity-types') that the software will have to deal with. Logical data models are useful later in the development process to show the data groups that correspond to the entity-types. But they typically also show more detail than the conceptual model, as the specification progresses. Finally, physical data models show the structure of the actual database or files of 'physical records' that will have to be developed and, by extension, the data requirements of screens and reports.

The entity-types of the conceptual model should map exactly onto the logical model, but the data groups of the logical model do not usually map exactly to the physical model because the latter must be designed to take into account performance, maintainability, access control and other non-functional requirements.

The important point for Measurers is to recognize that all the concepts defined and needed by FSM methods, i.e. FUR, functional processes, objects of interest, data groups, etc., should be at the logical level. This is rigorously true for the COSMIC method.

2.2 Data analysis principles

We will discuss three of the most widely-used data analysis methods, namely Entity-Relationship Analysis (E/RA), Class diagrams as proposed by the Unified Modeling Language (UML) and Relational Data Analysis (RDA). Readers who normally use only one of these methods are nevertheless encouraged to read the whole of this chapter. In principle, it does not matter which of the three data analysis methods are used to identify COSMIC's objects of interest; they should all lead to the same set of objects of interest. However, the habit of using just one of these methods may not give sufficient insights needed for success in identifying all the objects of interest correctly.

All three methods have similar aims, namely, for a piece of software to be built or maintained, to produce a model or models of the 'things' in the real-world, and the static relationships between them, about which the software is required to store and/or process data. Confusingly, the three methods use different terms, have different diagramming conventions and show different details for these same 'things'.

As a general rule, these 'things' correspond to 'objects of interest' in the COSMIC method, but it is not always so and it is vital to understand why there can be differences. As an example, both E/RA and UML Class diagrams allow the data analyst to show only the 'things' that are of interest and only in sufficient detail for a specific purpose. The diagrams resulting from these methods may therefore not show all the objects of interest that a Measurer using the COSMIC method needs to recognize. We shall discuss other examples of mis-matches in section 2.6 below.

In the COSMIC method, the generic term 'object of interest' was chosen instead of 'entity-type', 'class' or '3NF (Third Normal Form) relation', for the three data analysis methods respectively, in order to avoid using a term related to a specific method.

In this guideline we can give only a brief outline of the principles of the three methods that are relevant to functional size measurement using the COSMIC method. For fuller accounts the reader is referred to References [7], [8], [9] and/or [10].

---

[10] 'Thing' is the word we use to represent literally anything for which software must process data; it may be a physical thing, i.e. a person, or a conceptual thing, e.g. an order.

2.3 E/R Analysis

An entity-type is defined as ‘something in the real world about which the software is required to store and/or process data’. The E/RA method can therefore be used to produce a model (at the conceptual or logical levels) of these things in the real world and their relationships that is completely independent of any considerations imposed by software implementation. The relationships that are shown are ‘static’, i.e. the diagrams do not show how the relationships vary over time. The ‘degree’ (or cardinality) of the relationship between entity-types is also usually shown. For example, an order-processing system may be required to store data about customers and about the orders they place. An E/R diagram might show this requirement as below (drawing conventions vary).

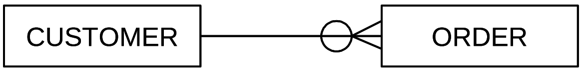

Figure 2.1 - E/R diagram showing data about customers and orders with a one-to-many relationship

The symbol on the line showing the relationship between the two entity-types Customer and Order indicates that a Customer may be associated with zero, one or more Orders at any given time. Conversely, an Order can be associated with one and only one Customer. Both entity-types will be objects of interest as a result of this requirement.

E/RA also has a specific technique for identifying additional entity-types (and therefore objects of interest) that is important for COSMIC. This is best illustrated by an example.

Consider the following FUR:

‘Data must be stored and maintained about employees and the departments in the organization in which they have worked. From this data the software must be able to supply the employment history of any given employee, that is, the list of departments in which this employee has worked in the past and the start and end dates of each employment’.

The two most obvious entity-types (or objects of interest) mentioned in these requirements are ‘Employee’ and ‘Department’, as shown in the figure below. The figure also shows the two relationships that must exist between these two objects of interest: an employee may be attached to many departments over his/her career and a department may contain many employees at any given time.

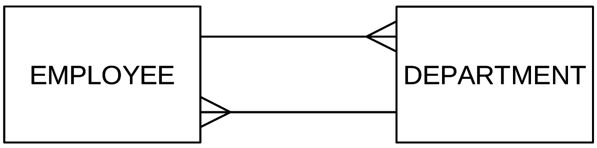

Figure 2.2 - E/R diagram showing entity-types and two one-to-many relationships

These two ‘one-to-many’ relationships, when combined, confirm that the Employee/Department relationship is actually ‘many-to-many’. In practice, such a many-to-many relationship always implies the existence of another object of interest which holds data about the relationship between any one employee and any one department. We can call this ‘intersection-entity’ the ‘Department/Employee History’, as shown in Figure 2.3, and from the requirement we can deduce it must have (at least) two attributes of its own, namely the date an employee joined a department and the date he/she left.

Figure 2.3 - The E/R diagram above resolved into two one-to-many relationships joined by an ‘intersection-entity’

ER/A teaches that whenever a many-to-many relationship is found it can and should always be resolved into two one-to-many relationships joined to an ‘intersection-entity’. The latter always represents a ‘thing’ in the real world and typically has its own attributes. For the purpose of measuring the functional size of this FUR, the COSMIC method would identify three distinct objects of interest in this example.

Note that in most practical circumstances, for an intersection-entity to be an object of interest, it must have attributes in addition to its key attributes. The key attributes identify the 'thing' that exists; the other attributes provide data about the thing. When other non-key attributes are present, the definition of an object of interest is obviously satisfied - it is a 'thing' in the world of the user 'about which the software is required to process and/or store data'. In the example above, the 'thing' has been named as the 'employee/department history'. Its key attributes indicate only that a relationship exists between a particular 'employee' and a particular 'department'. The other attributes: 'date the employee joins the department' and 'date employee leaves the department', provide data about the relationship. Hence the intersection-entity (i.e. the relationship) is clearly an object of interest, which might be defined as ‘any period of time during which a given employee worked or works in a given department’.

It is possible, but quite rare, for an intersection-entity to be an object of interest even though it has no non-key attributes. For such a case to be acceptable as an object of interest, it should be quite clear from the FUR that there is a business requirement for the software to record that a many-to-many relationship exists between two entities and that there is no requirement to hold any other data about the relationship.

> *EXAMPLE: Suppose a requirement for a matrix to record that there is a relationship between certain columns and certain rows, as in a spreadsheet, in order to keep track of which pupils have completed which assignments in a school course. In this example, the 'pupil/assignment' relationship is an object of interest. In a paper-based system the relationship would be recorded by entering a ‘tick’ in a chart as each pupil completes each assignment. In a computer system it is sufficient to establish the relationship, i.e. to store the ‘tick’ to create the intersection entity, to indicate that a given pupil has completed a given assignment.*

2.4 UML class diagrams (and use cases)

The Unified Modeling Language [10] defines a set of diagramming standards that are valuable to use as a project progresses from requirements determination through to the development of object-oriented (OO) software. UML does not prescribe any processes to produce the various diagrams and it is therefore not really a software engineering method.

(N.B. In the following do NOT confuse the COSMIC concept of an ‘object of interest’ with the OO-world concept of an ‘object’ or ‘object-class’.)

When mapping the concepts of UML to those of the COSMIC method, the most important is the one-to-one mapping of a UML ‘class’ to a COSMIC ‘object of interest (-type)’, though they are not the same concept. (A ‘class’ is defined [10] as ‘a description of a set of objects that share the same attributes, operations, methods, responsibilities and semantics, where an ‘object’ is an instance that originates from a class’.)

Contrast the above with an object of interest (type) which is “any ‘thing’ in the world of the functional user …. about which the software is required to process and/or store data”. In the COSMIC Generic Software Model, an object of interest is only described by its data attributes.

Broadly speaking therefore, the part of a UML class definition concerned with the attributes, corresponds to a COSMIC object of interest and its data attributes. So once a class has been identified, so has a COSMIC object of interest. The ‘methods’ defined in a UML class are also a source of candidates for functional processes as the methods describe the functionality associated with the class.

Class diagrams may be drawn from different ‘perspectives’ and/or at different phases in the software life-cycle showing different information through those phases. For the purposes of functional size measurement with the COSMIC method, class diagrams drawn from the ‘specification’ perspective or at the ‘analysis’ phase correspond most closely to the logical level and are the ones that matter. Class diagrams typically show the static relationships between classes (as per E/R diagrams), the attributes of each class, and the constraints that apply on the means by which objects are connected, i.e. more detail than an E/R diagram.

Class diagrams should also show 'sub-types' where they exist. (E/RA has the same concept, but sub-types are not always shown on E/R diagrams.) A sub-type of a class (or entity-type) inherits all the attributes and other properties of that class but also has its own unique attributes and other properties. For more on sub-types and whether separate objects of interest should be recognized for sub-types, see [11] and also example 5 in section 4.2.3.

This question of whether or not sub-types are shown on E/R or class diagrams illustrates the point that both techniques may be used at different levels of granularity. As an example, in the early stages of requirements elicitation, it may be established that data must be held about customers and hence 'Customer' is recognized as an object of interest. Later in the process, it may be recognized that there are significant sub-types of 'Customer' (e.g. personal, retailer, wholesaler) and these sub-types might also need to be considered as separate objects of interest in certain functional processes, depending on the requirements.

It further follows that final decisions on the objects of interest in any piece of software cannot be made until the requirements for the functional processes are known. E/R analysis and the drawing of UML class diagrams cannot be relied upon by themselves to determine the objects of interest. The required functional processes must also be known. Accurate functional sizing with COSMIC requires that the functional processes and their data interactions be known.

In this context, as UML 'use cases' are widely used, it is important to point out that a use case does not *necessarily* correspond to a COSMIC functional process. A use case construct is 'used to define the behaviour of a system or other semantic entity without revealing the entity's internal structure. Each use case specifies a sequence of actions, including variants, which the entity can perform, when interacting with actors[11] of the entity' [10].

UML practitioners are thus free to choose to define a use case for a dialog with an actor comprising a sequence of functional processes, or for a single functional process, or for just a part of a functional process according to their purpose. UML does have the concept of an event with the same meaning as in COSMIC, namely 'an event is a specification of a type of observable occurrence … that has no duration' [10], but UML defines no relationship between an event and a use case. Therefore in order to measure a use case, UML practitioners must fully understand the definition of a COSMIC functional process and all its distinguishing elements so that they can map their local practices for defining use cases to functional processes.

2.5 The normalization process of RDA

Relational Data Analysis (RDA) is a rather different, mathematically-based method that is often best considered as a 'bottom-up' approach, rather than 'top-down' as with UML (E/RA may be used both top-down and bottom-up). RDA defines a rigorous process to produce a 'normalized' logical data model from the physical data model of a real database that is typically not normalized. By 'normalized' we mean a model where each resulting data 'relation' is as independent as possible of every other data relation in the problem area. (A normalized relation is a set of data attributes – a 'data group' in COSMIC terminology – of a single object of interest.) A 'relational' database built to correspond to a normalized data model will be simpler to maintain than any non-normalized database because of this independence of the relations.

RDA defines detailed rules that enable a non-normalized set of data (such as the physical records of a non-normalized database) to be 'unraveled' through three relational forms, to Third Normal Form (3NF). The resulting 'relations' are the starting point for identifying the objects of interest and data groups for the COSMIC method.

[11] Note that an 'actor' is defined as 'an entity that starts scenarios and gets results from them'. Whilst this definition differs from that of a 'functional user', the intent of the definition seems to be similar. However, functional users include 'entities' that may only send data to and/or receive data from a functional process, but that did not start the process.

Normalization is carried out in five steps on any particular group of data (e.g. in a form, screen, report, database record or file):

1. Represent the data as a non-normalized table.
2. Identify the key for the non-normalized data.
3. First Normal Form (1NF): move repeating data groups into separate relations.
4. Second Normal Form (2NF): move data attributes dependent on only part of the key to separate relations.
5. Third Normal Form (3NF): move data attributes dependent on attributes other than the key into separate relations.

Relations in 3NF are called 'normalized' and the subjects of such relations are almost always objects of interest (but see section 2.6.3 for some additional criteria). The characteristic property of a relation in 3NF may be summarized as 'all its attributes are dependent on the key, the whole key and nothing but the key'.

2.6 From entity-types, classes or relations to objects of interest

In almost all cases, entity-types that may be identified from E/R analysis, classes shown on UML class diagrams and the subjects of relations in 3NF identified from RDA will be COSMIC objects of interest. But experience shows that cases arise in practice where data analysis methods do not consistently lead to the correct identification of all the objects of interest that must be identified for a COSMIC measurement. Other criteria are needed in addition to the rules given in the Measurement Manual. The cases are described in the next two sections and the additional criteria in section 2.6.3.

2.6.1 Input, output and transient data structures

E/R analysis, UML class diagrams and RDA are usually applied to understand the structure of *persistent* (i.e. stored) data. By applying such methods to persistent data, we can identify the objects of interest that are the subjects of the data groups of Read and Write data movements.

However, for COSMIC purposes, we also need to apply these same principles to the data structures of the input and output components of functional processes to identify the objects of interest that are the subjects of the data groups of the Entry and Exit data movements.

For many functional processes:

- the Entries will move data that will become persistent (e.g. enter data about a new customer);
- the Exits will move data that is already persistent, (e.g. enquire on an existing employee).

So in such cases the analysis to determine the objects of interest of the data groups moved by the Entries and Exits can use the same structures as established by the analysis of the persistent data.

But many <u>enquiry</u> functional processes, e.g. in management information reporting systems, generate *derived* data that is only output, not stored. The Entry (occasionally Entries) of the input and the one or more Exits that make up typical outputs of such functional processes will form structures of and will move *transient* data groups, i.e. data groups that do not survive the functional process in which they occur.[12]

EXAMPLE. Suppose there are FUR to develop an enquiry to calculate and display:

- *the total value of goods sold in a given time period by customer-type (where the latter is coded P = Personal, R = Retailer or W = Wholesaler), and*
- *the total value of goods sold to all customers in the time-period.*

The data needed for this enquiry will be taken from a database of completed sales.

[12] To clear up one possible source of confusion: the output report or the output screen displaying the result of an ad hoc enquiry is NOT the object of interest that we are discussing here. It is the data on the report or on the screen that must be analysed to find out the objects of interest. In the business application software domain, the Measurer should focus on the data of the FUR, not on the data about physical devices, documents or media.

('Customer-type' is an attribute of Customer; all customers have the same attributes, so in this example the attribute customer-type does not indicate that we have sub-types of Customer as described in section 2.4.)

These two levels of aggregation of sales data indicate that we have transient data about <u>two</u> objects of interest and hence two Exits from the enquiry. The two objects of interest are:

- *The goods sold to customers of a given customer-type in a given time-period, and*
- *The goods sold to all customers in a given time-period.*

As a cross-check on this conclusion, E/RA would say that 'the goods sold to a customer of any one type in a period' is a different 'thing' from 'the goods sold to all customers in the period'. In this example, both 'things' are really different (physical) 'things' in the world of the functional user … about which the software is required to process and/or store data, as per the definition of an object of interest. (The 'goods sold' are real 'things', i.e. physical products, that left a warehouse.)

The same conclusion is reached by applying RDA to the enquiry output. The result will show three occurrences of sales value by customer-type in the time-period (for P, R and W customers) and one value of sales for all customers in the time-period. Step 3 of the RDA normalization process (as described above) will conclude that the data about goods sold by the three customer-types is a repeating group and is therefore in a different relation from that of data about 'all customers'. See also the elaboration of this example in section 4.2.3, example 8.

2.6.2 Parameter (code) tables and objects of interest

It is possible that some 'thing' can be an object of interest to certain types of functional users for the functional processes they use, but not be an object of interest to other types of functional users for their functional processes. (The two such sets of functional processes for different types of functional users may or may not be defined as within the same measurement scope, since the definition of a measurement scope depends only on the purpose of the measurement.)

This situation often arises in business application software that is required to be highly parameterized for ease of maintenance. The normal business user of the application does not regard the parameters as objects of interest, whereas such parameters are objects of interest to those who must maintain about them, e.g. a system administrator.

We will illustrate the cases that can arise with some simple examples. See section 4.2.3, example 6, for more examples.

EXAMPLE a): Suppose an order-processing application that enables order-desk staff to enter and store a lot of data about customers, e.g. customer-ID, customer-name, customer-address, customer-contact telephone, customer-type-code, customer-credit-limit, customer-type-code, date-of-last-order, etc. If there is either an error in the entered data or the data have been processed abnormally, the order-desk staff must be notified.

In this system, 'customer' is clearly an object of interest to many functional users, e.g. to the order-desk staff. So the simplest functional process to enter data about a customer would have one Entry and one Write for the data entered about the customer object of interest, and one Exit for error/confirmation messages[13]. Total size: 3 CFP.

EXAMPLE b): Notice the attribute 'customer-type-code' in the above list. It can have several code values, e.g. P, R, W, etc., standing for Personal, Retailer, Wholesaler, etc. respectively. Suppose when entering the value of this attribute, the order-desk user is presented with a drop-down list showing the permitted descriptions for 'customer-type'. The user then selects the appropriate description and the corresponding customer-type-code is entered and stored as an attribute of customer. The descriptions are provided only for human interpretation. There is only a requirement to store customer-type-code as an attribute of 'customer'. There is no requirement to store data about customer-type in the customer record, so customer-type is not an object of interest to the order-desk user. There is still only one Entry for this functional process; the presence of the list of customer-type descriptions does not affect the size of the functional process, which is still 3 CFP.

[13]. See section 4.4.7 for a discussion of error/confirmation messages.

The pairs of values of 'customer-type-code' and 'customer-type-description' used to generate the drop-down list could be hard-coded in the software or stored in a general table of codes along with other coding systems and perhaps other parameters for ease of maintenance by a 'non-business functional user'. (This term includes 'system administrators', 'application managers', or technical or development staff, i.e. anyone whose task is to support the application by, for example, maintaining valid codes and descriptions, but who is not a normal, authorized 'business functional user'.)

The functionality needed for such parameter table maintenance would have functional processes to create new customer-type codes and descriptions, and to update, delete and read them.

Hence for any of these functional processes to create, update, etc. the customer-type code and description attributes, the software processes data about customer-type. Customer-type is thus an object of interest for the non-business user in these functional processes.

For these functional processes, therefore, any movement of data (E, X, R, W) about customer-type in these functional processes must be identified. (Examples of the functional processes that might be needed to maintain parameter tables are given in section 4.2.3, Example 6.)

If we elaborate the FUR for this example further, 'customer-type' *could* also be an object of interest to business functional users such as order-desk staff.

EXAMPLE c): Suppose this same order-processing application also stores other data about this customer-type 'thing' e.g. 'customer-type-description', 'customer-type-order-value-discount-%', 'customer-type-payment-terms, etc.

Customer-type is now also a 'thing' with its own attributes and would appear as a separate entity on a logical data model. In this system it is an object of interest to all business functional users including those who set the policy on commercial terms and those on the order-desk (and regardless of who maintains the attributes of each customer-type).

In this Example c), therefore, when entering data about a new customer, the customer-type-code must be validated against an attribute of the object of interest 'customer-type'[14]. So the simplest functional process to enter data about a new customer, as described in Example a), must now in Example c), include a Read of the data of the customer-type object of interest.

Note that in this same functional process to enter data about an individual customer, no separate Entry should be identified for when customer-type-code is entered, because it is being entered as an attribute of (i.e. data about) a customer (we are not entering data about customer-type in this functional process).

The functional process to enter data about a new customer, when customer-type is an object of interest, requires one Entry of data about customer, one Read of data about customer-type for validation, one Write of data about customer and one data movement for error/confirmation messages. Total size: 4 CFP.

In summary, we see from these examples that:

- To the business functional users on the order-desk or who maintain commercial terms for customers, customer-type is, or is not, an object of interest, simply depending on whether it has its own attributes (as in Example c)) or not (as in Examples a) and b)) respectively. In Examples a) and b), customer-type-code is simply an attribute of customer.
- To the non-business functional users who use functional processes to maintain customer-type codes and descriptions as well as other system parameters, customer-type is an object of interest in those functional processes, as in Example b).
- As a consequence, different types of functional users may 'see' different objects of interest in their respective functional processes of the same application. See also the example 7 in section 4.4.3.

---

[14] A GUI drop-down list that enables an order-desk user to select a valid customer-type description might be identical for all examples a), b) and c). But in examples a) and b), this list is related to the customer object of interest, whilst in example c), the list is related to the customer-type object of interest.

The functional processes of the business users and the non-business users may be defined in separate measurement scopes, e.g. when the purpose is project effort estimating and where different technology will be used for the main application software and for the parameter maintenance software. But there is no principle that requires these two sets of functional processes to be in separate scopes.

2.6.3 Additional guidance for identifying objects of interest

The effect of these cases given in section 2.6.2 is that we need to add some guidance for recognizing an object of interest to those given in the Measurement Manual, as follows

The key to understanding whether a 'thing' is an object of interest, or not is:

a) to determine whether the thing is really 'of interest', i.e. that it would be specified in the FUR as a thing about which the software is required to process and/or store data, and
b) to recognize that whether a 'thing' is an object of interest or not may depend on the (type of) functional user for whom the FUR are specified;
c) to remember that entity-types identified in entity-relationship analysis of logical data structures, and the subject of relations in Third Normal Form arrived at from Relational Data Analysis of physical data structures are almost always candidates for objects of interest. But the transient data of the input and output parts of functional processes must be analyzed as well as the persistent data to identify all the objects of interest.

2.6.4 Summary: different kinds of objects of interest

In the following it must be kept in mind that some 'thing' is an object of interest only if it satisfies the definition of an object of interest and the criteria of section 2.6.3. The purpose of this section is to draw the attention of Measurers to the many different kinds of objects of interest that may be found in business application software, as follows.

- 'Primary' (or 'core') objects of interest about which business applications are built to maintain persistent data, e.g. customers, employees, accounts, product-types, payments, contracts, etc. Business applications hold high volumes of these data, the objects of interest typically have many attributes and they and their attribute values change very frequently, as a result of the normal business processes.
- 'Secondary' (or 'non-core') objects of interest about which business applications need to maintain persistent data in support of the 'primary' objects of interest. We have seen an example in section 2.6.2, case b), where 'customer-type' is an object of interest with a few important attributes, e.g. 'customer-type-order-volume-discount-%'. Other examples would be:
  - in a payroll application, tables of salary bands by grade, or of income-tax bands with validity dates;
  - in an accounting system, tables of budgeted exchange rates by country;
  - in a manufacturing control system, tables of work-centres and their capacity, or of job-types and required skill levels;
  - in a banking system, savings interest rates by product, by level of deposit, with validity dates.

  'Secondary' objects of interest tend to have low volumes of data that change relatively infrequently.
- Objects of interest about which only transient data ever exists. Data about these are either input by a functional user or are derived by the application. They are not stored persistently, and appear only in the input and output of functional processes that handle enquiries or output reports.
- Objects of interest arising from the parameterization of business application software for flexibility and ease of maintenance. These 'things' are not objects of interest to the normal business functional user, but may be objects of interest to 'non-business functional users' such as a system administrator if their attributes are stored persistently and are maintainable by functional processes available to the non-business user. We have discussed examples of simple coding systems where only the code and description of the object of interest are stored in maintainable tables (for a more detailed discussion of code tables, see section 4.2.3, example 6).
- Many other types of parameters may be held in maintainable tables, for example rules for data manipulation, or the parameters for sets of rules. This is common practice in financial services industry software where there is a need to change or to introduce new financial products with new variants of rules and to be able to do this quickly for competitive reasons. Examples include loan

interest rate terms and repayment terms, life insurance policy sales commission terms, savings product trade-in or termination conditions, etc. As long as these tables of rules are required to be available directly to genuine 'functional users' (see further in section 3.2.1) for them to maintain via functional processes, each rule-type may be considered as an object of interest in those functional processes.

Note, however, that there is nothing absolute about the distinctions between these various kinds of objects of interest and the distinctions are not important for the purpose of applying the COSMIC method. The distinction will vary, depending on the industry of the software user.

EXAMPLES: 'Country' may be a secondary object of interest, or not an object of interest at all to a manufacturing company, but would be a primary object of interest to a freight-forwarding company. 'Currency' and 'Currency exchange rates' will be primary for a bank and either secondary or of no interest at all to the applications of a city administration. 'Department' may be primary for an application that maintains an organizational structure but only a attribute for the asset register.

We mention these distinctions only because such classifications are often used and it helps us understand the extent of what may be considered as objects of interest.

3

THE MEASUREMENT STRATEGY PHASE

The reader is assumed to be familiar with chapter 2 of the Measurement Manual which explains this phase in detail. This chapter of this Guideline will therefore only discuss aspects of the Measurement Strategy and give examples that are specific to the business application domain.

The importance of agreeing the 'Measurement Strategy' before starting an actual measurement cannot be over-emphasized. As a reminder, the following Measurement Strategy parameters must be established before starting a measurement.

- The *purpose* of the measurement;
- The *overall scope* of the measurement and the *measurement scope* of individual pieces of software that may have to be measured within the overall scope. This may involve determining the *layers* of the architecture in which the software resides, and the *level of decomposition* of the pieces to be measured;
- The *functional users* of each piece of software to be measured;
- The *level of granularity* of the artefacts of the software (e.g. a statement of requirements or the physical software) to be measured.

The scope of a measurement is defined in COSMIC as 'the set of FUR to be sized'. As the scope depends only on the purpose of the measurement, the purpose and the scope of a measurement are closely related. Therefore they are treated together in the following.

3.1 The purpose and scope of the measurement

There are many reasons for measuring software; in this section some examples will illustrate how approaches can vary.

3.1.1 Examples of purposes and scope

EXAMPLE a): If the purpose is to measure the work-output of a project that must develop some application software and also some software that will reside in other layers, then a separate measurement scope must be defined for each piece of software in each layer.

EXAMPLE b): If the purpose of the measurement is to determine the software project productivity (or other performance parameter) and/or to support estimating, and the business application is one piece of software running on one technical platform, then the scope will be the whole application.

EXAMPLE c): If, as in the previous example, the purpose of the measurement is related to performance measurement or estimating, and the application must be distributed over different technical platforms, then it will be desirable to measure separately the size of each component of the application that runs on a different platform by defining a separate measurement scope for each of the components. The performance measurement or estimating can then be carried out separately for each component on each platform. Note, however, that measuring the performance per platform pre-supposes that the effort per platform as well as the size will be or has been registered separately.

For further examples of measurements on application components, see section 4.3.

EXAMPLE d): If the purpose is to measure the total size of an application portfolio (excluding any infrastructure software) in order to establish its financial value, e.g. at replacement cost, then it may be sufficiently accurate to measure sizes ignoring any functionality arising from any distributed architecture. A separate scope should be defined for each application to be separately measured.

(Sizes may also be measured to sufficient accuracy using an approximation variant of the COSMIC method to speed up this task[15].) An average productivity for replacing the whole portfolio can then be used to determine the replacement cost.

EXAMPLE e): In any particular software customer/supplier relationship it is always possible for the two parties to limit the scope of the measurement of the software supplied in any way that sensibly satisfies the purpose of the measurement.

For instance, it might be that the purpose of the customer is to control only the size of the FUR resulting from 'pure business' requirements, ignoring FUR resulting from 'overhead' requirements (the two parties would need to define the latter). Or it might be that the agreement is to define two scopes, one to measure the size of the pure business application software, and the other to measure any 'support software', e.g. for security access control, logging, maintenance of system parameters and code tables, etc. (perhaps because of different pricing arrangements for the two scopes). As stated before, the scope depends only on the purpose of the measurement.

A key point to ensure, whenever the purpose is related to performance measurement or estimating for a software project, is that the scope of each piece of functionality to be measured must correspond exactly to the time and effort data for the work (to be) done by the project team on that piece of functionality.

3.1.2 Software in different layers

Business applications reside in the so-called 'application layer'. Software in the application layer relies on 'infrastructure' software (operating systems, data management software, utilities, device drivers, etc.) in other layers of the architecture for it to perform. The requirements of this kind of functionality are therefore not part of the FUR of a business application.

Readers are strongly recommended to study section 2.2.2 of the Measurement Manual on layers, particularly the definitions and rules for layers, Figure 2.2 which shows the typical layered architecture on which a business application relies, and Figure 2.4 which shows how a business application may have different layered structure depending on the architecture 'view'.

Where the deliverables of an application development project consist of the main business application and supporting utilities such as to provide logging, back-up and recovery, security access control, etc. there may be questions on whether this supporting software resides in the application layer and is therefore 'peer' to the business application, or whether it resides in a different layer. In general, when the software to be measured resides in an architecture of layers that can be mapped to the COSMIC layering principles, the Measurer should assume the layers defined in that architecture. Where there is uncertainty on the architecture, the Measurer should define the layers considering the definition, principles and rules for distinguishing layers and the examples given in the Measurement Manual.

The two most important of these rules are that software in one layer provides a cohesive set of services that software in other layers can utilize, and that a 'correspondence rule' exists between software in any two layers. This rule defines that the relationship between software in the layers A and B is either 'hierarchical' (software in layer A is allowed to use software provided in layer B but not conversely) or bi-directional (software in layer A is allowed to use software provided in layer B and vice versa).

EXAMPLE: A data logging function (for recovery purposes) may be provided as part of the operating system for any application that is set up to use it. An application can use the logging function, but not vice versa. So the correspondence is a hierarchical relationship between them and if the other conditions for layers are met, the logging function is in a different layer to the application.

Note, it is an entirely separate question as to whether support utilities should or should not be within one overall scope, together with the business application. The overall scope depends only on the purpose of the measurement. The purpose might be to measure only the size of the business application or it might be to measure the total size of all deliverables of a project team including any

[15] See the document 'COSMIC Approximation guideline [6].

utilities. However, pieces of software that reside in different layers must *always* be defined as having different measurement scopes.

3.2 Identifying the functional users of business application software

For a piece of business application software or for one of its major components (of a defined scope) being measured, the 'functional users' that are identified in its FUR may be any of:

- human users who are aware only of the application functionality that they can interact with;
- other major[16] components of the same application that are required to exchange or share data with the major component being measured;
- other peer applications (or major components of other peer applications) that are required to exchange or share data with the application (or its major component) being measured.

Note that staff who maintain software programs or who maintain stored data or rules via utilities only available to those with computer software knowledge may be considered as 'functional users' of the software if the functionality they use is defined as within the scope of the measurement.

3.3 Identifying the level of granularity

As noted in section 1.2.4, early in the life-cycle of a software development project, requirements typically exist only at a conceptual or 'high' level of granularity, i.e. not in much detail and may be at varying levels of granularity.

Consequently, the first measurement task is to check the level or levels at which the requirements exist. Then, if the requirements are not at the level of granularity required for an accurate COSMIC measurement, the Measurer should use an approximate variant of the method to determine the functional size.

For more on this, see the Measurement Manual [1] and the 'Guideline for approximate COSMIC functional size measurement' [6] which describes various approaches for approximate sizing of requirements at high levels of granularity.

[16] The Guideline for 'Measurement Strategy Patterns' [12] recognizes three standard levels of decomposition: 'Whole application', 'Major Component' (first level below te Whole Application) and 'Minor component' (any lower level of decomposition).

4

THE MAPPING AND MEASUREMENT PHASES

In the Mapping phase, the FUR of the software to be measured must be mapped to four main concepts of the COSMIC method, namely to functional processes, objects of interest, data groups and data movements. In the following Measurement phase, the data movements that must be added (plus those that must be changed or deleted in the case of an enhancement project) are counted in order to measure the functional size. As the explanation of most examples involves both the mapping and measurement processes, both phases are considered together in this one chapter.

4.1 Identifying functional processes

The reader is assumed to be familiar with the section ‘Identifying functional processes’ of the Measurement Manual [1]. The key steps of this identification are:

- Identify the separate events in the world of the functional users that the software being measured must respond to – the ‘triggering events’;
- Identify which functional user(s) of the software may respond to each triggering event;
- Identify the triggering Entry (or Entries) that each functional user may initiate in response to the event;
- Identify the functional process started by each triggering Entry.

The purpose of this section is to describe various examples that can help the Measurer to understand how to distinguish the functional processes. This can be particularly difficult for business application software where only the existing installed software is available for measurement.

We start by emphasizing a key aspect of the COSMIC method (and of any other FSM method), namely that Measurers must always seek to identify *types* rather than *occurrences* of these concepts Most of the time, the distinction is obvious; the following example is less obvious.

> *EXAMPLE: Suppose a requirement for an enquiry to display a list of employee names, selected by any combination of three parameters, namely, ‘age’, ‘gender’ and ‘education level’. If no employees meet the selection criteria, an error message must be issued. The enquiry can be satisfied by one functional process type. The data group type moved by the Entry of the functional process may have up to three data attribute types that may occur in seven possible combinations:*
>
> - *one combination of all three parameters together (age, gender, education level);*
> - *three combinations of any two parameters (age & gender, age & education level, gender & education level);*
> - *three instances of a single parameter (age, gender, education level).*
>
> *It is vital to recognize that these seven possible combinations of the enquiry input parameters <u>all</u> describe one object of interest which could be called ‘the set of employees that satisfies the selection parameters’ (i.e. there is one Entry, not seven Entries.) The main Exit of the enquiry moves a data group that describes the object of interest ‘employee’; it occurs as many times as there are employees that satisfy the selection parameters. The most likely data movements of this functional process (depending on the exact FUR) are given in the table below, giving a size of 5 CFP.*

Data Movement	Name of the data group moved	Object of interest of the data group moved
Entry	*Employee selection parameters*	*The set of employees that satisfies the selection parameters*
Read	*Employee data*	*Employee (Note: the whole employee file must be searched whatever the selection parameters.)*
Exit	*Employee selection parameters*	*The set of employees that satisfies the selection parameters (Assumption: the selection parameters are output to enable understanding)*
Exit	*Employee name*	*Employee*
Exit	*Error/confirmation message*	*Assumed to be needed in case no employees satisfy the selection parameters.)*

4.1.1 CRUD(L) transactions

It is good practice in the analysis and design of business application software to check the required stages of the life-cycle of every object of interest for which persistent data are held, because each possible transition from one stage to another (in UML terms a 'state transition') should correspond to a functional process.

The acronym 'CRUD' where C = Create, R = Read, U = Update and D = Delete (sometimes known as 'CRUDL' where L = List) can be used to help identify functional processes. Data about every object of interest must be created and will usually be updated and deleted (each of which corresponds to a step in the life-cycle of the object of interest, or a state transition). In addition data about each object of interest must invariably be read and maybe listed (neither of which involve a state transition).

Therefore, when an object of interest is identified in the FUR, the Measurer should next determine the part of its life-cycle that is within the scope of the measurement (e.g. by identifying the events that trigger state transitions), so as to identify which of the five 'CRUDL' functional processes are required in the FUR.

In practice, for some objects of interest, the FUR might specify several update functional processes corresponding to different stages in the life-cycle of the object of interest. For example, the FUR might specify separate functional processes to change the data of a person as he/she moves between the states of single, married, widowed and divorced because of different needs to add, modify or delete various relationships. Alternatively, the FUR might specify one functional process to handle all such changes of status. The number of update functional processes depends solely on the FUR.

Apart from these CRUDL functional processes that deal with persistent data, business application software usually also has requirements for enquiry or management reporting functional processes that produce (or create) transient data via Reads of the persistent data. (See section 4.1.3, Examples 1 and 4.)

4.1.2 Elementary parts of FUR: functional processes

According to its definition, a functional process represents 'an elementary part' of the FUR of the software being measured. To identify the functional processes in the FUR it is sometimes helpful first to break down the FUR into their 'elementary parts', or 'the smallest units of activity that are meaningful to the user(s)', (for example: definitions of screen, or report layouts). The converse however is not true: not every elementary part of the FUR corresponds to a functional process.

For a set of data movements to be a functional process it is necessary:

- that the set of data movements represents an elementary part of the Functional User Requirements for the software being measured, that this set is unique within those FUR and that it can be defined independently of any other functional process in that FUR;
- that the functional process is triggered by an event in the world of the functional users' and
- that the set of data movements of the functional process is the set that is needed to meet its FUR for all the possible responses to its triggering Entry.

‘Triggered’ means that a functional user – by definition outside the boundary – can be identified that detects or creates an event and then generates a data group that is moved by a triggering Entry into the functional process.

4.1.3 Screen design style and functional processes

A single physical transaction screen for entering data can, and often does, provide both the create functional process for the data about an object of interest and the update functions for each stage in the life-cycle of the object of interest, because there is so much common functionality. So depending on the design style, a single physical transaction could account for the implementation of several functional processes. The Measurer must examine the FUR to identify a separate functional process for each logically distinct function that is triggered by a separate event.

4.1.4 Physical screen limitations

In contrast to the previous case, due to the physical limitation of screen sizes, it often occurs that the input or output of a functional process must be spread over more than one screen display. Be sure to ignore the physical screen limitation and inter-screen navigation controls. They do NOT determine where a functional process begins or ends.

4.1.5 Separate functional processes owing to separate functional user decisions

Most commonly in on-line business application software, FUR to update persistent data requires two separate functional processes. The first is an 'enquire-before-update' in which the data about the object(s) of interest to be updated is/are first read and displayed. This is followed by the 'update' functional process in which the changed data is made persistent by one or more Write data movements.

The enquire-before-update functional process allows the human user to retrieve and verify that the correct record(s) has/have been selected for updating. The subsequent 'update' functional process allows the user to enter the data that should be changed, added or deleted and to complete the updating via the Write(s).

Logically, the enquire-before-update and the update are two separate, self-contained functional processes. The user may or may not decide to proceed with the update functional process, depending on the outcome of the enquire-before-update. The update is entirely independent of the enquiry. (These are examples of a human user ‘creating’ triggering events by deciding, or not, to make an enquiry and then deciding, or not, to proceed with an update.)

The FUR for an enquire-before-update can even require two separate functional processes, as follows. The task is to update an employee record, where the user knows the employee’s name but not the unique employee ID. The analysis leads to the functional processes FP1, FP2 and FP3:

EXAMPLE a): First, in Functional Process 1 (‘FP1’), the user is invited to list all employees by name:

E Request ‘list employees’

R Employee data

X Employee data (e.g. only name and ID, sufficient to choose the desired employee)

X Error/confirmation messages

Size of FP1 = 4 CFP

EXAMPLE b): The user can now, in FP2, choose the particular employee from the list and display the data that needs to be updated:

E Employee ID (= selecting the desired employee)

R Employee data

X Employee data (detailed form, all attributes)

X Error/confirmation messages

Size of FP2 = 4 CFP

EXAMPLE c): The update functional process, FP3, is then (ignoring any functionality that may be needed to validate the updated data):

E Updated employee data

W Employee data

X Error/confirmation messages

Size of FP3 = 3 CFP

Size of this FUR = Size (FP1) + Size (FP2) + Size (FP3) = 4 CFP+4 CFP+3 CFP = 11 CFP.

In this case there are two separate enquiry (or 'retrieve') functional processes FP1 and FP2 followed by the update functional process FP3. At each of the three stages the user must make a conscious decision to continue or to stop and to do something else. (The user may not find the employee he is seeking in the first list and may decide to try another tactic.) Each need for a conscious decision in the world of the functional user implies a separate triggering event, and hence a separate functional process.

Note also that the FUR could require several variations of FP2 as shown above, that is, the functional process to display all attributes of a given employee. If we examine typical FUR for FP2 in more detail, it may be that FP2 must be restricted to certain personnel staff that are authorized to update employee data and that it may be required to display some fields in protected mode that must not be updated.

In addition to this FP2, the FUR might specify a general enquiry functional process available to all personnel staff such as FP4: 'display all data for a given employee except current salary in a form that may not be updated'. There could be many other such requirements. Each such variation of these enquiries that has a separate FUR should be considered as a separate functional process (-type) and should be separately sized.

4.1.6 Retrieve and update of data in a single functional process

In contrast to the previous case where separate functional processes are needed for retrieve and update, there are also various circumstances which can lead to a FUR for the retrieval and update of data about an object of interest in *one* functional process. In such a case a Read is followed by a Write of the 'same data', without intervention by the functional user. By 'same data' we mean either:

- the data group (type) that is written is identical to the data group (type) that was read but its values have been updated, or
- the data group (type) that is written is for the same object of interest as the data group (type) that was read, but the data group that is written differs from the one that was read due, for example, to the addition of data attributes.

In such cases a Read and a Write of the 'same data' should be identified in the one functional process. As an example, applications executed in batch processing mode, often have a single update functional process containing a Read and then a Write of the same object of interest.

4.1.7 Drop-down lists arising in functional processes

When, during a functional process FP1, entering a data group describing an object of interest A, the value of a data attribute Z may be selected from a drop-down list (as in a graphical user interface), three examples the range of possible situations:

EXAMPLE a): The drop-down list displays the valid values of the attribute Z of the object of interest A that is being entered, without needing to refer to the data of any other object of interest to obtain these values. In this case the preparation and display of the drop-down list does not involve any other data movements and the existence of the drop-down list is ignored when analyzing the Entry.

EXAMPLE b): The drop-down list displays the valid values of the attribute Z of the object of interest A of which data is being entered. These valid values are obtained from an attribute of another object of interest B. Use of the drop-down list for value selection is <u>mandatory</u>. In this case identify *a Read and an Exit for the retrieval and display of the list, i.e. a total of 2 additional CFP in this functional process FP1.*

EXAMPLE c): The functional user is offered a choice of two ways of entering the value of the attribute Z when entering data for the object of interest A, depending on whether the functional user knows the valid value to be entered or needs to select a value from a valid list. Hence either (i) the functional user enters the value directly and the value is validated against an attribute of an object of interest B, or (ii) the functional user may optionally choose to display the valid values of the attribute in a drop-down list and then select the correct value, where these valid values are obtained from an attribute of another object of interest B.

For alternative (i), identify one additional Read (of object of interest B) in the functional process FP1 for the validation of the direct entry of the value of the attribute of object of interest A of which data is being entered. AND for alternative (ii) recognize a <u>separate</u> functional process FP2 for the optional display of the list of valid values of the attribute. This separate functional process FP2 has size 3 CFP (1 Entry, 1 Read, 1 Exit).

Note 1: See sections 2.6 and 4.2 for more guidance and examples on distinguishing objects of interest.

Note 2: When clicking on a drop-down list it is unlikely (if not impossible) to get an error/confirmation message. Therefore no Exit is identified for 'error/confirmation messages' in relation to entering this attribute Z.

Example b) arises very commonly in web-based business applications intended for use by the general public where quality control of the entered data needs special attention.

Example c) arises where facilities are provided both for experienced users and for inexperienced users. First, the experienced user using alternative (i) can enter the attribute value directly and we must identify one Read for the validation of this attribute value. Such a Read must be identified for every such attribute value that may be so entered (i.e. that must be validated against an object of interest) in each functional process.

Second, the inexperienced user, using alternative (ii) must make a distinct conscious decision to display the drop-down list of valid values for selection and entering, i.e. the functional user triggers a separate functional process FP2. Such a functional process FP2, of size 3 CFP should be measured once for the whole application. However, there will be as many functional processes of type FP2 in the application as there are attributes for which valid values may be displayed in this way. Use of this optional alternative (ii) is similar to using a help function from any screen.

4.1.8 Measurement of apparently inter-dependent functional processes

(See also section 4.1.9, 'Independence of functional processes sharing some common or similar functionality' of the Measurement Manual.)

The following example illustrates some of the difficulties that sometimes arise when Measurers are faced with the need to distinguish between a logical and physical view of software for the purposes of functional size measurement.

EXAMPLE: An application has FUR for two functional processes FP1 and FP2.

Functional process FP1 enables the functional user to maintain a 'credit-worthiness indicator' for any customer, that has three values (levels) 'excellent', 'normal' and poor'.

Functional process FP2 enables a customer account to be debited or credited. The FUR state that if the value of a debit using functional process FP2 causes the account balance to become negative, then the existing credit-worthiness indicator must be automatically reduced by one level.

The developer sees an opportunity to avoid duplicate coding of the same functionality. He implements functional process FP1 according to its FUR and then implements functional process FP2 using the part of the functionality of functional process FP1 that deals with setting the credit-worthiness indicator.

Two functional processes are identified:

- *functional process FP1 to maintain the credit-worthiness indicator;*
- *functional process FP2 including the functionality of lowering the credit-worthiness indicator which has been implemented in functional process FP1.*

Discussion

a) The above solution results in part of the functionality of functional process FP1 being measured twice. This is correct because when measuring a functional size of software, we measure the size of the FUR, not the size of the physical transactions that the developer has chosen to implement. This is an example of the developer taking advantage of functional re-use, which is very common in software design.

b) Suppose the scope of the measurement in the above example is defined to include functional process FP2, but not functional process FP1 which is part of another system B. When sizing functional process FP2, its data movements must still enable customer accounts to be debited and credited but must now include any Exit/Entry messages across the boundary to maintain the credit-worthiness indicator in the software B, outside the scope of the measurement. See section 4.3 for more on sizing components of business applications.

4.1.9 The 'many-to-many' relationship between event-types and functional process types

The definition of a functional process acknowledges that a single functional process may be triggered as a consequence of one or more events.

In the domain of business application software, an example might be that a requirement to display the latest transactions of a bank account 'on demand' might be triggered by a bank clerk requesting the statement from a terminal at the bank counter on behalf of the account holder or by a clerk in a call centre when telephoned by the account holder. Assuming the functional process is identical in both cases and both cases are within the same measurement scope, measure only one functional process.

The converse of this situation is also possible. A single triggering event may lead to one or more functional users initiating multiple functional processes.

EXAMPLE: When a new employee accepts an offer of employment (a single event):

- *A Personnel Officer may enter the new employee's data in a personnel database, and send a message to Security to authorize issue of a building pass (assumed to be two functional processes);*
- *A Pensions Officer may enter data about the employee in the company's pension administration system, etc.*

For more examples of the cardinality of event-types and functional process types, see the Measurement Manual, section 3.2.1 and Appendix C.

4.2 Identification of objects of interest, data groups and data movements

The reader is assumed to be familiar with the sections 'Identifying data groups' and 'Identifying data attributes' of the Measurement Manual.

4.2.1 Introduction

We assume that for the given (stated or implied) FUR, we have made a first pass at identifying the functional processes using the guidance of section 4.1 and that we must now size each functional process by identifying the data movements of its input, process and output phases. In principle this can be achieved by applying your preferred data analysis method to the input, process and output phases of each functional process in turn, to identify the various data groups that are moved. In practice, however, it is usually most effective to proceed as follows.

- First, construct a logical data model of the objects of interest of the persistent (stored) data within the scope of the measurement, using your preferred data analysis method.
- Then analyze the input, process and output phases of each functional process for the data groups to be sent to, or obtained from, persistent storage Be prepared to iterate around these two steps,

since analyzing the data of the input, process and output phases may result in identifying new objects of interest, and identifying a new object of interest may result in identifying new functional processes for its maintenance (see 4.1.1 on 'CRUDL transactions').

- Use your preferred data analysis method to analyze the transient data groups in the input and output in each functional process that outputs derived (i.e. non-persistent) data.

The first two steps of this procedure are described in more detail in the next section 4.2.2.

Warning on misleading attribute names

The reader is warned of the following pitfall. Especially when analyzing complex or unfamiliar input and output of functional processes, it may be advisable to examine all the attributes in order to identify the separate objects of interest, and hence the separate data groups. But names commonly given to data attributes in software artefacts may misleadingly indicate more separate objects of interest than actually exist. Two examples will illustrate:

EXAMPLE a): In the input to a simple 'create employee' functional process, an attribute may be labelled 'grade' when what is really meant is 'employee grade'. This is an attribute of the inbound data group about an employee; it would be incorrect to assume that this indicates a separate data group 'grade'. There is only one Entry 'employee data' in this simple case.

EXAMPLE b): For an enquiry to calculate the sales to a given customer of a given product over a given time period, the input parameters could be 'Customer ID', 'Product ID', 'Period start date' and 'Period end date'. These are the key attributes of a transient data group about an object of interest (the 'goods sold to a given customer of a given product over a given time period'). This is not an enquiry about the persistent data of objects of interest 'Customer', or 'Product', but about data that is the result of an intersection of these two and the given time-period – a transient data group.

In this example, the four input parameters should really be named as e.g. 'ID of customer to whom sales made in period', 'ID of product sold to customer in period', 'Start date of period of sales', 'End date of period of sales'. There is one Entry for the enquiry moving a data group comprising the four key attributes, and one Exit moving a data group comprising the same four attributes plus the sales value. (The Reads needed to compute the sales value could vary depending on how data concerning sold goods is stored.)

4.2.2 Identification of objects of interest, data groups and data movements

The recommended process outlined in section 4.2.1 requires the problem area to be examined in turn from the perspective of the persistent data, then from the perspective of the functional processes and any transient data, and then to iterate between these two perspectives.

The choice of data analysis method is left to the Measurer, depending on his/her experience and the software artefacts available for analysis. Relational Data Analysis (RDA) is particularly useful for analyzing the artefacts of existing software.

Construct the logical data model of the persistent data

Start by constructing a logical model of the objects of interest for which persistent data is required to exist for all of the software within the scope of the measurement. Examine the FUR for nouns, i.e. the names of 'things' about which data must be or is held. Be sure to follow the guidance given in section 2.6.2 and 2.6.3 for identifying objects of interest for different types of functional users.

Then analyze the functional processes.

All of the following must be subject to the FUR of the functional processes and to the rules for data movements given in section 3.5 of the Measurement Manual, in particular to the 'data movement uniqueness' rules of section 3.5.7 of the Measurement Manual.

For each functional process

For the input phase of each functional process

Identify one Entry for each unique data group required in the input, regardless of whether some or all of the entered data is intended for persistent storage, or whether it is transient. (Remember we are seeking data group types; differing frequencies of occurrence always indicate differing data group types.)

In addition, identify one Read for any input attribute whose value must be validated against existing persistent data of an object of interest for this functional process, plus one Exit if the valid values of the existing persistent data of the object of interest must be displayed for user selection, e.g. in a drop-down list.

Every functional process must have at least one Entry.

For the processing phase of each functional process

For each object of interest for which persistent data is required to be retrieved or for which data is required to be made persistent, identify a Read or Write respectively.

For the output phase of each functional process

Identify one Exit for each unique data group that is required in the output, regardless of whether the data group is obtained directly from data in persistent storage or is derived and hence transient. (Remember we are seeking data group types; differing frequencies of occurrence always indicate differing data group types.)

Every functional process must have an 'outcome', that is at least one Write or one Exit.

Functional processes of business applications very commonly include an Exit for error/confirmation messages, though this is not always required. (See also the Measurement Manual, section 3.5.11 and section 4.4.7 of this Guideline.)

4.2.3 Examples

(Key attributes are underlined)

EXAMPLE 1a: Simple enquiry

We have a database with two objects of interest:

Client (Client ID, client name, address, ...).

Order (Order ID, client ID, product ID, order date, ...)

The FUR for example 1a says 'an enquiry is needed to enter the start date and end date of a time-period and to output these dates plus a list of client ID's for each client that has placed orders in the period, with the number of those orders. Orders are single-item, i.e. one product-per-order.'

The solution for this functional process, ignoring possible error/confirmation messages, is as below:

E Period start and end dates

R Order data

X Period start and end dates.

X Client ID, number of orders[17]

[17] In version 1.1 of this Guideline, this example 1a showed only one Exit having attributes Client ID, period start and end dates, number of orders'. This was a mistake (which was soon corrected in MUB 6). Even if the FUR specified this layout, which would be unusual, the fact that the period start and end dates would be repeated for

In this example, the Entry moves a transient data group with the attributes 'enquiry_period_start_date' and 'enquiry_period_end_date' of one object of interest, 'time-period of the enquiry'. We assume a first Exit outputs this same data group once (it is the same for all customers). The second Exit moves one transient data group, about an object of interest which could be defined as 'the business of a given client in the given time-period'. The key of this object of interest is (enquiry_period_start_date, enquiry_period_end_date, Client ID) and its only non-key attribute is 'the number of the client's orders placed in the period'. There is one occurrence of the Exit for every client that placed orders in the period.

Since the two data groups that are output have different frequencies of occurrence (one for the period start and end dates; many for the client data), there must be two Exits.

Note that in this Example 1a, logically it is only necessary to read the Order object of interest to obtain the data needed for the output, which is why no Read of Client has been measured. The size of this functional process is 4 CFP.

EXAMPLE 1b: More complex enquiry

The FUR of example 1b is identical to that of example 1a but with the addition 'also, output the Client name and address with the Client ID for each Client that placed an order in the period.'

The solution for this functional process is now:

E Period start and end dates

R Order data

R Client data

X Enquiry period start and end dates

X Client ID, client name, client address number of orders

In this example it is now necessary to read the Client object of interest in order to obtain the client name and address, so there are now two Reads. However, as in Example 1a, we still have only two Exits. Adding the Client name and address attributes increases the size of the data group moved by this Exit, but there is still only one movement of data describing a Client.

The size of this functional process is now 5 CFP, ignoring possible error/confirmation messages.

EXAMPLE 1c: More complex enquiry

The FUR of example 1c is identical to that of example 1b, but the functional process must allow up to 3 different time-periods to be entered and the output must show the number of orders placed by the client in each period.

The report could be laid out in many ways. If laid out as in Example 1b, with three blocks, one for each of the three time periods, the number of both the Entries and Exits will be the same as in examples 1a and 1b. We would simply have multiple occurrences of the same Entry and Exit types for each time period.

A second possibility is that the report could be laid out as below.

Client ID, client name, client address

Period 1 start and end dates, No. of orders placed

Period 2 start and end dates, No. of orders placed

Period 3 start and end dates, No. of orders placed

---

every occurrence of a client that placed orders in this period tells us that the record is not normalized and that we should have counted two Exits. A comparable remark applies to example 1b.

(These two blocks would be repeated for each client that placed at least one order in at least one time period.)

A third possible layout could be:

Period 1 start and end dates; Period 2 start and end dates; Period 3 start and end dates

(As a report heading)

Client ID, client name, client address, no. orders place in period 1, no. orders placed in period 2, no. orders placed in period 3

(The latter Exit would be repeated for each client that placed at least one order in at least one time period.)

It is important to note that all three report layouts convey the same information, and all three have two Exits, even though the data groups differ for the three layouts. The size of this functional process is therefore also 5 CFP, ignoring possible error/confirmation messages.

EXAMPLE 2 – A data entry functional process

The FUR specifies a functional process that 'enables the entry of a multi-item order into a database which already has persistent data about clients and products, where

- *the multi-item orders have attributes as follows:*
- *Order header (<u>Order ID</u>, client ID, client order ref, required delivery date, etc.)*
- *Order item (<u>Order ID</u>, <u>item ID</u>, product ID, order quantity, etc.)*
- *the identifiers order ID and item ID are generated automatically by the software, and*
- *the client ID and the product ID must be validated on entry.'*

Solution for this functional process:

E Order header

R Client data

E Order item

R Product data

W Order header

W Order item

X Error/confirmation messages

In this example, the unique key of the order header is its order ID, and the unique key of the order-item is the combined (order ID, item ID). The client ID in the order header is 'the client that places the order'. It is an essential piece of data about the order. We are not entering data about the client. So we do not identify a separate Entry for client. Similarly the product ID is an essential piece of data about the order-item, so we do not identify a separate Entry for Product. Data are entered about only two objects of interest, the Order header and the Order Item. The Reads of the Client and Product data are required to check that the entered client ID and product ID are valid. The size of this functional process is 7 CFP.

EXAMPLE 3a: Simple enquiry with entered condition

Consider the following FUR1: 'The software must support an ad hoc enquiry against a personnel database to extract a list of names of all employees older than a certain age, where that age must be entered.'. Solution for the functional process of FUR1:

E The employee age limit (of the ad hoc enquiry)

R Employee data

X Employee name (for all employees within the given age limit)

To understand this solution, let us look at the data groups mentioned in this requirement. The most obvious is the persistent data group ‘Employee data’.

The query selection parameter, delivered by an Entry, is the sole attribute of a second, transient data group, whose object of interest is ‘the set of employees of the given age limit’. A third transient data group is formed by the query results displayed on the screen following the enquiry, whose object of interest is ‘an employee whose age is older than the given limit’. Note that the object of interest of the Entry and the Exit are not the same. A set, and a member of that set, are not the same ‘thing’.

The size of the functional process of FUR1 is 3 CFP, ignoring any need for an error/confirmation message.

EXAMPLE 3b: If there were also a requirement (FUR2) to output the age limit in addition to the requirement of FUR1, then there would be an extra Exit because ‘employee age limit’ is an attribute of <u>the set of employees</u> of the enquiry. The total size of FUR1+ FUR2 would be 4 CFP.

EXAMPLE 3c: If there were now a further requirement (FUR3) to output the total number of employees that satisfy the age-limit criteria in addition to the age limit (of FUR2), this would be a second attribute of the same object of interest (‘the set of employees of the given age limit’) that has ‘age limit’ as an attribute. The size of the functional process satisfying FUR1 + FUR2 + FUR3 would be unchanged at 4 CFP.

EXAMPLE 4: Report with multi-level aggregations.

Consider the following set of Functional User Requirements:

‘The software holds all data about the company’s personnel in a file. An attribute of each employee is the department ID to which each employee currently belongs. A separate table lists all department ID’s giving also the name of the division to which each department belongs.

A report is required that lists all company employees by name, sorted by division and by department-within-division. The report should also show sub-totals of the number of employees for each department ID and for each division name, and the total number of employees for the whole company’.

Solution for the functional process that satisfies these FUR:

E Selection of the report

R Employee data

R Department data

X Employee names (grouped by department within division)

X Department employee subtotal

X Division employee subtotal

X Company employee total

In this example, to produce this report requires one functional process, which must be triggered by an Entry. The functional process must involve two Read data movements, of the ‘Employee’ and of the ‘Department’ objects of interest, to obtain the data it needs from persistent storage. (‘Department’ must be an object of interest to the functional users of this application since it effectively defines the company structure.) As the report will show the names of all employees, there must also be an Exit for the Employee object of interest since all names must be printed.

However, the report must also show the total number of employees at three levels of aggregation, namely for every department, for every division and for the company as a whole. These totals are attributes of three objects of interest respectively, namely:

- the set of employees in any department;
- the set of employees in any division;
- the set of employees in the company.

Hence we must identify one Exit for each of these (even though the 'company total' object of interest occurs only once in the output). In total, therefore, this functional process has 1 Entry, 2 Reads and 4 Exits, i.e. its size is 7 CFP (ignoring the possible need for an error/confirmation message, which is not stated in the FUR).

EXAMPLE 5: Functional processes with object of interest sub-types

Sometimes, separate objects of interest should be recognized for sub-types of a particular object of interest. We ignore the possible requirement for error-confirmation messages in the following.

a) Suppose (FUR1) the object-class or entity 'Customer' has an attribute that indicates its type as 'Personal', 'Retailer', or 'Wholesaler'. If the rules governing the processing of the data of all customers are the same, and all customers have the same data attributes, then these three types are NOT regarded as sub-types of Customer. 'Customer-type' is simply one attribute of Customer amongst many.

b) Alternatively suppose the following FUR2.

- *'There shall be a customer database where all customers have the attributes: Customer ID, Customer Name, Address, Telephone no., Customer-type.*
- *'Customer-type' has three values P = Personal, R = Retailer, W = Wholesaler*
- *Personal customers have no additional attributes*
- *Retailer customers have additional attributes: Retailer Sales Region, Credit limit, Last annual turnover, Retailer price discount scale*
- *Wholesaler customers have additional attributes: Account manager, Credit limit, Last annual turnover, Wholesaler price discount scale, Wholesaler payment terms.'*

Given that Customers have some common attributes, but also have different attributes according to their Customer-type, it follows that Personal, Retailer and Wholesaler are sub-types of Customer. Logically, therefore, the object of interest 'Customer' has three sub-type objects of interest ('Personal Customer', 'Retailer Customer' and 'Wholesale Customer'). The model of the OOI and its sub-types, based on E/RA conventions, is shown in Figure 4.1 (a).

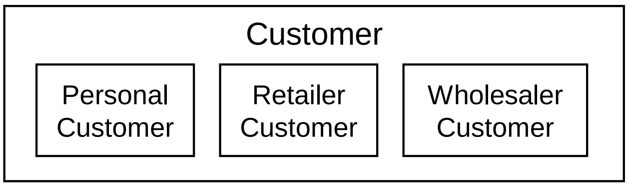

Figure 4.1 (a) – Object of interest type with its sub-types

The physical database structure would probably be as shown in Fig 4.1(b) below. The Customer record would hold the data that is common to all three customer-types. There is no need for a separate database record for Personal customers, as they only have the attributes that are common to all Customers.

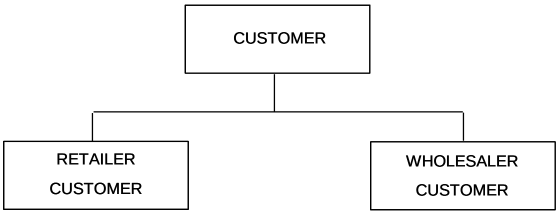

Figure 4.1 (b) - The probable corresponding physical database record structure

We now introduce various additional FUR for different types of functional processes that will illustrate the effects of needing to reference these object of interest types and sub-types.

FUR3. 'A functional process is required to produce name-and-address labels for all Customers'. This functional process would Read only the object of interest, 'Customer'.

FUR4. 'A functional process is required that enables a user to enter orders for a Personal Customer.' This functional process would need only one Read of the object of interest 'Personal Customer' in order to validate the existence of the particular Personal Customer placing the order.

FUR5. 'A single functional process is required to enable a user to enter orders for any Retailer or Wholesaler Customer.' This functional process would need to distinguish three separate objects of interest namely 'Customer' in order to validate that the Customer exists and that orders can be accepted, and then to apply the correct discount rules for pricing the order which means accessing the Retailer Customer or the Wholesaler Customer objects of interest. So this functional process FP3 would have three Reads for these sub-types.

FUR6. 'A single functional process is required to enter data about a new customer of any type.' This functional process would need to have separate Entries and Writes for each of the three sub-types Personal, Retailer, Wholesaler, i.e. a total of six CFP for these sub-processes.

The general principles are:

- where the FUR do not require the entity sub-types to be distinguished in a particular functional process, only one object of interest should be identified;
- where the FUR require more than one sub-type to be distinguished in the same functional process, each sub-type that must be distinguished must be treated as a separate object of interest.

EXAMPLE 6: Maintenance of parameter tables

In order to ease the task of maintaining data, it is common practice to store attributes of coding systems (e.g. codes and names of countries), lists of standard names (e.g. accepted credit card suppliers), textual clauses (e.g. standard letter clauses), the parameters of processing rules, etc. in tables and to provide software to help maintain these data. Each type of data – we refer to them collectively as 'parameter (-types)' - typically has few attributes. For example a coding system may have only codes and descriptions and possibly start and finish validity dates as its attributes.

Most such parameter types will not be objects of interest to business functional users of the application software that uses the parameters, but some may be. As we have seen in paragraph 2.6.2, in order for a parameter to be an object of interest, it must have attributes of its own that are of interest to, and which are usually maintained by, the business user. The Measurer must therefore normally examine any parameter tables defined as within the scope of the measurement in order to determine if any of the parameters are objects of interest to the business user.

In this section the measurement of the functions to maintain parameter tables is examined where the maintenance is carried out by a functional user who is a 'non-business user'. (As above we use this term to include 'system administrators', 'application managers', or technical or development staff, i.e. anyone whose task is to support the application by maintaining the parameters, for example, valid codes and descriptions, but who is not a normal, authorized 'business user'.) For anyone, e.g. a 'non-business' functional user, who must maintain the parameter tables, the attributes of the parameters do describe objects 'of interest' to that user.

Several situations may arise.

EXAMPLE 6a: Parameter tables are typically provided with a set of 'CRUD' (Create, Read (i.e. enquiry), Update, Delete) functional processes to maintain the parameter values, which must be available to the non-business user.

In the simplest possible case, one set of CRUD functional processes might be provided to maintain all parameters in one table. For example, to change any one existing parameter occurrence, the non-business user would have to display the parameter table contents and page through them to find the particular parameter occurrence and its attributes to be maintained. Effectively there is only one object of interest to the non-business user, namely 'parameter'. The R (enquiry) functional process would have size 3 CFP (1 Entry, 1 Read, 1 Exit to show the result, assuming no error/confirmation message is needed. The C, U and D functional processes would each have size 3 CFP (1 Entry and 1 Write and 1 Exit for an *error/confirmation message arising from data validation failure or success). The total size of the set of four processes of the CRUD set is hence 12 CFP.*[18]

EXAMPLE 6b: A more realistic situation than described in a) is that special utility software is provided to maintain the parameters. Unfortunately it is impossible to generalize on sizing the functional processes of such software since there are so many potential variations.

At the other extreme from case a), it could arise that the attributes and validation rules for each parameter-type are so different that each parameter-type needs its own set of CRUD functional processes to maintain its value occurrences and each parameter-type is a separate object of interest.

More likely there would be some commonality of attributes and validation rules across the various parameter-types. For example, for coding systems, each system must have an associated set of validation rules which impact the 'Create' and 'Update' functional processes; these will be identical for some coding systems, but unlikely to be the same for all.

EXAMPLE 6c: Suppose a case of seven coding systems whose code/description attributes have identical validation needs, so that their values can be maintained by one set of CRUD functional processes. Consider the 'Create' functional process. Suppose the non-business user calls up this functional process and must first select from a list the particular coding system that needs new values. He/she can then enter one or more pairs of new code/description attributes. This create functional process has 2 Entries (one for the coding system selection and one for the code/description data entry), 1 Write and 1 Exit for an error/confirmation message. Total 4 CFP. It might appear that there should be 7 Writes, but in this case the subjects of these seven coding systems have really been abstracted to one object of interest, hence there is only 1 Write.

As to the 'Read' functional processes for enquiring on these seven coding systems, there are endless possibilities for the requirements, e.g.

- *enquire on the description corresponding to a given code,*
- *display all codes/descriptions for a given coding system.*

Probably all of these requirements could be met by one or two functional processes for all coding systems of some general code table maintenance software.

The general utility of example b) could serve several business applications. Whether its size should be included with that of any one of the applications is again a question for the definition of the scope of the measurement.

EXAMPLE 7: Measuring an expert system

An expert system is a computer system that emulates the decision-making ability of a human expert. It does so by storing the rules used in decision-making and providing logic to exploit those rules. The following is an initial outline statement of requirements for a simple expert system.

'An expert system must be developed for retrieving possible holidays for clients. The data of a great number of holidays will be stored. The client user must answer a number of questions, such as the type of the holiday (beach, cruise, city travel, etc.), destination and price. 'Doesn't matter' is allowed as an answer. When all questions have been answered the expert system will list and provide details of the holidays that satisfy the entered answers, if any are found. The user may store any set of questions and the given answers for re-use, under a name to be entered.

For ease of both understanding, maintenance and explanation the expert system shall be rule-based. The expert system shall explain why a question has been posed when the user enters 'why' rather than an answer. The expert system shall then show the rules in which the answer to this question is a factor. When the expert system shows the holidays that satisfy the entered answers it

[18] In version 1.1 of this Guideline, we wrote 'assuming no error/confirmation message, for each of the CRUD functional processes. In fact an error/confirmation message Exit is almost certainly needed for each of the C, R and U processes in case of validation failiure or to confirm success, but probably not for an enquiry process (the 'R' of the 'CRUD') assuming the list of existing parameters can always be displayed.

shall explain how it derived these results when the user enters 'how'. The expert system shall then show the rules that were used in deriving the results.

Functional processes to be expected are, amongst others:

For the client user

- *Specify the client's wishes by entering answers to the questions, store these, and show the holidays that satisfy the entered answers;*
- *Show the list of names of the previously stored question and answer sets;*
- *Show the questions and answers of a specific set;*
- *Allow the use to modify the answers to one or more question and show the resulting holidays that satisfy the modified answers (the 'what-if' scenario);*
- *Show the rules in which the answer to a given question is a factor (the user's 'why' request);*
- *Show the rules that were used in deriving the results for a particular question and answer set (the user's 'how' request).*

For the system administrator

- *Maintenance of holiday data (add, change, delete holiday data. List all holidays stored. Show data of a specific holiday;*
- *Maintenance of the rules (add, change, delete rules). List all rule names. Show data of a specific rule;*

To be decided

- *Depending on the functionality of the expert system, there may be one or more functional processes for the drop-down lists for answering the questions.'*

Analysis

The first functional process listed above, assuming there will be one functional process to enter answers for all questions, for a client user who has not previously used the system, has the following data movements:

E	*Answers*	*Object of interest: Set of holiday requirements*
R	*Holiday rules*	*Object of interest: Holiday knowledge*
R	*Holiday data*	*Object of interest: Holiday*
X	*Show holidays*	*Object of interest: Holiday*
X	*Error/confirmation message*	*(In case no holiday satisfies the requirements)*

If the user wishes to save this set of questions and answers for future use, he/she initiates a separate functional process with the following data movements

E	*Questions & answer set name*	*Object of interest: Client_Questions & answers set*
W	*Questions & answers set*	*Object of interest: Client_Questions & answers set*
X	*Error/confirmation message*	*(The user needs assurance that the set has been saved.*

The first of these two functional processes obviously has rule-processing logic to generate the list of recommended holidays from the entered set of holiday requirements and the stored holiday knowledge. This rule-processing logic, which is pure data manipulation, is associated with the 'Show holidays' Exit.

EXAMPLE 8: See the example in 2.6.1. The two objects of interest mentioned there are:

- *The goods sold to customers of a given customer-type in a given time-period, and*
- *The goods sold to all customers in a given time-period.*

The Exit data movements shown below are followed by the data groups moved, each consisting of one or more attributes of one of the objects of interest above:

E Time-period of the business

R Customer

R Goods sold (or so[19])

X Time-period of the business (necessary to understand the output data)

X Value of goods sold per customer-type (in the time-period)

X Value of goods sold to all customers (in this time-period)

4.3 Sizing components of business applications

When a business application is distributed over two or more technical platforms and the purpose of a measurement is to measure the size of each component of the application on each platform, a separate measurement scope must be defined for each application component. In such a case the sizing of the functional processes of each component follows all the normal rules as already described. The only new point is how to handle data movements involved in inter-component communications.

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

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

In the examples below, we first give the case (a) when the purpose is to size a whole application ignoring that it is split into two components on separate technical platforms and then the cases (b) and (c) when the purpose is to size each component of the application separately.

EXAMPLE 1: Suppose a simple two-component, client-server application. Component A executes on a PC front-end that communicates with component B on a main-frame computer holding some 'legacy' data describing one object of interest. The (human) functional user triggers a functional process on the PC that requires this data to be read from the main-frame and displayed on the PC. (The following analysis ignores possible error/confirmation messages. For a fuller discussion of this example in which error/confirmation messages are also considered, see the Measurement Manual, section 3.5.8.)

Case (a) Scope of the measurement is the whole application

The (human) functional users of the application have no knowledge of how the application is physically distributed over the PC and the main-frame. The split of the application into two components is invisible, as also are the data movements between the two components. Similarly,

---

[19] The data needed to determine the 'goods sold' could be obtained in many ways. Examples: the computation might involve one Read of a file of orders of status 'sold, invoiced and paid for' or it might require two or more Reads of a data warehouse of historic sales data, or other solutions, depending on how the required data are stored.

any additional functionality in lower layers needed to achieve the communications between the PC and the main-frame is invisible and is ignored when measuring the application. The data movements of this functional process are then as shown in the Message Sequence Diagram of figure 4.2 below (total 3 CFP):

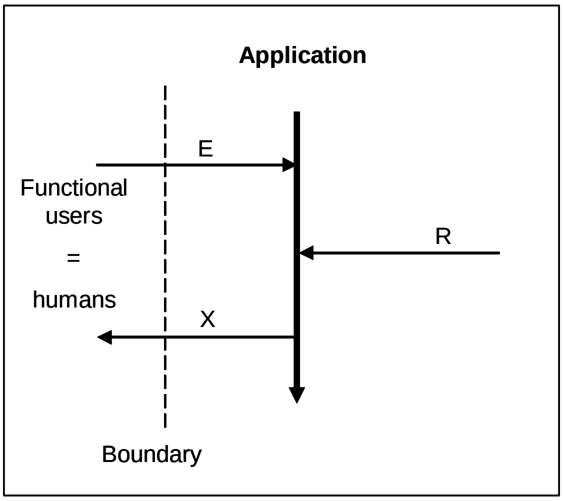

Figure 4.2 - Scope of the measurement is the whole application

Case (b) Scope of the measurement is component A

From the viewpoint of the (human) functional users, this case (b) of the PC component A is identical to that of case (a) for the whole application. The (human) functional users of the PC component have no knowledge of how a Read on the PC is executed. From their viewpoint, they trigger a functional process to retrieve data from persistent storage and receive back the results.

But limiting the scope of the measurement to component A results in the functionality being revealed that is actually needed to obtain the data from component B rather than by a Read - see figure 4.3 below. Component A must issue a 'request to obtain' (or 'get' command) with the data selection criteria as an Exit to component B and receive back the required data from component B as an Entry. (Component A has no knowledge of how component B obtains the requested data; it could be via a Read, or by local calculation or from some other software.)

Component B has become another functional user of component A, in addition to component A's (human) functional users that have already been discussed in case (a). The data movements of case (b) are therefore as shown below (4 CFP).

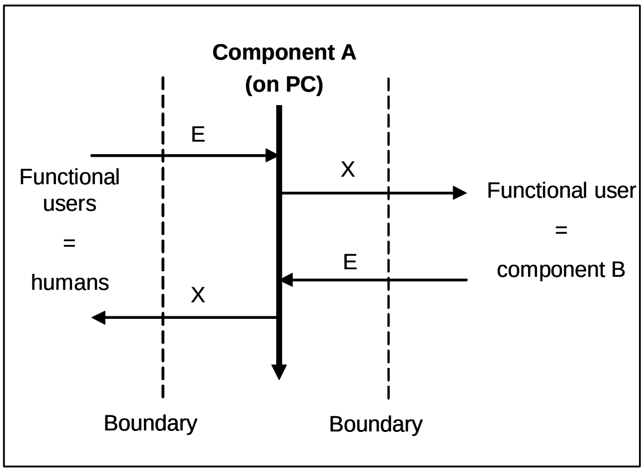

Figure 4.3 - Scope of the measurement is component A

Case (c) Scope of the measurement is component B

Defining the scope as 'component B' reveals that component B's functional user is now the PC application software component A, where the triggering event of a request-to-obtain-data occurs. Note that the components A and B are functional users of each other; their exchange of data takes place across a mutual boundary.

In component B, the 'Read from database' functional process is triggered by the Entry 'request to obtain' from component A with the read parameters. Component B executes the Read and outputs an Exit to component A containing the requested data.

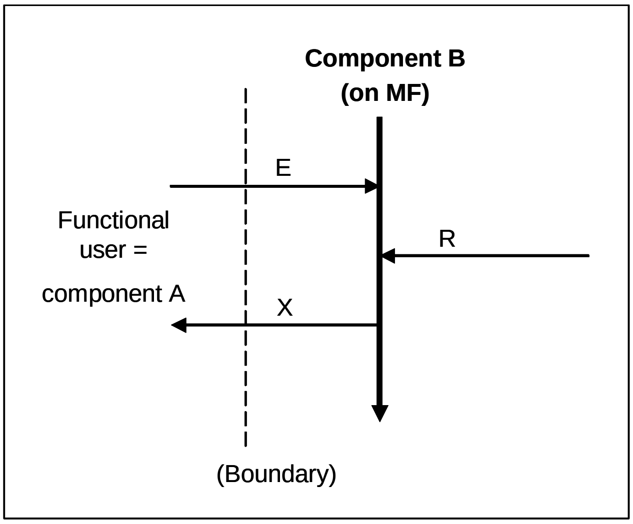

Figure 4.4 - Scope of the measurement is component B

We see that when measured separately, the size of component B is 3 CFP.

It follows that the increase in total size of the application when the purpose is to measure the size of the two application components separately is (4 + 3 – 3) = 4 CFP compared with case (a).

As another cross-check on this conclusion, and as pointed out in section 3.1.1, example c) above, the size of the application ignoring the physical split into components should be: (the total size obtained by adding up the size of each component measured separately, i.e. 7 CFP) less (the size of the inter-component data movements, i.e. 4 CFP) resulting in 3 CFP, as in case (a).

EXAMPLE 2: Now suppose an application with two components distributed over separate technical platforms, where each component may trigger functional processes in the other component. An example might be an application that has component A on a laptop that enables field-sales staff to enter data over the internet to another component B on a server. In addition, the server can trigger functional processes to send data back to the laptop, independently of the functional processes on component A.

Case (a) Scope of the measurement is the whole application

This case now differs from Example 1 case (a) above. The application has two functional users, the human (e.g. field sales staff) functional users of the laptop and 'something' (e.g. a clock) that triggers the transmission of data from the server to the laptops. But the functionality that enables the two components to communicate with each other remains invisible.

So, for example, a functional process that is triggered on the server to send data to the laptops might have:

- *An Entry from the clock to trigger the process (physically on the server)*
- *One of more Reads (physically on the server) to retrieve the data from persistent storage*
- *One or more Writes to make the data persistent and/or Exits to display the data (both data movements physically on the laptops)*

Case (b) Scope of the measurement is component A

Component A now has two functional users, namely the human (e.g. field sales staff) and component B, both of which can trigger functional processes on component A. These functional processes should be analyzed in the normal way.

Case (c) Scope of the measurement is component B

Component B also has two functional users, namely component A and the 'something' (e.g. a clock) that triggers its functional processes. These functional processes of component B should be analyzed in the normal way.

The effect on the total size of the application for this Example 2 of having a purpose to size each component separately, as in cases (b) and (c) is likely to be a significant number of additional CFP due to the inter-component data movements compared to case (a) where the purpose was to size the application ignoring its split into separate components.

4.4 Other measurement conventions

4.4.1 Control commands and application-general data

'Control commands' are defined as ' commands that enable human functional users to control their use of the software but which do not involve any movement of data about an object of interest of the FUR of the software being measured'.

The concept of control commands is defined only in the business application software domain. Control commands must be ignored when measuring business applications and their major components.

Examples of control commands are the functions that enable a functional user to display/not display a header, display/not display (sub-) totals that have been calculated, navigate up and down and between physical screens, click 'OK' to acknowledge an error message or to confirm some entered data, etc. Control commands also include menu commands and links to web pages that enable the user to navigate to one or more specific functional processes but which do not themselves initiate any one functional process. See also section 4.4.2.

Similarly, 'application-general' data, i.e. any data related to the application in general and not related to an object of interest of a specific functional process, must be ignored. Hence header and footer data (company name, application name, system date, etc.) appearing on all screens and reports, that is not related to objects of interest on those screens or reports, is not measured.

4.4.2 Menus and the triggering Entry

As stated above, a menu command that enables the user to move around the software, but which does not launch any functional process (e.g. that only enables the user to navigate to other sub-menus) should be regarded as just a control command and should be ignored in the business application software domain. Similarly, ignore a menu command that results only in the display of an 'empty' input screen for a specific functional process. The Entry (or Entries) for this functional process is (or are) in the filled-in screen and the functional process is considered to be triggered when it first receives data via an Entry.

But often a menu command will launch a specific functional process, with or without input data for that process, for example an enquiry functional process. When a user presses a button on a menu to launch a specific enquiry functional process P, a message goes to the software saying 'start this specific enquiry P'. If it happens that this enquiry also needs selection parameters that the user must input before pressing 'Enter', these are additional attributes of the same data group (but now the message is: 'start this specific enquiry P with these parameters'). In both cases this message is the one triggering Entry for this functional process. The object of interest is 'the subject of enquiry P'.

Every functional process must have a triggering Entry and the latter usually has associated data manipulation - in this case the initialization of the functional process. So even when an enquiry functional process that needs no input data is triggered by the click on a menu button and the latter appears to be a control command, there will always be some data manipulation for the specific enquiry. Hence, even if there is 'no data' entered, this use of the menu button provides the triggering Entry for the functional process.

4.4.3 Applications processed in batch mode

Fundamentally, when measuring the size of functional processes, it should make no difference whether the functional process is required to be processed on-line or in batch mode. But in practice the Functional User Requirements for processing in batch mode do sometimes necessarily differ from their equivalents for on-line processing. For example GUI features are unique to on-line processing, while in a batch processing stream, all the functional processes may be required to output error messages to a common file that is printed as a common error report.

By definition, all data that has been entered as *input* for batch processing must be temporarily stored somewhere before the process can start. See Figure 4.5. (N.B. we distinguish temporarily-stored input data - all the Entries - from persistent data that might also need to be read or written by the batch process.) When measuring a batch-processed application, any temporarily-stored input data should be analysed in the same way as if it were being entered directly to the application. This input data is not 'persistent data'.

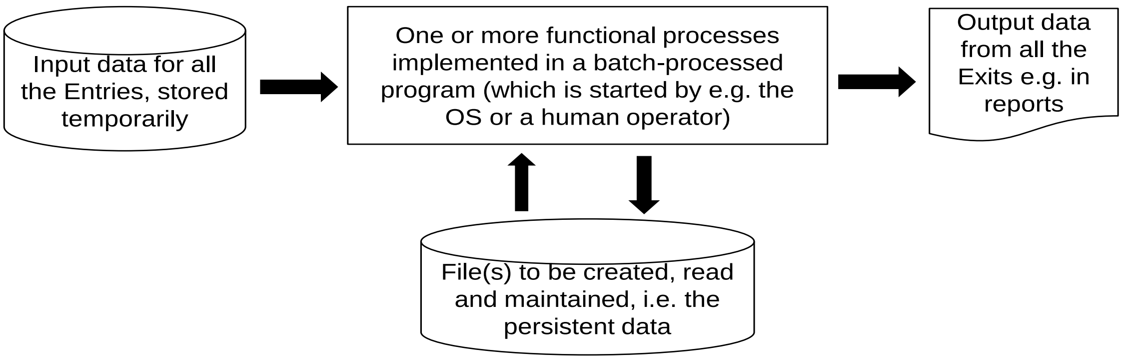

Figure 4.5 - A batch-processed 'job', implementing a collection of functional processes

NOTE: A requirement that some input data be batch-processed is a non-functional requirement (NFR). The effect of this NFR is that the input data must be available (as a 'batch') for input to the batch processed application. How that happens in practice does not concern the analysis of the FUR of the batch-processed application.

Note also that each functional process (-type) to be processed in a batch 'job' should be analyzed in its entirety, independently of any other functional processes in the same job. Each functional process in the job must have its own triggering Entry.

The key tasks of the Measurer in analyzing batch application streams are to identify the separate triggering Entries and hence the separate functional processes, and who are the functional users involved in the batch processes.

The question of who or what are the functional users of the functional processes to be processed in batch mode, can be illustrated with three cases.

a) The input data to the batch process has been entered on-line by humans and has been accumulated in a temporary file for batch processing. The processing of the separate functional processes in the batch stream may be analyzed as if the human functional users entered the data on-line for each separate functional process.
b) A file of data records is transmitted in batch mode from application A as input to application B for processing; applications A and B are functional users of each other.
c) A batch process requires no input data, e.g. a batch process to produce end-of-month reports that is physically triggered by a clock-tick or operator command or by a user from a menu selection. The batch process may be analyzed as if a human functional user triggers the end-of-month reports.

EXAMPLE 1 for case a): The batch stream for an order-processing system might contain functional processes to add new clients, add new and delete obsolete products, enter new orders, enter order cancellations, etc. Each of these different functional processes should be analyzed 'end-to-end' and independently of any other functional process in the same stream. Each functional process is triggered by its own triggering Entry, and should be analyzed as if the data (including other possible non-triggering Entries needed by the functional process) were being entered on-line by the human functional user.

EXAMPLE 2 for case b): Application A, the software being measured, is required to transmit some persistent data to application B in batch mode (applications A and B are functional users of each other). The functional process of application A that transmits this data has the same data movements regardless of whether the transmission of the data from A to B is in batch mode or whether the data records are sent individually, one at a time. The functional process must have:

- *One Entry to start the process;*
- *One Read and one Exit for every object of interest for which persistent data must be transmitted out.*

EXAMPLE 3 for case c): Suppose a batch process that does not require any input data. An example would be a batch 'job' to produce a standard set of reports, none of which need any external input. The Measurer must first decide whether the 'job' consists of one or more functional processes, noting that each functional process in a batch stream must have its own triggering Entry. An example criterion for distinguishing separate functional processes might be that different reports or sets of reports are produced for different types of functional users or are required at different frequency, e.g. weekly versus monthly reports in the same stream. There must be a good business reason why such a batch stream is divided into more than one functional process. Each report or set of reports that is considered to be produced by a separate functional process must then have one Entry and as many Reads and Exits as are needed to create and output the reports. The Entry may convey no external data across the boundary, but it conveys the signal to 'start this particular functional process' and may well involve initialization data manipulation.

Other examples that may help analyse batch processes are as follows;

EXAMPLE 4: Often, a single summary report will be produced for the processing of the batch stream. It will normally be found to contain at least one Exit (-type) for each functional process (-type) in the stream. As an example, if the report shows, for a specific functional process (-type), a count of the number of its occurrences that have been processed, plus a list of the error messages relating to failed occurrences of that functional process, then identify two Exits for that functional process (one for the count and one for the error messages). Then repeat that measurement for each functional process (-type) whose summary data may be shown on the report so as to add up the total number of Exits on the report.

EXAMPLE 5: In batch mode, a single update functional process usually needs to read the existing record about a single object of interest before updating it and writing back the updated record. (This is in contrast to processing on-line, where usually an update functional process is preceded by a separate enquiry functional process.) Further, in batch mode, normally a delete functional process needs only a Write to delete the record.

EXAMPLE 6: Sub-processes or program steps that may be needed in the middle of processing a batch stream such as a sort, or a checkpoint/re-start 'save' (which is a 'control' feature, implementing roll-back technical requirements) should be ignored in the business application software domain.

EXAMPLE 7: Suppose a business requirement for the application being measured to import some employee data by an interface file from another application in a batch stream. Suppose subsequently, the computer production manager adds a requirement for a general-purpose utility to move any particular version of any file type and to ensure that it is not processed twice. As a result, a standard file header is then added to every interface file in the organization. The file header contains data about the file (file type, file ID, processing date, number of records, hash totals of specific fields, etc.). These data describe an object of interest to the computer production manager called 'Interface file'.

In this situation, whenever the importing application must process any one physical interface file, two types of functional processes are involved namely:

- *The functional process of the general-purpose utility that processes the standard file header and checks that the file has not been previously processed before passing it to the importing application.*
- *The functional process that is specific to the importing application and to the file type being processed; in this example the employee files conveying data describing an object of interest to business users ('Employee').*

These two functional processes could have the following data movements, respectively, for instance:

For processing the header:

E	*Interface file data*	
R	*Interface file history*	*(file already processed?)*
W	*Interface file history*	*(store result of processing)*
X	*Error/Confirmation Message*	

For processing the employee data:

E	*Employee data*	
W	*Employee data*	*(data about an existing employee is overwritten)*
X	*Error/Confirmation Message*	

Note: When sizing an application that requires data to be entered via one or more batch interface files, all using the utility:

- the utility functional process should be counted once for the application
- each functional process-type that enters and writes a specific file-type should be counted, i.e. there are as many of these functional processes as there are interface file-types to be entered in the application (assuming the validation and processing is different for each file-type).

4.4.4 Multiple sources, destinations and formats of a data movement – applications of the 'data uniqueness' rule

The 'data uniqueness' rule b) for Entries and Exits in the Measurement Manual states:[20]

"A Functional User Requirement may specify data groups to be entered into one functional process from different functional users, where each data group describes the same object of interest. One Entry may be identified and counted for each of these data groups that must be distinguished as different according to the FUR.

The same equivalent rule applies for Exits of data to different functional users from any one functional process.

NOTE: Any one functional process may have only one triggering Entry".

Note that it is not the multiplicity of source or destination functional users, as such, that determine if we have different data movements, but the multiplicity of different data groups required to be received or sent according to the FUR. For instance, if the same identical Exit is sent to two physical devices or to two destinations, only one Exit is identified. But if the Exits to the two devices or destinations differ significantly (i.e. beyond the completely trivial, differences that cause some extra analysis or design), two Exits are identified.

[20] See the 'Data Uniqueness' rule in section 3.5.7 of the Measurement Manual for v4.0 of the COSMIC Method. The version of the rule quoted here is taken from an 'Errata and Corrections List' (published August 2014) which will be included in version 4.0.1 of the Measurement Manual to be published early in 2015.

Two data groups describing the same object of interest should be considered as different if they have different attributes and/or different formats of the same attributes.

EXAMPLE 1: A data group is displayed as output on a screen, and printed in the same format. One Exit is identified.

EXAMPLE 2: As per EXAMPLE 1. The output contains the same attributes as the screen display, but the printed layout is different[21]. Two Exits are identified. An example would be where a data group is displayed on a screen in graphical form but is printed in numerical form. The data manipulation and formatting differ for the two presentations of this data.

EXAMPLE 3: A file (i.e. one or more outbound data groups) must be distributed by a business application to more than one destination (= functional user) and the FUR of the application require differences in processing for these outbound data groups (i.e. resulting in different data manipulations, hence different attributes in the Exits) to the different functional users. One Exit per object of interest is identified for each functional user for which different processing is required.

EXAMPLE 4: A set of FUR state: 'Client data must be displayed after entering a client name. If there is only one client with the entered name, all the client details are shown immediately. If there are more clients with the same name, a list is shown of the clients with this name, plus sufficient client data (e.g. their addresses) to distinguish them. The right client can then be selected and its details are shown.'

Solution: two functional processes are identified. The first functional process produces the client details for the entered name or alternatively the list of clients with that name, plus the distinguishing data (e.g. their addresses). The second functional process is needed to select from the list and enter the client-id if there is more than one client with the entered name, as follows:

Functional process 1, showing details or list:

E Client name

R Client data

X Client data (one is found)

X Client list data (more are found)

X Error/confirmation messages

Functional process 2, for selecting from the Client list and then showing Client details:

E Client-id (by selecting from the Client list)

R Client data

X Client data

X Error/confirmation messages

Functional process 1 illustrates the exceptional situation referred to in the Measurement Manual where two different data groups ('Client data' and 'Client list data') describing the same object of interest ('Client') must be moved in data movements of the same type (Exit), in the same functional process.

4.4.5 GUI elements

For GUI elements conveying control data, the rules for control commands apply (see section 4.4.1), i.e. they are ignored.

For GUI elements conveying data related to objects of interest, the rules for identification of functional processes and data movements apply. See especially section 4.1.7 'Drop-down lists arising in functional processes'.

[21] 'Different' implies that the differences are recognized in the FUR and that some additional analysis, design and testing effort has been needed to implement the differences.

4.4.6 Authorization, help and log functionality

Authorization, help and log functionality may be measured if their functionality or changes to their functionality are explicitly described in the FUR, the functionality is in the application layer and it is agreed they should be within the scope of the measurement.

Existing authorization, help or log functionality that is exploited by the application being measured, but is not specific to it should normally be excluded from the scope of the measurement.

A Help button that is provided on each screen and which provides the same service from wherever it is pressed should be measured as one functional process for the whole application. (This assumes that the functional process type is the same for all screens of the application and that the context-dependent output simply represents different occurrences of the same output-type. The Help function, once invoked, may offer other functional processes for more searching enquiries. In this case each of these functional processes must be analyzed according to the usual rules, assuming the Help sub-system is within the measurement scope of the business application)

Depending on the FUR, help and log functionality may be required either as <u>separate</u> functional processes for the application (in which case these functional processes are measured once each for the application), or as <u>part of</u> some or all of the functional processes of the application.

> *EXAMPLE: Suppose the FUR require that each functional process that creates or updates persistent data moves a copy of each new or changed record to a log file. Assuming the movement of all logged records is identical, identify one Write data movement for which the log data groups are written in each functional process where logging is required.*

4.4.7 Error and confirmation messages

A business application is usually required to issue many types of error messages to its human functional users. All are generated from within the application software itself. Some messages are triggered as a result of human error whilst entering data. Others result from interpreting a message received from software in another layer or another peer-item, e.g. a return code that says ‘customer does not exist’. A failure to complete a functional process, e.g. an enquiry that does not find anything will normally return an error message.

The reader is strongly advised to read the Measurement Manual [1], section 3.5.11, for the definition of an error/confirmation message, for the rules on ‘error/confirmation messages and other indications of error conditions’ and for the related examples.

Notes on other specific cases:

- If a message to a human user provides other data than just confirming that entered data has been accepted, or that entered data is in error, then this message should be identified as an Exit and not as an error/confirmation message. (See also the rule for Exits in the Measurement Manual).

  EXAMPLE: In an order-entry functional process, for the automatic production of a letter when an order is not accepted due to a credit-check failure, identify 1 Exit.

- Any message that confirms that a data movement has been successfully processed, e.g. ‘Update OK’, should be treated the same as if it were another occurrence of an error/confirmation message, that is it should be regarded as accounted for by the single Exit ‘error/confirmation messages’.
- A function that provides for re-tries following an error condition should be regarded as a control command and ignored. But additional data movements resulting from an error condition in a functional process, e.g. for an alternative processing path, should of course be measured.

4.4.8 Fixed text

‘Fixed text’ is text that contains no variable data values. An enquiry which outputs fixed text, e.g. the result of pressing a button for ‘Terms & Conditions’ on a shopping web-site, should be modeled as having one Exit for the fixed text output.

4.5 Measurement of the size of functional changes to software

The reader is assumed to be familiar with the following sections of the Measurement Manual [1].

- The rules for measuring the size of changes to a functional process and for aggregating the sizes of changes, in section 4.3.1, and the following Example 1
- Section 4.4 on 'more on measurement of the size of changes to software', which includes the definition of 'modification (of the functionality of a data movement)'.

The approach of sizing changes to a functional process is illustrated by two examples.

4.5.1 Examples of functionally changed functional processes

EXAMPLE 1: The object of interest 'Employee' contains 'no. of dependents' as an attribute. It is decided to store more data about dependents. Consequently, an object of interest 'Dependent' is added and linked to Employee, data about individual dependents must now be included in the 'create employee' input, and the attribute 'no. of dependents' is removed from the Employee object of interest.

Old situation: There is persistent data about one object of interest

Employee (Emp-id, …, no. of dependents, …)

The 'create employee' functional process is (ignoring the detail of Reads for validation purposes that would most likely be needed in practice)

E	*Employee data*
W	*Employee data*
X	*Error/confirmation messages*

Total size 3 CFP

New situation: There is now persistent data about two objects of interest

Employee (Emp-id, …)	*('no. of dependents' removed)*
Dependent (Dep-id, Emp-id, …)	

The 'create employee' functional process will now be, noting the changes:

E	*Employee data*	*data movement modified (attribute removed)*
E	*Dependent data*	*data movement added*
W	*Employee data*	*data movement modified (attribute removed)*
W	*Dependent data*	*data movement added*
X	*Error/confirmation messages*	*(no change)[22]*

Hence the size of the functional change to the 'create employee' functional process is 2 Entries + 2 Writes = 4 CFP. Probably many more functional processes that must deal with the modified or added data groups must be functionally changed and the changes to these functional processes must also be measured.

Example 2: A bank statement shows the interest payable each month on positive balances. The algorithm to calculate the interest must be modified in some detail although no change is needed for the input data for the interest calculation. The bank statement is unchanged in content and layout but the data manipulation associated with one attribute is modified. The functional change is measured as 1 CFP for the modified Exit that shows the monthly interest.

[22] There may be new or modified error/confirmation message occurrences, but the data movement type is unchanged.

4.5.2 Data conversion software

When a new business application replaces an old system or replaces a manual system, it is frequently necessary to develop some software to convert data from the format or technology used in the old application to that needed by the new one, or to take on the data from the manual system. Such 'data conversion' software will typically be used once and then discarded. As such software serves the purposes of functional users it may, however, be considered as business application software and its FUR may be measured.

Whether or not it should be included in the scope for a particular measurement depends on the purpose of the measurement. If the purpose is to measure the work-output of a project team then the size of any data conversion software would be included in the scope. If the purpose is to measure the size of the delivered application, then data conversion software would be excluded from the scope.

4.5.3 Size of the functionally changed software

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

In example 1 of section 4.5.1 above, the net change in size of the 'create employee' functional process as a result of the functional change is an increase of 2 CFP (5 – 3 = 2).

In example 2 of section 4.5.1 above, the net change in size of the functional process exiting the monthly interest as a result of the functional change is 0 CFP (no added or deleted data movements, only a modified data movement).

REFERENCES

All COSMIC publications are available for free download from the portal of www.cosmicon.com

[1] Measurement Manual. (The COSMIC implementation guide to ISO/IEC 19761:2011), v4.0, April 2014

[2] ISO/IEC 19761:2003, Software Engineering – COSMIC – A functional size measurement method ISO/IEC 14143/1:2011 Software Engineering – COSMIC: a functional size measurement method, www.iso.org.

[3] Introduction to the COSMIC method of measuring software, v1.0, May 2014

[4] Quick reference guide to the COSMIC method for sizing business applications, v1.0.1, 2012

[5] ISO/IEC TR 14143/5 'Information technology – Software measurement – Functional size measurement – Determination of Functional Domains for use with functional size measurement'.

[6] Guideline for approximate COSMIC functional size measurement (under development, to be published 2014).

[7] The entity-relationship model – toward a unified view of data, Peter Chen, ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976.

[8] An Introduction to Database Systems, Date, C.J., Addison-Wesley, 1990.

[9] Data Models, Tsichritzis, D.C. and Lochovsky, F.H., Prentice-Hall, 1982.

[10] Unified Modeling Language Specification, March 2003, Version 1.5 (Version 1.4.2 of UML formal/05-04-01, published by the Object Management Group, which has been accepted as an ISO standard ISO/IEC 19501).

[11] Function Point Counting Practices Manual, Release 4.2, Part 4 'Appendices and Glossary', the International Function Point User Group, 2004.

[12] Guideline for Measurement Strategy Patterns, v1.0, March 2013

[13] Advanced and Related Topics, December 2007

[14] Guideline on Managing Non-Functional Requirements for Software (under development, to be published 2014)

[15] Guideline for ensuring the accuracy and repeatability of measurements, v1.0, February 2011

APPENDIX A – MAIN CHANGES IN V1.2 FROM V1.1 OF THIS GUIDELINE

Note. The nature of a change is indicated by

- 'Method' when the content of the COSMIC method was changed, or by
- 'Editorial' when the description of the method (but not the method itself) was changed
- 'Correction' when an error in v1.1 of this Guideline has been corrected.

Abbreviation: MUB – Method Update Bulletin

References in version 1.1.01	Nature of change	Comment
Foreword	Method	The 'limitations of the method' has been extended in accordance with the Measurement Manual v4.0.
-	Editorial	Text styles have been changed in accordance with the Measurement Manual v4.0.
-	Editorial	Simplified figure numbering has been applied, sequenced per chapter rather than per (sub-) section.
-	Editorial	www.gelog.etsmtl.ca/cosmic-ffp has been replaced by a reference to the portal of www.cosmicon.com.
-	Editorial	References to removed or replaced documents have been modified.
-	Method	'Peer component' has been changed into 'component' (MUB 9).
-	Editorial	Here and there 'entry' has been replaced by 'entering' so as to avoid confusion with 'Entry'.
1.2.1	Method	The more restricted COSMIC use of the ISO terms 'Functional User Requirements' (FUR) has been introduced, now meaning 'functional user requirements that are completely defined so that a precise COSMIC functional size measurement is possible'.
1.2.3	Method	The title and content of this section has been generalized to cover all Non-functional requirements (NFR) not just 'technical & quality requirements. The new definition of NFR has been added (MUB 10).
1.2.6	Correction	The footnote to the final bullet point of this section has been removed. Functionality, or changes to functionality, to help improve technical performance may be measured if defined to be within the purpose and scope of the measurement. See also 3.2 below.
2.0	Editorial	New text has been added to help improve understanding of an 'object of interest'
2.1	Editorial	The description of 'Conceptual' level of data models has been changed to more clearly distinguish it from the 'Logical' level'
2.6.1	Editorial	The last sentence of the Example in this section has been updated, but not exactly as was suggested in MUB 6.
2.6.2	Editorial	The examples have been re-structured and reduced from four to three so that they support the text more logically
2.6.3	Editorial	The text has been more clearly structured into three pieces of guidance for identifying an object of interest.
3.1	Method	The definition of 'layer' has been changed (MUB 9); text and examples have been changed accordingly.
3.1.2	Method	The description of layers has been changed according to MUB 9. Two

		examples are no longer relevant and the third example has been adapted.
3.2	Correction	The following statement in v1.1 is incorrect. 'Note that staff who maintain software programs or who maintain stored data or rules via utilities only available to those with computer software knowledge are NOT considered as 'functional users' of the software.' If the functionality maintained by these staff must be included within the measurement scope, then these staff must be functional users.
3.2.2	Method	This section has been deleted as it adds nothing to what is stated in the Measurement Manual.
4.1	Method	A new example has been added to emphasize the importance of distinguishing 'types' from 'occurrences'.
4.1.2	Method	The text has been changed according to the new definition of functional process (MUB11).
4.2.1	Editorial	The steps of analyzing the data and functional views of FUR in order to identify objects of interest, have been slightly re-sequenced in order to separate the analysis of transient data groups in the input and output components of functional processes, which does not involve iteration with the model of persistent data.
4.2.3	Correction	Example 1a shows a single Exit with attributes: Client ID, enquiry period start and end dates, number of orders. It is possible that this layout was specified in the FUR. But if no layout was specified in the FUR, then COSMIC rules would require that this un-normalized data record be normalized, as it contains attributes describing two objects of interest, 'time period' and 'client business in the time period'. (The clue that the record is non-normalized is that these two objects of interest have a 1:n cardinality. The solution now shows the correct size according to COSMIC rules. Example 1b) also has the same error of data analysis and has been corrected similarly, though in this case the size given was correct. The need for these corrections was first pointed out in MUB 6 issued on 30th April 2009.
4.2.3	Correction	Example 3c has been corrected as per MUB 6.
4.2.3	Correction	In Example 5 concerned with sub-types, the diagram in v1.1 of this Guideline was wrongly labelled as 'Object of interest type with its sub-types'. In fact it showed a possible physical database implementation of the sub-types. A new diagram, Fig. 4.1 (a) has been added to show the E/RA model of the entity and its sub-types. The figure previously labelled 4.1 now becomes 4.1 (b), labelled 'The probable corresponding physical database record structure.' Further, the Example of FUR 5 was wrongly measured, due to using the physical database record structure. This has been corrected.
'4.2.3	Correction	Example 6a) concerning the maintenance of parameter tables gives the sizes of the Create, Read Update and Delete functional processes as 3 CFP each, the same as in the previous version 1.1 of this Guideline. However, in the previous version, the FUR assumed 'no error/confirmation message'. This was a mistake. An Exit is needed for the C, U and D functional processes for error/confirmation messages to inform about validation failures or success.
4.2.3	Method	An expert system example (from MUB 8) has been added, to illustrate that COSMIC can measure some 'data-manipulation-rich' software'
4.4.1	Method	The definition of 'control commands' has been updated, according to the Measurement Manual v4'
4.4.3	Editorial	Text has been copied from the Measurement Manual to improve the explanation.
4.4.3	Correction	In v1.1 there is a statement: 'Note: a clock 'tick' or operator command that triggers the processing of a batch stream is a control command and is never measured in the business application software domain. Such a command launches the batch stream, not an individual functional process.' Whilst the second sentence is correct, we now recognize that a clock tick or operator command may effectively act as a triggering Entry for a functional process that requires no input data.
4.4.4	Correction	The text has been changed to bring it in line with the revised data uniqueness rules of according to the Measurement Manual. V4. The analysis of the examples, including their measurement does not change.

4.4.7	Method	The text has been changed to take into account the new definition and rules for ‘error/confirmation messages’ and other error conditions.
4.4.7	Editorial /Correction	The example of a requirement for logging within functional processes has been changed for clarity, which changes the measurement result for the logging functionality from one Write per object of interest to one Write per functional process.
4.4.8	Method	The description of ‘fixed texts’ has been added, in line with the Measurement Manual.

APPENDIX B - GLOSSARY

This glossary contains only terms and abbreviations that are defined in this guideline and that are specific to the business application software domain. For most terms and definitions of the COSMIC method, please see the Measurement Manual'[1]).

E/RA. Abbreviation for 'Entity Relationship Analysis' – see section 2.3 of this guideline.

Entity-type. Any physical or conceptual thing in the real world about which software is required to process and/or store data.

RDA. Abbreviation for 'Relational Data Analysis' – see section 2.5 of this guideline

Relation. Any set of data attributes.

Note: A 'relation in Third Normal Form' is a set of attributes describing a single object of interest, i.e. it is a synonym of a COSMIC 'data group'.

Statement of requirements. A document containing all the requirements for a piece of software, that is, Functional User Requirements and Non-Functional Requirements.

UML. Abbreviation for 'Unified Modeling Language' – see section 2.4 of this guideline.

APPENDIX C - 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 this guideline. 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 guideline, 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 guideline 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 organization 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 business application guideline 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 organizations exist that can provide training and consultancy or tool support for the method. Please consult the www.cosmicon.com web-site for further detail.