The COSMIC Functional Size Measurement Method

Versions 4.0.1/4.0.2

Guideline for Sizing Business Application Software

VERSION 1.3

May 2017

Acknowledgements

Authors and reviewers of this version 1.3		
Diana Baklizky TI Metricas Brazil	Peter Fagg Pentad Ltd United Kingdom	Cigdem Gencel Free University of Bozen-Bolzano Italy
Arlan Lesterhuis* The Netherlands	Dylan Ren Measures Technology LLC China	Hassan Soubra ESTACA France
Bruce Reynolds Telecote Research USA	Charles Symons* United Kingdom	Sylvie Trudel Université du Québec à Montréal Canada
Carlos Eduardo Vazquez Fatto Consulting Brazil	Frank Vogelezang Ordina The Netherlands	

*Authors of this document. For authors and reviewers of previous versions of this Guideline, please see the version 1.1 of May 2008 and version 1.2 of September 2014.

Version Control

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

DATE	REVIEWER(S)	Modifications / Additions
5 Dec 2005	COSMIC Measurement Practices Committee	First public version 1.0 issued
25 May 2008	COSMIC Measurement Practices Committee	Revised to v1.1 to bring in line with the COSMIC method v3.0
26 Sep 2014	COSMIC Measurement Practices Committee	Revised to v1.2 to bring in line with the COSMIC method v4.0
26 May 2017	COSMIC Measurement Practices Committee	Revised to v1.3 to bring in line with the COSMIC method v4.0.1 and Method Update Bulletins 13 and 14, and to make several additions and improvements – see Appendix A

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 v4.0.1 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 'First 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, or when using an Agile method, requires a good understanding of certain systems analysis methods, especially data analysis methods. One difficulty is that 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 aim 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 system and data analysis methods to concepts of the COSMIC model.

Readers of this guideline are assumed to be familiar with the COSMIC Measurement Manual, version 4.0.1 as enhanced by Method Update Bulletins 13 [21] and 14 [22] [3]. 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. The 'Introduction' 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.

---

[1] For a full description of what characterizes 'business application software,' see section 1.1.

[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.

[3] When verson 4.0.1 of the COSMIC method is enhanced by applying Method Update Bulletins 13 (as amended by the rule in section 2.6.2 of this Guideline) and 14, it will be published as version 4.0.2. This version 1.3 of this Guideline is intended to be compatible with version 4.0.2 of the Measurement Manual when it is pubilshed.

Measurers who have used a 'First generation' functional size measurement method and who wish to advance to the COSMIC method may carry a lot of experience specific to the earlier method. Some of this experience is relevant to COSMIC, and some is not. Much has to be re-learned. Someone converting from a First generation 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 the 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 4.2.5, example 7).

However, until more experience has been obtained, the Common Software Measurement International Consortium claims only that the detailed guidance of this document 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 in Appendix B.

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 guidance 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.

Changes for version 1.3 of this Guideline

The main changes for v1.3 are listed in Appendix A. In summary they are:

- Two changes to align this Guideline with v4.0.1 of the Measurement Manual (concerning the revised definition of Non-Functional Requirements and the revised 'Data Uniqueness' rules).

- In section 2.6.2, a new rule has been added for distinguishing data groups and objects of interest within a single functional process, based on Method Update Bulletin 13 but with further development and simplificaton. (This new rule will be added to the next version 4.0.2 of the Measurement Manual.)
- Some re-structuring and expansion of the previous section 4.2. A new section 4.2.2 discusses the relationship between the physical sequence of data entry and the COSMIC model. Section 4.2.3 concerns measuring the functionality to validate the input to a functional process. Section 4.2.4 concerns measuring the functionality to produce enquiries and reports and has new examples. The remaining examples of the previous section 4.2.3 are now in a new section 4.2.5.
- Corrections to the measurement of an example in section 2.6.3.
- Many editorial improvements to improve ease of understanding. These include describing the measurement results of all examples in a standard format.

It must be emphasized that in updating to versions 4.0.1 or 4.0.2 of the COSMIC method, no changes were made to the existing functional sizing principles or rules.

The COSMIC Measurement Practices Committee

Table of Contents

1 THE FUNCTIONALITY OF BUSINESS APPLICATION SOFTWARE 9

1.1 Characterization of business application software 9

1.2 Functional User Requirements 10

1.2.1 The meaning of 'functional' 10

1.2.2 The evolution of FUR in a typical software development project life-cycle 11

1.2.3 Non-functional requirements that evolve into software FUR 12

1.2.4 When the requirements lack sufficient detail to apply the COSMIC method 13

1.2.5 Measurement as a quality control on FUR 13

1.2.6 Who can specify FUR (and NFR)? 13

2 INTRODUCTION TO DATA ANALYSIS 14

2.1 Data modelling 'levels' 15

2.2 Data analysis principles and terminology 15

2.3 E/R Analysis 16

2.4 UML class diagrams (and use cases) 18

2.5 The normalization process of RDA 19

2.6 From entity-types, classes or relations to data groups and objects of interest 20

2.6.1 Input, output and transient data structures 20

2.6.2 Differing key attributes and frequencies of occurrence of data groups 21

2.6.3 Parameter (code) tables and objects of interest 23

2.6.4 Different 'kinds' of objects of interest 25

2.6.5 Summary of additional guidance for identifying data groups and objects of interest 26

3 THE MEASUREMENT STRATEGY PHASE 28

3.1 The purpose and scope of the measurement 28

3.1.1 Examples of purposes and scope 28

3.1.2 Software in different layers 29

3.2 Identifying the functional users and persistent storage 30

3.3 Identifying the level of granularity 30

3.4 Measurement patterns for business application software 30

4 THE MAPPING AND MEASUREMENT PHASES 32

4.1 Identifying functional processes 32

4.1.1 CRUD(L) transactions 33

4.1.2 Elementary parts of FUR: functional processes 33

4.1.3 Screen design style and functional processes 34

4.1.4 Physical screen limitations 34

4.1.5 Separate functional processes owing to separate functional user decisions 34

4.1.6 Retrieve and update of data in a single functional process 35

4.1.7 Drop-down lists and functional processes 36

4.1.8 Apparently inter-dependent functional processes 36

4.1.9 The 'many-to-many' relationship between event-types and functional process types 37

4.2 Identification of objects of interest, data groups and data movements 37

4.2.1 Overview of the process 38

4.2.2 Mapping between physical on-line data entry and data movements 39

4.2.3 Data movements needed to validate input data 41

4.2.4 Data movements in enquiries and reports 43

4.2.5 Other Examples 52

4.3 Sizing components of distributed business applications 57

4.3.1 Examples 57

4.3.2 Synchronous versus asynchronous communications ... 60
4.4 Other measurement conventions ... 61
4.4.1 Control commands ... 61
4.4.2 Authorization, help and log functionality ... 61
4.4.3 GUI elements ... 62
4.4.4 Menus and the triggering Entry ... 62
4.4.5 Applications processed in batch mode ... 62
4.4.6 Multiple sources, destinations and formats of a data movement – applications of the 'data uniqueness' rule ... 65
4.4.7 Error and confirmation messages ... 66
4.4.8 Fixed text and other fixed information ... 66
4.4.9 Time-out functionality ... 67
4.4.10 Date/time functionality ... 67
4.5 Measurement of the size of functional changes to software ... 67
4.5.1 Examples of functionally changed functional processes ... 68
4.5.2 Changes to fixed text ... 68
4.5.3 Data conversion software ... 69
4.5.4 Size of the functionally changed software ... 69
References ... 70
Appendix A – Main changes in v1.3 from v1.2 of this guideline ... 71
Appendix B - Glossary ... 75
Appendix C - COSMIC Change Request and Comment Procedure ... 76

1

THE FUNCTIONALITY OF BUSINESS APPLICATION SOFTWARE

1.1 Characterization of business application software

'Business application software' is distinguished by the following characteristics[4]:

- 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 section 2.2.2.
- 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). This underlying structure will not be visible to the human functional users. However, if the purpose of the measurements is to provide sizes as input to an estimating process, it may be necessary to define a separate measurement scope for each component, for example when different development or processor technologies are used for each component.

Nowadays it is increasingly difficult to define the boundaries of what is 'business application' software. Requirements for a new software system to serve a business may result in a 'hybrid' project that

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

assembles the system from a traditional business application, a web-site, a mobile app, some real-time data feeds and some infrastructure software, etc. It is the responsibility of the Measurer to decide on the parts of such a new system for which this Guideline is applicable.

Business application software is often implemented via general application 'packages' or with the aid of specialized software tools such as enquiry languages. These products are often highly flexible in the functionality they can provide and thus inherently complex. However, functional size measurement is normally concerned with measuring the size of the requirements for the implementation. The way of implementing the requirements, via a package or customer software, is irrelevant to this task.

Note that there are several Guidelines and other publications concerning measurement of special types of business application software, notably web software [15], data warehouse software [16], service-oriented architecture software [17], and mobile apps [18]. Also relevant are the Guidelines on early or rapid sizing [6], on non-functional and project requirements [13] and on the use of COSMIC FSM to manage Agile projects [19].

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 known completely 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 for 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. This topic is elaborated in section 1.2.3.

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 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. We can always infer the functional requirements of the software at any stage so that an approximate size can be measured [6], and the FUR can be measured when sufficient detail is known, even long after the software has been installed, implemented and is in regular use[5].

This means that at any point in the life-cycle, the Measurer must be able to derive the functional requirements and eventually the FUR from whatever artefacts of the software are available, whether they are an outline statement of requirements, an agreed 'specification' or 'design document', a series of Epics or User Stories (as in an Agile project) 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* functional requirements or FUR, though the difficulty of doing this may vary greatly 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 analyse 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' of an entity. FSM measures the separate logical functions of the software, not its design or 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

[5] 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.

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 Guideline on Non-Functional & Project Requirements ([13]), a 'non-functional requirement' (NFR) is defined as

> "Any requirement for a hardware/software system or for a software product, including how it should be developed and maintained, and how it should perform in operation, except any functional user requirement for the software. Non-functional requirements concern:
>
> - the software system or software product quality;
> - the environment in which the software system or software product must be implemented and which it must serve;
> - the processes and technology to be used to develop and maintain the software system or software product, and the technology to be used for their execution."

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 that will always remain non-functional include those to write the software in a given programming language, to execute it at a particular data centre or via a particular cloud service or that the software should have zero major defects in the first month of operation. Other requirements may evolve partly or wholly into software FUR.

EXAMPLE 1: A statement of FUR may define that a new system must enable a stockbrokers' clients to enquire on the value of their portfolio of investments over the Internet. Early in the project, a target response time is specified as a NFR. After further study, 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, real-time feeds of stock market prices to the portfolio enquiry system.

The original target response time requirement is still a valid NFR, but we now have an additional NFR (specified hardware) plus the FUR of some new application software to receive the feeds of stock market prices. The functional size of the software that must be built will inevitably have increased from the size before the consequences of the response-time NFR were worked out.[6]

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 4.1.7 and 4.4.3 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 for measurement purposes that some software FUR happened to start their life as apparently 'non-functional' system requirements.

---

[6] 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.

Where a measurement is being undertaken in the context of a software contract, and when a NFR evolves into additional, agreed, 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 [13] 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 Early or Rapid 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 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 [14]).

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 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.[7]

1.2.6 Who can specify FUR (and NFR)?

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 that is 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 not relevant to business users.

[7] Measurers and authors of software artefacts should cooperate in order to obtain good quality artefacts, 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')[8] 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.

Readers who are familiar with data analysis methods may wish to skip sections 2.1 to 2.5. However all readers are strongly recommended to read section 2.6 which discusses cases where conventional data analysis methods may be insufficient to correctly identify all the objects of interest in a given set of FUR, and therefore the corresponding data movements.

Given its importance, we remind readers of the definition of an 'object of interest'[9].

> "Any 'thing' in the world of the functional user that is identified in the Functional User Requirements, about which the software is required to move a data group either to or from a functional user, or to or from persistent storage. It may be any physical thing, or any conceptual thing.
>
> NOTE 1: 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.
>
> NOTE 2: There is nothing absolute about an object of interest. A 'thing' may be an object 'of interest' to a functional user via one or more functional processes, but not be an object 'of interest' to another functional user via other functional processes, even in the same software being measured."

Identifying the objects of interest in any data therefore involves asking what 'thing' or 'things' 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.*

(Note: the reader should be aware of the 'Warning on misleading attribute names' in section 4.2.1.)

---

[8] 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. (For the latter, see section 4.4.8)

[9] As amended by Method Update Bulletin 13, June 2016, with further simplifying changes.

2.1 Data modelling 'levels'

Data analysis methods usually define and distinguish 'data models' at different 'levels'[10] [7]. For our purposes it is sufficient to define the three modelling levels, as follows:

- 'Conceptual'. This level shows the main things[11] 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;
- 'Logical'. This level usually shows all the things, i.e. the objects of interest, in the real world and the relationships between them, about which data must be stored by the software to be built, and the data groups describing these things;
- '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 and terminology

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.

---

[10] 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'

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

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.

Another terminology problem is that the words used to describe (or quantify) the nature of the relationship between objects of interest vary with the data analysis method.

- Entity/relationship analysis describes the '*degree of the relationship'* (or its '*cardinality*') between two entity-types as 'one-to-many' or 'many-to-many', or 'many to one'.
- Relational data analysis, in the first normalization step states 'move *repeating data groups into separate relations*' This rule requires the data analyst to seek any 'one-to-many' relationships between a non-normalized relation and any 'repeating groups' within the non-normalized relation.

These various expressions are all used to describe what we call the relative *'frequency of occurrence'* of two different data group types (or two different object of interest types), an expression that we introduce in section 2.6.2. To help ensure understanding, here are some examples of 'different frequencies of occurrence'.

EXAMPLE 1: A holiday travel company's system can hold data for a 'booking' (or 'reservation') and data on the number 'n' of travellers covered by the booking. The records for the booking and the travellers have different frequencies of occurrence (one booking for 'n' travellers) both on the data input screen and in the database.

EXAMPLE 2: A report shows a list of of sales in each month for a given product and the total sales for the year of that product. The annual and monthly sales figures have different frequencies of occurrence (12 monthly figures for one annual figure).

EXAMPLE 3: Consider a message to transmit a file of data records to another system, The message consists of a header data group whose attributes describe the file, followed by the data records. The header and the data records have different frequencies of occurrence (one header for multiple occurrences of the data records).

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].

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 level) 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).

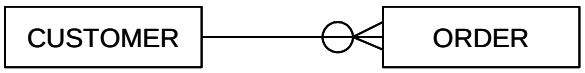


Figure 2.1 - E/R diagram showing the entity-types customer and order with a one-to-many relationship

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.

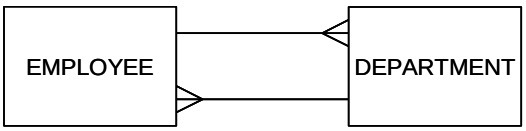


Figure 2.2 - E/R diagram showing entity-types and two one-to-many relationships

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.

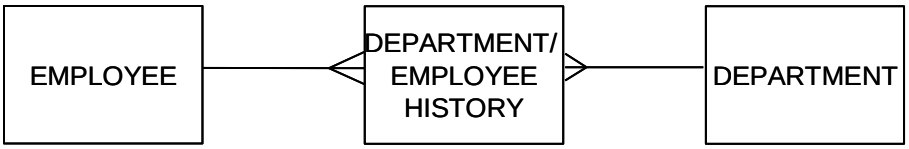


Figure 2.3 - The E/R diagram above resolved into two one-to-many relationships joined by an 'intersection-entity'

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 (in this case Employee ID and Department Name – see the Glossary for the definition of 'key'). The key attributes uniquely 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 move a data group either to or from a functional user, or to or from persistent storage'.

In the example above, the 'thing' has been named as the 'employee/department history'. Its key attributes indicate only that a relationship exists, or did exist, 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.

(Note In the following do NOT confuse the COSMIC concept of an 'object of interest' with the OO software 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 move a data group to or from a functional user, or to or from persistent storage'*. 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[12] 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, because 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 2 in section 4.2.5.

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 movements be known.

[12] More precisely a 'concrete class', not an 'abstract' class.

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*[13] *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 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 – describing 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 structured 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.

If relational database software is used, persistent data processed by business applications is usually stored in a normalized structure. The design of the data input and output of software processes is also usually in a normalized structure for logical and efficiency reasons, though the design will also be influenced by ergonomic factors for data input and output to humans.

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 table (by definition non-normalized).
2. Identify the key for the non-normalized data, i.e. the attribute-type or types that identify each row of the table.
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'.

[13] 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.

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

In almost all cases, entity-types that may be identified from E/R analysis at the level of granularity of functional processes, classes shown on UML class diagrams, and the subjects of relations in 3NF identified from RDA will be COSMIC objects of interest. But in some practical cases, data analysis methods do not 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 four sections and the additional criteria are summarized in section 2.6.5.

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 moved in 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 data about 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.

However, many enquiry 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 move structures of *transient* data groups. (Our three conventional data modelling methods are not normally used to identify the objects of interest described by transient data groups.)

EXAMPLE. Suppose a database in which each customer is allocated a 'customer-category', (where the latter is coded P = Personal, R = Retailer or W = Wholesaler). There are FUR to develop an enquiry to calculate and display:

- *the total value of goods sold in a given time period by customer-category, and*
- *the total value of goods sold to all customers in the time-period.*

The start and end dates of the time-period must be entered and must also be output to enable understanding. There is no requirement for an error or confirmation message.

The data needed for this enquiry will be taken from a database of completed single-item orders, where each order could have attributes, e.g. order ID, product ID, customer ID, customer-category code, order value, and date payment received. This date is the criterion for deciding if the product has been sold.

Figure 2.4 shows an example output from this enquiry:

Sales by Customer-category

Period: Jan 01 2017 to Mar 31 2017

Customer-category	**Sales ($k)**
Personal	159
Retailer	2,014
Wholesaler	748
Total	**2,921**

Figure 2.4 – Display of enquiry on sales by customer-category for the given time period

Discussing first the output of this enquiry, two levels of aggregation of sales data can be distinguished. The sales per customer-category and the total sales have different frequencies of

occurrence. This indicates that we have transient data describing two objects of interest and hence two Exits. The two objects of interest are:

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

As a cross-check on this conclusion, E/RA would say that 'the goods sold to a customer of any one category in a time-period' is a different 'thing' from 'the goods sold to all customers in the time-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 move a data group either to or from a functional user, or to or from persistent storage, 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-category 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-categories is a repeating group and is therefore in a different relation from that of data about 'all customers'.

Discussing next the input to this enquiry – the time-period – this is also a transient data group moved by the triggering Entry of the enquiry. (No data is held about the time-period.) This data group describes an object of interest that we could call 'the time-period of the enquiry'. Data defining the time-period must also be output so that the user can understand the sales values. The question is: should the time-period also be counted as a separate Exit when it is output? For the answer to that see the next section 2.6.2.

2.6.2 Differing key attributes and frequencies of occurrence of data groups

The analysis of the example in 2.6.1, which has so far resulted in two Exits, illustrates another fundamental result that arises from both E/RA and from RDA, namely that 'Differing identifying key attributes, resulting in different frequencies of occurrence, always indicate differing data group types'. The statement applies to Entry, Exit, Read and Write data movement types, i.e. it is not limited to transient data in input and output.

The following rule should be used to distinguish data groups and hence objects of interest, especially when measuring functional processes that produce transient data, as in enquiries and to produce reports[14].

RULE - Identifying different data groups (and hence different objects of interest) moved in the same one functional process
For all the data attributes appearing in the input of a functional process: 1. sets of data attributes that have different frequencies of occurrence describe different objects of interest; 2. sets of data attributes that have the same frequency of occurrence but different identifying key attribute(s) describe different objects of interest; 3. all the data attributes in a set resulting from applying parts 1 and 2 of this rule belong to the same one data group *type*, unless the FUR specify that there may be more than one data group type describing the same object of interest in the input to the functional process (see Note 3) This same rule applies for all the data attributes appearing in the output of a functional process, or all that are moved from a functional process to persistent storage, or all that are moved from persistent storage into a functional process.

[14] A first version of this new rule was published in Method Update Bulletin 13 (June 2016). Further work has shown that this first version needed to be generalized and simplified. The new rule will be included in the next version of the Measurement Manual.

NOTE 1. It can be helpful when analyzing complex output, e.g. reports with data describing several objects of interest, to consider each separate candidate data group as if it were output by one separate functional process. Each of the data group types identified this way must also be distinguished and counted when measuring the complex report. See the analysis at the end of this section 2.6.2 of the example first described in section 2.6.1. See also the analysis of the examples 4 and 5 in section 4.2.4.

NOTE 2. Examining how data attributes are physically grouped or separated on *input or output* may help distinguish different data group types, but cannot be relied upon to distinguish them. As an example, two or more sets of data attributes occuring on the same input or output that are physically separated for aesthetic reasons or for ease of understanding, will belong to the same one data group type if they satisfy the rule above.

NOTE 3. For definitions, other guidance and other examples of identifying objects of interest and data groups, see the Measurement Manual [1], section 3.3.

Also, see section 3.5 of the Measurement Manual for the definitions, principles and rules for the *data movements* that move data groups, and section 3.5.7 and 3.5.11 for exceptions to these rules for data movements, as per rule 3 above.

Applying Part 1 of the rule to the example of 2.6.1, we can now complete its analysis and measurement.

Note In the table below and in all other similar tables in this Guideline, the following abbreviations are used:

- **'DM' = ' Data Movement'**
- **'# Oc's' (if shown) = 'Number of Occurrences'.**
- **'ID' = 'Identification', i.e. a unique code, name or description.**

DM	***Key Attributes***	***Data Group***	***# Oc's***
Entry	*Start date, End date*	*Time-period definition*	*1*
Read	*Order ID*	*Order details*	*Many*
Exit	*Start date, End date. Customer-category*	*Sales per Customer-category in the Time-period*	*3*
Exit	*Start date, End date*	*Sales to all Customers in the Time-period*	*1*

Total size – 4 CFP.

Comment on the analysis:

- Applying the rule above tells us that we must have two Exits as the frequency of occurrence (and the key attributes) of 'Sales to all Customers' differ relative to 'Sales per Customer-category'.
- The key attributes of the 'Time-period definition' data group on the output (Start date' and 'End Date') and the total 'Sales to all Customers in the Time-period' ($2,921K), both occur once on the report and have the same key identifying attributes (Start date and End date). The rule above tells us that they are both attributes of the same one data group, hence there is one Exit to move these attributes.
- The output heading 'Sales by Customer-category' is fixed text; it is not counted as an Exit – see section 4.4.8.

The value of Note 1 to help distinguish data groups can be illustrated by the simple example discussed above in this section, whose output on a single report is illustrated in Figure 2.4.

EXAMPLE: Suppose the two sales figures of this example 'Sales per Customer-category in the Time-period' and 'Sales to all Customers in the Time-period' were required to be output by two separate functional processes on separate reports, instead of on the same one report as in Figure 2.4.

The report showing 'Sales per Customer-category in the Time-period' would have two Exits:

DM	Key Attributes	Data Group	# Oc's
Exit	*Start date, End date. Customer-category*	*Sales per Customer-category in the Time-period*	*3*
Exit	*Start date, End date*	*Time-period*	*1*

The report showing 'Sales to all Customers in the Time-period' would have one Exit:

DM	Key Attributes	Data Group	# Oc's
Exit	*Start date, End date*	*Sales to all Customers in the Time-period*	*1*

We observe that the two different data group types shown as output on the report of Figure 2.4 from a single functional process become three different data group types when reported separately by two functional processes. (The three data group types have different attributes and the two functional processes output a total of three Exits). However, there are only two unique sets of key attribute combinations, regardless of whether the data are reported separately by two functional processes or together by one functional process on the one report.

This Note 1 can be helpful when measuring functional processes that produce output with multiple data groups, (see section 4.2.4, Example 5.) This section has more examples of measuring the number of Exits output from an enquiry or reporting functional process.

2.6.3 Parameter (code) tables and objects of interest

As described in Note 2 to the definition of an object of interest (see above), 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 them, e.g. a system administrator.

We will illustrate the cases that can arise with some simple examples. See section 4.2.5, example 3, for more examples.

EXAMPLE 1: Suppose an order-processing application for a manufacturer that supplies lighting goods in bulk to individuals and companies. The application 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-credit-limit, customer-category-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 via an error/confirmation message.

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 be measured at 4 CFP, as below.

DM	Key Attributes	Data Group
Entry	*Customer ID*	*Customer data*
Read	*Customer ID*	*Customer data*
Write	*Customer ID*	*Customer data*
Exit	*Error ID*	*Error/confirmation messages*[15]*.*

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

The Read of the customer records is needed to check that there is not already a record for the customer whose data are being entered.

EXAMPLE 2: Notice the attribute 'customer-category-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-category'. The user then selects the appropriate description and the corresponding customer-category-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-category-code as an attribute of 'customer'. There is no requirement to store data about customer-category in the customer record, so customer-category 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-category descriptions does not affect the size of the functional process, which is still 4 CFP, as in Example 1.

The pairs of values of 'customer-category-code' and 'customer-category-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'.)

It does not matter for the size of the functional process to enter data about a new customer whether customer-category codes and descriptions are hard-coded, or stored in maintainable tables. That is an implementation issue, not part of the FUR.

However, supposing customer codes and descriptions are stored in maintainable tables. The functionality needed for such parameter table maintenance would have functional processes to create new customer-category codes and descriptions, and to update, delete and read them. In other words, they process data about customer-category. Customer-category 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-category in these functional processes must be identified. (Examples of the functional processes that might be needed to maintain parameter tables are given in section 4.2.6, Example 3.)

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

EXAMPLE 3: Suppose this same order-processing application also stores data about this customer-category 'thing' in addition to the customer-category code, e.g. 'customer-category-description', 'customer-category-order-value-discount-%', 'customer-category-payment-terms, etc.

Customer-category 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-category).

In this Example 3, therefore, when entering data about a new customer, the entered customer-category-code must be validated against the already-stored corresponding attribute of the object of interest 'customer-category'[16]. So the simplest functional process to enter data about a new customer, as described in Example a), must now in this Example c), include a Read and an Exit to display the valid customer-category codes and maybe other attributes of the customer-category object of interest, so that the user can select the correct code.

Note that in this same functional process to enter data about an individual customer, no separate Entry should be identified for when the selected customer-category-code is entered, because it is

[16] A GUI drop-down list that enables an order-desk user to select a valid customer-category 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-category' object of interest.

being entered as an attribute of (i.e. data about) a customer (we are not entering data about customer-category in this functional process).

The functional process to enter data about a new customer, when customer-category is an object of interest has the data movements shown below. The total size has now increased to 6 CFP.

DM	*Key Attributes*	*Data Group (Attributes)*
Entry	*Customer ID*	*Customer data (includes customer-category code)*
Read	*Customer ID*	*Customer data*
Read	*Customer-category code*	*Customer-category (code, description)*
Exit	*Customer-category code*	*Customer-category (code, description)*
Write	*Customer ID*	*Customer data*
Exit	*Error ID*	*Error/confirmation messages*

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-category is an object of interest if it has its own attributes (as in Example 3) or it is not an object of interest (as in Examples 1 and 2). In Examples 1 and 2, customer-category-code is simply an attribute of customer.
- To the non-business functional users who use functional processes to maintain customer-category codes and descriptions as well as other system parameters, customer-category is an object of interest in those functional processes, as in Example 2.
- 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 3 in section 4.2.6.

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.

This example also illustrates that a COSMIC analysis of a functional process as shown in the two tables in this section only results in a *list* of the data movements. This list is not a model of the *sequence* in which data are physically moved. To illustrate: the process in Example 3 starts with the entry of the first attribute of the Customer data group to be moved by the triggering Entry. But this Entry is not physically completed until its final attribute is entered and validated. This happens just before the Write of the Customer data group. See section 4.2.2 for a fuller discussion of this relationship between the physical steps of a functional process and the list of data movements needed for a COSMIC size measurement.

2.6.4 Different 'kinds' of objects of interest

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.

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.

- '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 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 c), where 'customer-category' is an object of interest with a few important attributes, e.g. 'customer-category-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.

  EXAMPLE: suppose a requirement to enquire on the value of sales to a given customer in a given month, where both the customer ID and the month are entered. The sales amount is calculated from all orders placed by the customer in the month. Data is stored about 'customer' and about 'order', but no persistent data is stored about any 'sales' figure or any 'month'. Data analysis as in section 2.3 would show that 'sales' is an 'intersection-entity' between 'customer' and 'month'. (To be more precise, the objects of interest of this enquiry should be named 'customer of the enquiry', 'month of the enquiry' and 'goods sold to the given customer in the given month'. For convenience, we abbreviate these names to 'customer', 'month' and 'sales'.)

- 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 an international transport 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 an attribute for the asset register.

2.6.5 Summary of additional guidance for identifying data groups and objects of interest

The effect of these cases given in sections 2.6.1 to 2.6.4 is that we need to add some guidance for recognizing an object of interest to that given in the Measurement Manual, as follows

To understand whether a 'thing' is an object of interest or not:

a) 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 move a data group either to or from a functional user, or to or from persistent storage;
b) recognize that whether a ‘thing’ is an object of interest or not depends on the (type of) functional user for whom the FUR are specified;
c) recognize that many ‘things’ must be identified as objects of interest about which data are stored that would not appear as ‘primary’ entities on an entity-relationship diagram, nor be the subject of relational data analysis;
d) 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 usually but not necessarily objects of interest[17]. The transient data of the input and output parts of functional processes must be analysed as well as the persistent data to identify all the objects of interest.
e) use the new rule given in section 2.6.2, namely that data to be entered into a functional process must be analysed into groups with different frequencies of occurrence and different identifying key attributes. Such groups describe different objects of interest. The same is true for data groups that must be read from or written to persistent storage, or that must be output by a functional process. Do not rely on how the data may be physically grouped for this analysis.

[17] An example of an entity appearing on an entity-relationship diagram that is NOT a candidate object of interest could be when an entity diagram is drawn at a high level of granularity e.g. for the purpose of designing an application architecture. For example, the diagram might show ‘Product’ as an entity whereas more detailed examination of the FUR might show that different sub-types of Product must be distinguished and that these are the ‘real’ 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 defining 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 individual piece of software to be measured;
- The *level of granularity* of the artefacts of the software (e.g. a statement of requirements) 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 1: 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 2: If the purpose of the measurement is to determine the software project productivity (or another 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 3: Suppose, as in the previous example, the purpose of the measurement is related to development project performance measurement or estimating, but in this case the application must be distributed over different technical platforms, and it is known that development productivity varies with the platform. Almost inevitably, the Measurer will need 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.

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

EXAMPLE 4: 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 measurement 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[18].) An average productivity for replacing the whole portfolio can then be used to determine the replacement cost.

EXAMPLE 5: 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 measurement 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.

EXAMPLE 6: Over 90% of the FUR for a new business application will be implemented by a standard package, and the remaining 10% by custom software. The goal is to estimate the effort to implement the whole FUR. For the scope of the FUR that will be satisfied by the package, it would be advisable to consult with the package supplier on how to estimate the effort. (Measuring the functional size of these FUR may or may not be useful; measuring the size of the package is almost certainly irrelevant to the goal.) The scope of the 10% of the FUR that will be implemented by custom software can be measured in the normal way and effort estimated using a productivity appropriate for the technology of the custom software.

An important point to ensure, whenever the purpose is related to performance measurement or estimating for a software project, is that the measurement scope of each piece of functionality 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.
- Alternatively, 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).

[18] See the document 'COSMIC Guideline for Early or Rapid Functional Size Measurement using approximation approaches [6].

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 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 and persistent storage

For a piece of business application software or for one of its major components (of a defined measurement scope), 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[19] components of the same application that are required to exchange or share data with the major component being measured (see also section 3.4);
- 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;
- minor components, e.g. re-usable SOA components, if the software calls for their service [17]

Note that staff 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 measurement scope.

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.

As a consequence, the Measurer must first 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 Early or Rapid Functional Size Measurement using approximation approaches' [6] which describes various approaches for approximate sizing of requirements at high levels of granularity.

3.4 Measurement patterns for business application software

Although many different strategies are possible, any one organization probably needs only a limited number of different strategies ('measurement patterns').

A measurement pattern defines a standard combination of parameters that must be determined in the Measurement Strategy phase of a COSMIC measurement process. A measurement pattern also provides a template for drawing the 'context diagram' for the software to be measured. Regular use of a limited set of measurement patterns should therefore speed up the Measurement Strategy phase, as well as ensure 'like-for-like' size comparisons. The three most common patterns for business application software are:

[19] 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).

- On-line business applications considered as a whole (‘Whole application’);
- Batch processed business applications (‘Whole Application’);
- On-line business applications considered as components of a multi-tiered implementation.

For more on this, see the Guideline for ‘Measurement Strategy Patterns’ [12].

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 next 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] and with Method Update Bulletin 14. The steps of this identification process are:

- Identify the separate events in the world of the functional users to which the software being measured must respond to – the 'triggering events';
- Identify which functional user(s) of the software respond to each triggering event by generating one of more data group(s);
- Identify the triggering Entry (or Entries) that move the data group(s) generated by each functional user 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 when 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 from a file of employee data, by any combination of three input parameters, namely, 'age', 'gender' and 'education level'. The three parameters must also be output. 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 key 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) with the third parameter value blank;*
- *three instances of a single parameter (age, gender, education level) with the other two parameter values blank.*

It is vital to recognize that these seven possible combinations of the enquiry input parameters all 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 data movements of this functional process are given in the table below, giving a size of 5 CFP.

DM	Key Attributes	Data Group
Entry	*Age, Gender code, Education-level*	*Employee selection parameters*
Read	*Employee ID*	*Employee data*
Exit	*Age, Gender code, Education-level*	*Employee selection parameters*
Exit	*Employee ID*	*Employee name*
Exit	*Error ID*	*Error/confirmation message*

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 in other words a state transition). In addition, data about each object of interest must invariably be read and maybe listed (both of which involve a 'state inspection', not 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 inspections or 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.

EXAMPLE: FUR might specify separate functional processes to update the data of a person for each of the five possible transitions that he/she may make 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 read persistent data to produce (or create) transient data to be output in Exits. (See section 4.2.3, 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 starts processing on receipt of a data group moved by its triggering Entry data movement, 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.

The second bullet 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 the 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 are first read and displayed. This is followed by the 'update' functional process in which the data to be updated is entered and made persistent.

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 changed data (i.e. modified, 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.

EXAMPLE: The task is to update an employee's data record, where the user knows the employee's name, but not the unique employee ID. There may be more than one employee with the same name. The analysis leads to the functional processes FP1, FP2 and FP3 (assuming error/confirmation messages in the FUR):

First, in Functional Process 1 ('FP1'), the user is invited to enter the employee's name.

DM	*Key Attributes*	*Data Group*
Entry	*Employee name*	*Employee name*
Read	*Employee ID*	*Employee data*
Exit	*Employee ID*	*Employee summary data (e.g. only name and ID, and sufficient other data to choose the employee whose data must be updated).*
Exit	*Error ID*	*Error/confirmation message (may be necessary if there is no employee with the entered name)*

Size of FP1 = 4 CFP

Second, the user can now, in FP2, choose the particular employee from the list and, by pressing e.g. an 'Update employee data' button, display the data that needs to be updated:

DM	Key Attributes	Data Group
Entry	*Employee ID*	*Employee ID (= selecting the desired employee)*
Read	*Employee ID*	*Employee data*
Exit	*Employee ID*	*Employee data (detailed form, all attributes)*
Exit	*Error ID*	*Error/confirmation message (could be necessary if the user is not allowed to update the employee data. In this case the Entry would be followed immediately by this Exit.)*

Size of FP2 = 4 CFP

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

DM	Key Attributes	Data Group
Entry	*Employee ID*	*Updated employee data*
Write	*Employee ID*	*Updated employee data*
Exit	*Error ID*	*Error/confirmation message (arising from validation failures on data entry)*

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 (via FP3) 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 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. The same may apply when a functional process receives input data from another system for immediate update.

4.1.7 Drop-down lists and functional processes

The examples of section 2.6.3 showed that entering the value of a single data attribute (called, say 'Z') via a drop-down list may or may not need more data movements than just the Entry of the data group that includes the attribute Z. Generalizing the description of two examples in that section.

EXAMPLE 1: The value entered for a data attribute Z of an object of interest A is selected from a drop-down list of valid values. The values in the drop-down list are either hard-coded, or obtained from a 'parameter table' of codes and maybe also descriptions. Measure only the one Entry of the whole data group of which Z is one attribute. The use of a drop-down list does not add to the size.

EXAMPLE 2: The valid values of the attribute Z are obtained from the values of an attribute of another object of interest B. In this case measure an additional Read and Exit for the display in the drop-down list of the valid values that may be selected for the attribute Z, because these values come from this other object of interest B.

In other cases, the requirement for a drop-down list may result in an additional, separate functional process. This can be illustrated by the following example.

EXAMPLE 3: Suppose, as for example 2, the valid values of the attribute Z of the object of interest A are obtained from an attribute of another object of interest B. In this case, however, the functional user is offered a choice of two ways of entering the value of the attribute Z, depending on whether the functional user knows the valid value to be entered or needs to select a valid value from a list.

For the case where the functional user enters the value of the attribute Z directly (i.e. not via a drop-down list), identify one additional Read (of object of interest B) in the functional process (say 'FP1') for the validation of the value of the attribute Z that has been entered,

AND for the alternative case where the user needs to select a value from a valid list, recognize a separate functional process (say 'FP2') for the optional display of the list of valid values of the attribute. Why a separate FP2? Because the functional user must make a conscious, separate decision to display the valid values of attribute Z.

This separate functional process FP2 has size 3 CFP (1 Entry, 1 Read, 1 Exit) and should be measured only once for the whole application. There will be as many functional processes of type FP2 in the application as there are attributes for which valid values may be optionally displayed in this way. Use of this optional alternative is similar to using a help function from any screen.

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 to get an error/confirmation message. Therefore, no Exit is identified for 'error/confirmation messages' in relation to entering this attribute Z.

Example 1 arises commonly in web-based business applications intended for use by the general public where quality control of the entered data needs special attention.

Example 3 arises where facilities are provided both for experienced users and for inexperienced users.

4.1.8 Apparently inter-dependent functional processes

(See also section 3.2.7, 'Independence of functional processes sharing some common or similar functionality: reuse' 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 <u>including</u> 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'. It might be triggered by a bank clerk requesting the statement from a terminal at the bank counter on behalf of the account holder. It might also be triggered 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 objects of interest and data groups' and 'Identifying data attributes' of the Measurement Manual.

4.2.1 Overview of the process

We assume that for the given 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. Your preferred data analysis method should help identify the objects of interest of 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, if not part of the documentation, construct a logical data model of the objects of interest of the persistent data 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 2.6.4 for identifying objects of interest for different types of functional users.
- Then analyse the input, process and output phases of each functional process to identify the data groups to be sent to, or obtained from, persistent storage using the rule of section 2.6.2.
- Use the additional guidance and examples in sections 4.2.3 and 4.2.4 respectively to identify the transient data groups in the input and output in each functional process that outputs derived (i.e. non-persistent) data.

Be prepared to iterate around these three 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').

All of the above must be subject 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 identify at least one Entry. The first Entry triggers its execution as well as moving a data group describing the triggering event.
- 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.
- Each functional process must have an 'outcome', i.e. at least one Write or one Exit.
- Identify one Exit for error/confirmation messages in each functional process if required by the FUR to be output to a human functional user. (See also the Measurement Manual, section 3.5.11 and section 4.4.7 of this Guideline.)

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 1: 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.

(However, this case may not be so simple. All the considerations of whether 'grade' might be another object of interest for these FUR, with its own attributes, e.g. 'grade salary', apply as in the examples discussed in sections 2.6.3 and 4.1.7.)

EXAMPLE 2: 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 labelled '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 time-period',*
- *'ID of product sold to customer in time-period',*
- *'Start date of time-period of sales',*
- *'End date of time-period of sales'.*

In practice, no-one has the time to write such long, but unambiguous, data attribute names and in this context, the meaning of the short labels 'Customer ID' Product ID', etc. is clear. But these same labels could have an entirely different meaning in a different context (e.g. 'Customer ID' could mean 'the ID of the customer placing this order', or 'the ID of the addressee of this invoice', etc.).

4.2.2 Mapping between physical on-line data entry and data movements

The sequence in which a user physically enters data to an on-line business application system rarely maps directly to a COSMIC Generic Software Model of the data movements in a functional process. If you try to measure a size based on the physical sequence of data moving in and out of a data entry screen, you will probably not measure the size in CFP correctly.

The table below compares a possible physical sequence of actions to enter data for a simple example of a 'Create' functional process, against the data movements that must be identified and counted for a CFP measurement. The FUR for the functional process are as follows:

EXAMPLE: A process is required for a human user to enter and store data about a new employee. The dialogue proceeds as follows:

- *The user selects the process from a menu and then enters the employee's name and Social Security Number (SSN). The SSN is the unique employee identifier.*
- *If the SSN is not valid, the software issues an error message. The user may repeat this step two more times but if these fail, the user is returned to the menu.*
- *If the SSN is valid, the software checks if an employee already exists in the employee master file with that name and SSN.*
- *If an employee is found with the entered name and SSN, (Implying the employee's data has already been entered) the software displays that employee's data and issues an error message. The user is then returned to the menu.*
- *If the name and SSN are not known in the system, the user continues by entering the new employee's title, address, date of birth, sex, and marital status.*
- *The software validates the entered data for format errors, etc., generates a unique employee ID, stores the new employee record and confirms that the process has been completed by displaying a screen for entry of the next new employee's data.*

Note: in the table below 'DM Measured' is shown in the row where the data movement is first observed in the execution of the functional process (not when the movement is completed).

Human user or software actions	Data crossing the boundary	Comment	DM Measured
User selects 'Enter data for new Employee'' from the menu	*Message to software: 'Display screen for new employee data entry'*	*Menu selection is not measured. (Except see Section 4.4.4)*	-
Software displays formatted data entry screen	*Text field headings for entry of data for a new employee*	*Ignore 'blank' data entry screens. Only the entered data are measured*	-
User enters the new employee title and Social Security Number (SSN).	*New employee name New employee SSN*	*These are just the first two new employee attributes of the triggering Entry; processing of this Entry is not complete yet.*	*Entry (Employee)*

Software issues an error message if the SSN is invalid	*Error message, e.g. 'Invalid SSN'*	*Ignore the user's repeated attempts to enter a valid SSN.*	*Exit (Error/Conf. message)*
Software searches the file of existing employees to find if the new employee name and SSN already exist	-	-	*Read (Existing Employee)*
Software issues a warning message if an existing employee is found with the same name and SSN	*Warning message (e.g. 'Employee already exists in system')*	*This is another occurrence of an error/confirmation message. It has already been measured. Ignore.*	-
If an existing employee is found with the same name and SSN, the software displays their details,	*Existing employee ID, title, name, address, SSN, date of birth, sex, marital status*	*This output enables the user to check that this employee's data really has already been entered.*	*Exit (Existing Employee)*
User closes screen showing data for existing Employee. Software re-displays menu	*'Close screen' command (in). Menu displayed (out)*	*Closing the screen is a 'control command'. Ignore. (See section 4.4.1)*	-
User enters the other new employee attributes.	*New employee title, address, date of birth, sex, marital status,*	*The system validates the entered data for field formats, etc. (Data manipulation)*	-
Software may issue other validation error messages	*Example: 'Error: address postcode is invalid'*	*More occurrences of an error/confirmation message Ignore.*	-
Software generates a unique Employee ID	-	*Data manipulation*	-
User presses 'Enter' when all employee attributes have been validated	*'Enter' command*	*A control command. (Think of this as completion of the Triggering Entry)*	-
Software makes the employee record persistent	-	-	*Write (Employee)*
Software signals successful completion by displaying a screen for entering the next new employee data	*Text field headings for entry of data for a new employee*	*Another occurrence of an error/confirmation message. Ignore.*	-

In the conventions of this Guideline, we would present the analysis of these same FUR as below.

DM	***Key Attributes***	***Data Group***
Entry	*New Employee SSN*	*New employee details*
Read	*Existing Employee SSN*	*Existing Employee details*
Exit	*Existing Employee SSN*	*Existing Employee details (indicating the employee data already exists in the system)*
Write	*New Employee SSN*	*New employee details*
Exit	*Error ID*	*Error/confirmation message*

The total size of this one functional process is 5 CFP.

4.2.3 Data movements needed to validate input data

The processing of an Entry data movement usually needs functionality to validate the entered data. Validation functionality that consists only of *data manipulation*, e.g. checking an attribute value for its format or that it is within a permitted range, can be *ignored*.

However, for some attributes in the data group moved by an Entry there may be FUR that the value entered for an attribute must be either:

- selected from an existing list of valid values,
- or checked against one or more value(s) that are stored or available somewhere to the functional process being measured.

Both these cases may result in the need to identify other data movements in addition to the Entry, to ensure that valid data is entered. The purpose of this section is to describe a process for identifying these additional data movements, if any are required by the FUR.

Consider Figure 4.1a, where the need is to ensure the entry of valid data for an attribute 'Z' of a data group 'P' moved by an Entry. The value of 'Z' must be either:

- selected from a list of values of an attribute 'Y' in an existing data group 'Q',
- or validated against a value 'Y' of a data group 'Q' that is available somewhere to the functional process being measured.

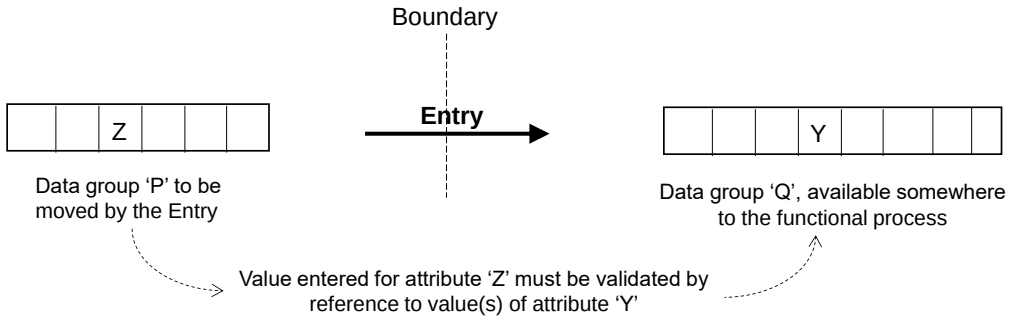


Figure 4.1a – Validation of a data attribute 'Z' of a data group 'P' moved by an Entry

The process to identify any other data movements needed to ensure valid data is entered for the attribute 'Z' is as shown in Figure 4.1b,

Note some important points about this process.

- The process is NOT needed for the validation of any attribute Z that only involves data manipulation, e.g. format or range checking.
- The process must be followed for <u>all</u> attributes 'Z' in any data group 'P' whose entered values require to be validated against data available to the software being measured. ('Available somewhere' means that the data is either stored persistently in the software being measured or is available via a request to another piece of software.)
- If the entry of valid values of the attribute Z is via a drop-down list that may be used optionally by the functional user, then (as in Example 3 in section 4.1.7), the validation involves a separate functional process, in addition to the functional process being measured. This additional functional process must have one triggering Entry, plus the data movements obtained by applying the steps of Figure 4.1b to the additional functional process.

The abbreviations used in Figure 4.1b are:

'FP'= the functional process being measured,

'OOI' = object of interest,

'End' indicates the process is finished and no more data movements need be counted.

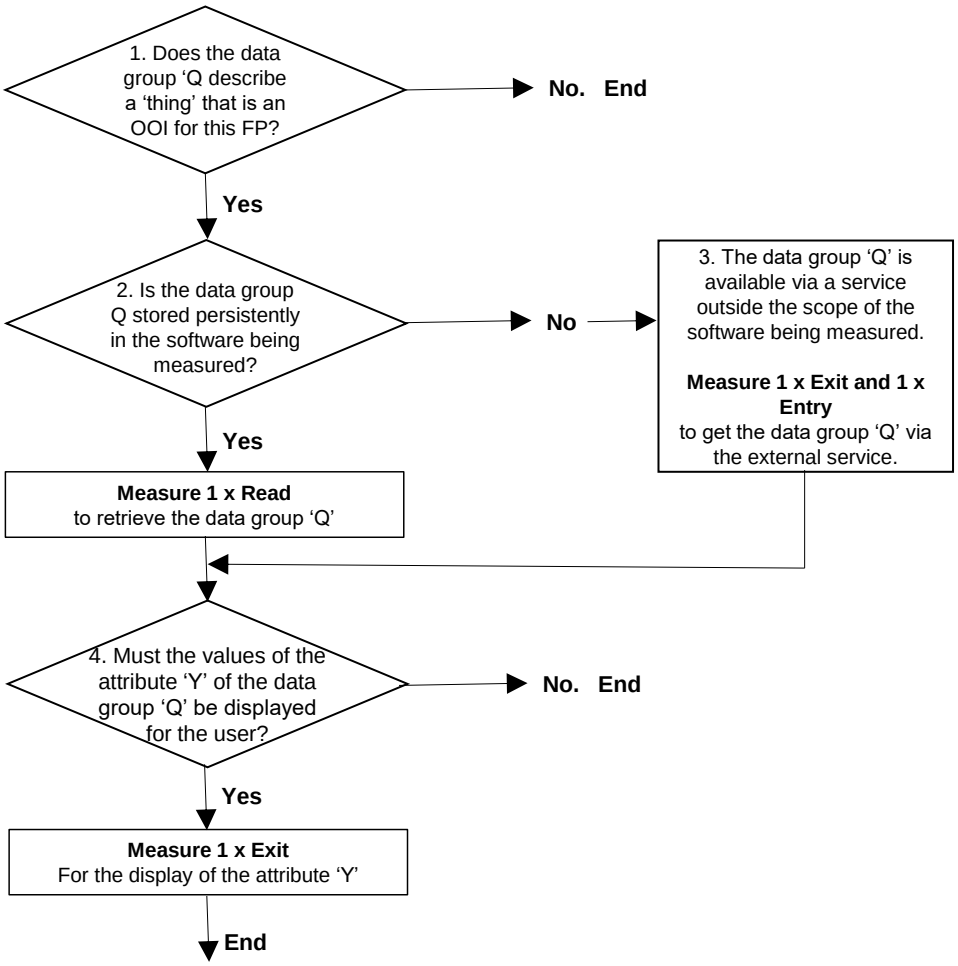


Figure 4.1b – The steps of a process to decide on the number of data movements that might be needed to validate the attribute 'Z' shown in Figure 4a

EXAMPLE 1 for Step 1. When selecting a Country name from a drop-down list of valid Country names, as part of entering an address, 'Country' is not an object of interest for this functional process (only Country name is held). No other data movements need be measured.

EXAMPLE 2 for Step 2. When entering data about an order, the software must check that the customer placing the order already exists. 'Customer' is an object of interest for this functional process. Measure one Read for validating that the entered Customer ID corresponds to a customer that already exists, so that orders may be accepted.

EXAMPLE 3 for Step 3. When entering a post-code (or Zip-code) as part of an address, the entered code must be checked against a list of valid codes available via a service that is outside the scope of the software being measured. Measure one Exit/Entry pair to account for the data movements needed to validate the entered post-code.

EXAMPLE 4 for Step 4. If as in Example 2 for Step 2, the valid customer names must be displayed for the user to select the customer that is placing the order, then measure one Exit for the display of the valid customer names.

Note: Only the one Entry is needed to move the data group 'P' into its functional process, regardless of the fact that functionality to validate any of the attributes of the data group 'P' may require additional data movements.

4.2.4 Data movements in enquiries and reports

Fixed or *ad hoc* enquiry or reporting functional processes may output one or more data group types occurring singly, or repeatedly (as in lists). Reports can be particularly difficult to analyse and measure because, depending on their FUR, they may contain any collection of data including data groups that are not normalized or even not related. (Example: an invoice may include an advertisement that is unrelated to the invoice data.)

The number of Exits on outputs of enquiries or on reports that comprise a fixed and/or limited number of possible data group types can always be determined by applying the rule of section 2.6.2 for distinguishing data groups and hence objects of interest dependent on their differing frequency of occurrence and differing key identifying attributes.

To assist understanding of how to apply this rule, this section gives several examples starting with simple enquiries and going on to processes that output complex reports.

Remember when using the rule of section 2.6.2 that:

- a data group type may contain only one attribute type;
- the output of an enquiry or reporting process may vary with the input parameters. All the possible Exits resulting from the input parameters should be identified and measured. (Obviously this remark does not apply to measure the output of a general purpose enquiry tool that may produce an unlimited range of possible data group types.)

There now follow several examples that illustrate how to measure various types of simple and more complex enquiry or reporting functional processes.

EXAMPLE 1a: Simple enquiry

Assume there is a database with two objects of interest; key attributes are underlined:

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

Order (Order ID, client ID, product ID, order date, ...) [i.e. these are single-item orders]

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:

DM	***Key Attributes***	***Data Group / (Data attributes)***
Entry	*Start date, End date*	*Time-period definition*
Read	*Order ID*	*Order data (Order ID, client ID, etc.)*
Exit	*Start date, End date*	*Time-period definition*
Exit	*Start date, End date, Client ID*	*Client_order data (Client ID, Number of orders)*

In this example, the Entry moves a transient data group defined by the attributes 'Start date' and 'End date' of one object of interest which could be called 'Time-period of the enquiry').

Since the two data groups that are output have different frequencies of occurrence (and different identifying key attributes), one for the time-period definition and many for the client order data), the rule of section 2.6.2 tells us that 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:

DM	*Key Attributes*	*Data Group / (Data Attributes)*
Entry	*Start date, End date*	*Time-period definition*
Read	*Order ID*	*Order data (Order ID, client ID, etc.)*
Read	*Client ID*	*Client data (Client ID, client name, address, etc.)*
Exit	*Start date, end date*	*Time-period definition*
Exit	*Start date, End date, Client ID*	*Client order_address data (Client ID, client name, 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 to the 'Client_order data' data group of Example 1a does not alter the fact that there is still only one movement of data describing a Client in Example 1b, hence one Exit.

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. First, it could be laid out as in Example 1b, with three blocks, one for each of the three time periods:

Period 1 start and end dates

Client A, client name, client address, # orders. (where # = 'number of')

Client B, client name, client address, # orders.

Client C, etc.

This block would then be repeated for Periods 2 and 3.

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, # orders.

Period 2 start and end dates, # orders.

Period 3 start and end dates, # orders.

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

A third possible layout could have a tabular report with four main column headings a) to d):

a). Client (ID, name, address); b). Period 1 start/end dates; c). Period 2 start/end dates; d). Period 3 start/end dates.

The headings would be followed by a row for each client that placed at least one order in at least one time period, with the following attributes.

Client ID, client name, client address; # orders placed in period 1; # orders placed in period 2; # orders placed in period 3.

For all three possible layouts, the data group 'Time-period definition' now has up to three occurrences and the 'Client' data group has one occurrence for every client that placed an order in at least one of the time-periods. But applying the rule of section 2.6.2 tells us that the functional process outputs two data groups (distinguished by differing frequencies of occurrence and different

key attributes), regardless of the report layouts. Hence the functional process of Example 1c is still the same size of 5 CFP as that of Example 1b, ignoring possible error/confirmation messages.

It is important to note that all three report layouts convey the same information, even though the physical groupings of data differ. This example illustrates Note 2 to the rule in section 2.6.2: the physical distribution of data attributes on a report cannot be relied upon to determine the data groups to which they belong. Use the rule of section 2.6.2 to decide on the data groups.

EXAMPLE 2a: 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:

DM	***Key Attributes***	***Data Group***
Entry	*Search parameter ID*	*The employee age limit (of the ad hoc enquiry)*
Read	*Employee ID*	*Employee data*
Exit	*Employee name*	*Employee name (for all employees older than the given age limit)*

Note that the data groups moved by the Entry and the Exit describe different objects of interest. The 'Search parameter ID' of the Entry defines an object of interest which occurs once and could be named 'the set of all employees that satisfy the search parameter'. The Exit occurs many times and moves data about the object of interest 'employee'. 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 2b: 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 'age limit' is an attribute of the object of interest: 'the set of all employees that satisfy the search parameter' of the enquiry. The two Exits have different frequencies of occurrence. The analysis would be as below.

DM	***Key Attributes***	***Data Group***
Entry	*Search parameter ID*	*The employee age limit (of the ad hoc enquiry)*
Read	*Employee ID*	*Employee data*
Exit	*Employee name*	*Employee name (for all employees older than the given age limit)*
Exit	*Search parameter ID*	*The employee age limit*

The total size of FUR1+ FUR2 would be 4 CFP.

EXAMPLE 2c: If there were now a further requirement (FUR3) to output the total number of employees that satisfy the age-limit criterion in addition to the age limit (of FUR2), this would be a second attribute of the same object of interest ('the set of all employees that satisfy the search parameter') that already has 'age limit' as an attribute. The size of the functional process satisfying FUR1 + FUR2 + FUR3 would be unchanged at 4 CFP.

(This is another example of the application of Note 2 of the rule in section 2.6.2. For example 2c), the 'age-limit' and the 'total number of employees that satisfy the age-limit' might appear on separate physical rows of the report. But they both occur once and both describe the same object of interest 'the set of all employees that satisfy the search parameter'.)

EXAMPLE 3: Multi-stage enquiry

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, the software must show a list of the clients with this name, plus sufficient client data (e.g. their address) to distinguish them – referred to as 'client summary data'. The user may then select the right client and the software will show its details.'

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 their address, i.e. the client summary data. (In the following, 'Part' refers to the three parts of the rule in section 2.6.2)

The two alternative data groups have the same identifying key attribute, namely 'Client ID' (Part 2). However, these two data groups have different frequencies of occurrence (Part 1) and both are required to be output according to the FUR (Part 3). Although on any one occurrence of this functional process there would be only one occurrence of an Exit with the key attribute 'Client ID', two different data groups may be output, so two Exits must be counted, plus an Exit for a possible error message.

The second functional process is needed if there is more than one client with the entered name so that the user may select the correct name from the list displayed by the first functional process. The detailed analysis is as follows:

Functional process 1, showing client details, or the list of clients with the entered name.

DM	***Key Attributes***	***Data Group***	***# Oc's***
Entry	*Client name*	*Client name*	*1*
Read	*Client ID*	*Client details*	*Many*
Exit	*Client ID*	*Client details (one is found)*	*1*
Exit	*Client ID*	*Client summary data (more than one Client is found with the same name)*	*2 or more*
Exit	*Errors*	*Error/confirmation messages (in case no client is found)*	*1*

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

DM	***Key Attributes***	***Data Group***	***# Oc's***
Entry	*Client ID*	*Client name (by selecting from the Client summary data list)*	*1*
Read	*Client ID*	*Client details*	*1*
Exit	*Client ID*	*Client details*	*1*

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 and for each division, and the total number of employees for the whole company. The report can be initiated at any time from a menu of functions available to Personnel staff. Today's date must also be output; we assume date and time can be obtained from the operating system, i.e. it does not have to be input or Read (See section 4.4.10 for how to measure this.)

The solution for the functional process that satisfies these FUR is shown below:

DM	***Key Attributes***	***Data Group (Data attributes)***
Entry	*Report request ID*	*Report-type selection (This is implied when selecting the menu item)*
Read	*Employee ID*	*Employee data*

Read	*Department ID*	*Department data (includes Division ID for the Department)*
Exit	*Employee ID*	*Employee name (grouped by department within division)*
Exit	*Today's date, Department ID*	*Department details (Department ID, Department employee subtotal at today's date)*
Exit	*Today's date, Division ID*	*Division details (Division name, Division employee subtotal at today's date)*
Exit	*Today's date*	*Company details (Today's date, Company employee total)*

In this example, one functional process is needed to produce the report, which must be triggered by an Entry and which must have 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.)

The four data groups that are output all have different frequencies of occurrence (and different key attributes), so must be distinguished according to the rule of section 2.6.2. There are four Exits.

Note that 'Today's date' must be a partial key identifying attribute of the two employee sub-totals and the unique key identifying attribute of the Company total, since all these sub-total or total values (and the list of employees) are only valid at 'Today's date'. The same three Exits would need to be measured even if the FUR did not actually require 'Today's date' to be 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: Report with two-dimensional multi-level aggregations

A company designs and produces items of clothing. Each of its designs, known as a 'Style', is available in many Colour and Size combinations. A unique combination of these three parameters is known as a 'Product-type'. This is a specification that the company manufactures and that can be ordered in bulk by a customer (e.g. a retailer)

Figure 4.2 shows the relationships between the various objects of interest that make up the company's product structure and how multi-item Orders relate to a Product-type. (The 'crows-foot' symbol in Figure 4.2 illustrates the degree of the relationship between the various objects of interest.)

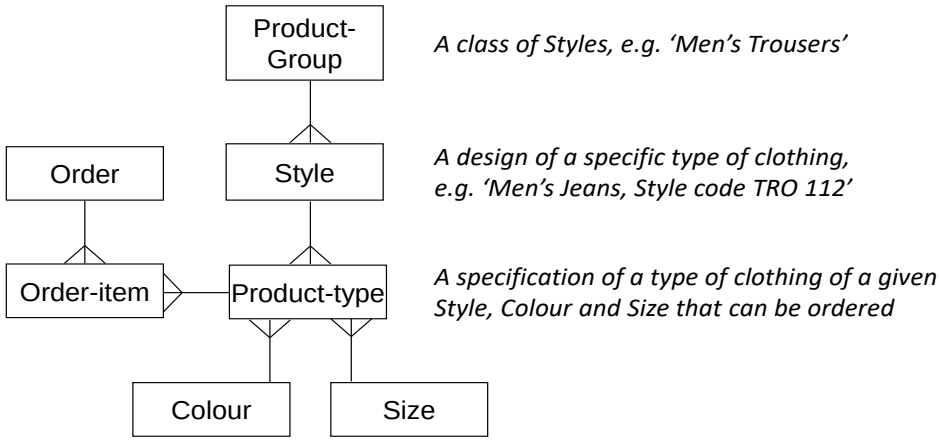


Figure 4.2 – The relationships between the objects of interest about which persistent data is held that define the company's product structure, and orders for its product-types

The company's coding and naming systems are as follows.

- *A Product-Group is identified by a 3-alpha Product-Group code, e.g. TRO (for trousers), SHI (for shirts), etc. Product-group has no other attributes.*
- *Each Style is uniquely identified by its 3-alpha Product-Group code and by a 3-digit Style Number which is unique within that Product-Group. Together, these form a 6-character 'Style code' key. Examples of Style codes are:*
    - *SHI049 (where '049' indicates this is the 49th shirt design)*
    - *TRO112 (where '112' indicates this is the 112th trouser design).*
- *A Style has other attributes such as Style name, Style description, Fabric code, Designer, Date of introduction, Manufacturing cost, etc.*

The company refers to its collection of all Product-types at any one time as its 'Product-Range'. No data is held persistently about the Product-Range.

An Order may have one or more Order-items and at any time there may be many Order-items for a given Product-Type. But each Order-item must specify only one Product-Type.

- *Each Order has as attributes – Order ID, Customer ID, Date of the sale, ID of salesman responsible, etc.*
- *An order can be placed for multiple Product-types, each in a separate 'Order-item'. Each Order-item has as attributes: Order-item number, Style Code, Colour name, Size code, quantity ordered, sale price.*

A report is required on sales of the Company's Product-types at the levels of aggregation of Style, Product-Group, and for the whole Product Range, by month, and for a time period of any number of months. The 'Time-period' must be defined by the user, by entering the 'Start Month/Year' and the 'End Month/Year'.

The data to produce this report must be derived from a system that holds data on all Orders that have been sold and paid for, over a long a period of time.

Figure 4.3 shows an example '2-dimensional' sales report, which is abbreviated for convenience, namely:

- *Sales of the whole Product Range appear in the bottom row, by month and for the whole Time-period;*
- *the Product Range comprises several Product Groups, but Figure 4.3 shows rows for only two occurrences: Shirts and Trousers;*
- *Each Product Group has several Styles, but Figure 4.3 shows rows for only three occurrences of Styles for each Product Group;*
- *The total Time-period of the report spans nine months, but Figure 4.3 shows columns for only four occurrences of months, as well as the final column for total sales in the whole Time-period. Note that the year associated with any month is found in the report heading. (For example the column headed 'July' is really 'July 2016', a month/year combination.)*

Sales Report ($) **Period:** July 2016 – Mar 2017

Prod. Grp	Style	July	Aug	Sept	…(etc)…	Mar	Total
	SHI123	1130	1450	2500		2120	12,340
	SHI456	300	650	920		480	5,990
	SHI789	1250	1500	2150		2080	14,480
	(etc)	(etc)	(etc)	(etc)		(etc)	(etc)
Shirts		**6990**	**7120**	**7250**		**7370**	**189,350**
	TRO112	1450	1410	1190		3320	10,980
	TRO113	350	570	1390		2640	8,410
	TRO114	730	890	1050		1540	12,340
	(etc)	(etc)	(etc)	(etc)		(etc)	(etc)
Trousers		**5432**	**5650**	**5990**		**10,350**	**171,110**
(etc.)							
Product Range		**112,422**	**127,700**	**142,490**		**157,220**	**949,320**

Figure 4.3 – Clothing Company's sales report for the period July 2016 to March 2017

We start to analyse the report by examining the sales amount data attribute types at six levels of aggregation (three levels of aggregation on the product dimension by two levels of aggregation on the time dimension). These six levels are colour-coded in Figure 4.3 to help distinguish them. Each sales amount is a single-attribute data group (-type). The six groups are as follows, with their key attributes:

i. *Sales for a given Style in a given month, in black [2 Keys: Style code, Month/Year name].*

ii. *Sales for a given Product-Group in a given month in brown [2 Keys: Product-Group code, Month/Year name].*

iii. *Sales for the whole Product Range in a given month, in red [1 Key: Month/Year name].*

iv. *Sales for a given Style in the whole Time-period, in green [2 Keys: Style code, Time-period definition], where 'Time-period definition' is a group key of two attributes: Start Month/Year, End Month/Year.*

v. *Sales for a given Product-Group in the whole Time-period, in blue [2 Keys: Product-Group code, Time-period definition].*

vi. *Sales for the whole Product Range in the whole Time-period, in purple [1 Key: Time-period definition].*

These six sales amounts all have different frequencies of occurrence (and different key identifiers, or combinations of key identifiers) so must, according to the rule in section 2.6.2 be different data groups giving rise to six different Exits.

Note that 'Product Range' is only a field heading or label. The data group for sales at the level of Product Range (purple) for the whole time-period occurs only once and depends only on the 'Time-period definition' key attributes. So the latter attributes and this sales amount are all attributes of the same one data group.

Note further that the 'Style code' (one of the single-attribute groups that is output) and the 'Sales for a given Style in the whole Time-period' have the same frequency of occurrence (one occurrence per Style). However, their key attributes differ. (The key to a Style is 'Style code'. The key attributes of 'Sales for a given Style in the whole Time-period' are 'Style code' and 'Time-period definition'.) Therefore applying Part 2 of the rule in section 2.6.2 these two data groups describe different objects of interest, so two Exits must be counted. The same is true for the 'Product-Group (code)' and the 'Sales for a given Product-Group in the whole Time-period'. They have the same frequency of occurrence but different key attributes. Two Exits must be counted for these data groups.

The following is an analysis of the whole functional process needed to produce the report. In the table, 'n' is the number of Months defined for the entered time-period. Note that, leaving aside the error/confirmation message, all the Exits have different frequencies of occurrence or the same frequency of occurrence but different key identifiers).

DM	Key Attributes	Data Group	# Oc's
E	Time-period definition*	Time-period	1
R	Order ID, Order-Item ID	Order-item details	'Very many'
R	Product-Group code	Product-Group code	'Several' (One for each Product-Group)
X	Product-group code	Product-Group code	'Several'
X	Style code	Style code	'Many'
X	Style code, Month/Year	Sales for a given Style in a given month	Many x n
X	Product-Group code, Month/Year	Sales for a given Product-Group in a given month	Several x n
X	Month/Year	Sales for the whole Product Range in a given month	n
X	Style code, Time-period definition*	Sales for a given Style in the whole Time-period	'Many' (One for each Style)
X	Product-Group code, Time-period definition*	Sales for a given Product-Group in the whole Time-period	'Several' (One for each Product-Group)
X	Time-period definition*	Sales for the whole Product Range in the whole Time-period	1
X	Errors	Error/confirmation message (e.g. if error in input Month/Year values)	1

** 'Time-period definition is the group of the two attributes 'Start Month/Year' and 'End Month/Year'.*

*The total size of the functional process to produce this report is **12 CFP**.*

It is now interesting to demonstrate the applicability of Note 1 to the rule of section 2.6.2. Figure 4.4 shows the reports that would be needed if, for example, the sales amounts i) and v) in the list above were required to be output by separate functional processes.

i) Shirt Style Monthly Sales (2015/16)

Style	Jul	Aug	Sept	(etc.)	Mar
SHI123	1130	1450	2500		2120
SHI 456	300	650	920		480
SHI 789	1250	1500	2150		2080
(**Etc**.)	(etc)	(etc)	(etc)		(etc)

v) Product-Group Sales

Period: July 2016 - March 2017

Product-Group	Sales ($)
Shirts	189,350
Trousers	171,110
(Etc.)	

Figure 4.4 – Separate reports for two of the six Sales objects of interest

A functional process to output only the sales amount i) – sales for a given Style in a given Month - would have three Exits, all with different frequencies of occurrence (and different identifying key attributes). The analysis of the Exits, would be:

DM	Key Attributes	Data Group	# Oc's

X	*Style code*	*Style code*	*'Many'*
X	*Month/Year*	*Month/Year*	*n*
X	*Style code, Month/Year*	*Sales for a given Style in a given month*	*Many x n*
X	*Errors*	*Error/confirmation message (e.g. if error in input Month/Year values)*	*1*

A functional process to output the sales amount v) – sales for a given Product-Group in the whole Time-period, - would have two Exits, both with different identifying key attributes and different frequencies of occurrence. The analysis of the Exits would be:

DM	***Key Attributes***	***Data Group***	***# Oc's***
X	*Time-period definition*	*Time-period definition*	*1*
X	*Product-Group code, Time-period definition*	*Sales for a given Product-Group in the whole Time-period*	*'Several' (One for each Product-Group)*
X	*Errors*	*Error/confirmation message (e.g. if error in input Month/Year values)*	*1*

It will be seen that the unique (or unique combinations of) identifying key attribute types for the Exits from these two functional processes are all found in the analysis of the Exits for the whole report.

If we were to apply this same analysis to find the unique (or unique combinations of) identifying key attribute types for all six sales amounts, i to vi, if they were output by six separate functional processes, the count of unique (or unique combinations of) identifying key attribute types for all the Exits output by the six processes would be the same, **nine** in total, including the error message as when the same data are output on the one report of Figure 4.3. This analysis confirms the guidance of Note 1 for the Rule in section 2.6.2.

EXAMPLE 6: Measurement of output data in graphical form

Measurement of output data does not change if the data are presented graphically rather than in table form. Figure 4.5 shows the expenses of a company's departments in 2014 and 2015.

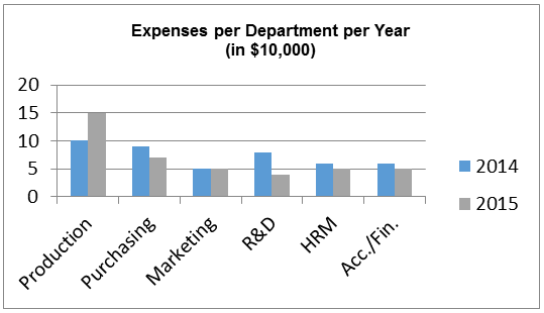


Figure 4.5 – Data in graphical form

The height of the bars represent the amounts, of which the one object of interest can be called 'Departmental Annual Expenses'. The amounts attribute is part of a group that has two key identifying attributes: 'Year of Expenses' (which appears in the legend) and 'Department name'. Both have a (different) one-to-many relationship with the expenses amounts. Hence the three attributes have different frequencies of occurrence, so three Exits must be identified to produce this graph, as shown below.

DM	***Key Attributes***	***Data Group***	***# Oc's***
Exit	*Year number*	*Year of expenses*	*2*
Exit	*Department name*	*Department name*	*6*
Exit	*Department name. Year number*	*Department annual expenses*	*12*

The graph title explains the meaning of the amounts; it is fixed text that should not be measured (see section 4.4.8).

If the graph were required to show only the expenses for the one year 2015 where the year must be entered as a variable, there would be two Exits with key attributes 'Year number' and 'Department name'. The 'Year number' still has a one-to-many relationship with the expenses amounts. However, the 'Department name' now has a one-to-one relationship with the amounts, i.e. the amounts and the Department names are both attributes of the object of interest 'Department 2015 Expenses'. Note that if the legend '2015' were removed and the diagram title changed to 'Expenses per Department in 2015 (per $10,000)', two Exits must still be identified.

See also Example 3 in section 4.4.5 on 'Multiple sources, destinations and formats of a data movement: application of the 'data uniquenss rule''. If, for example, the data of Figure 4.6 may be displayed in graphical form on a screen, and the first functional process above allows the user to also print the Expenses figures in table form, then six Exits must be counted for the one functional process.

4.2.5 Other Examples

This section contains various examples to illustrate how to measure particular types of requirements.

EXAMPLE 1 – 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 (Order ID, client ID, client order reference, required delivery date, etc.)*
- *Order item (Item ID, product ID, order quantity, etc.)*
- *the key attributes Order ID and Item ID are generated automatically by the software*
- *the client ID and the product ID must be validated on entry*
- *the entered data must be validated and confirmed, or an error message be given.*

Solution for this functional process:

DM	***Object of interest***	***Data Group (Attributes)***
Entry	*Order ID)*	*Order data (Client ID, client order reference, required delivery date, etc.)*
Read	*Client ID*	*Client data*
Entry	*Item ID*	*Order-item data (Product ID, order quantity, etc.)*
Read	*Product ID*	*Product data*
Write	*Order ID*	*Order data (Order ID, client ID, client order reference, required delivery date, etc.)*
Write	*Item ID*	*Order-item data (Item ID, product ID, order quantity, etc.)*
Exit	*Errors*	*Error/confirmation messages*

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 the 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 2: Functional processes with object of interest sub-types

Sometimes, separate objects of interest should be recognized as sub-types of a particular object of interest type. A sub-type of an object of interest type inherits all the attributes of the object of interest type but also has its own unique attributes. (This is directly comparable to the concept of classes and sub-classes in UML – see section 2.4.)

We ignore the possible requirement for error-confirmation messages in the following.

a) *FUR 1: Suppose the object of interest 'Customer' has an attribute that indicates its category 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 categories are NOT regarded as sub-types of Customer. 'Customer-category' is simply one attribute of Customer amongst many.*

b) *Alternatively suppose the following FUR 2.*

- *'For the data stored about Customers, all customers shall have the attributes: Customer ID, Customer Name, Address, Telephone no., Customer-category.*
- *'Customer-category' 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-category, 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 4.6 (a).

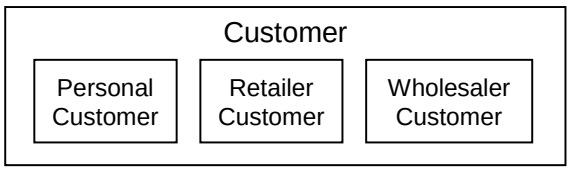


Figure 4.6 (a) – Object of interest type with its sub-types

[Note: The physical database structure would probably be as shown in Fig 4.6 (b) below. The Customer record would hold the data that is common to all three customer-categories. There is no need for a separate database record for Personal customers, as they only have the attributes that are common to all Customers.

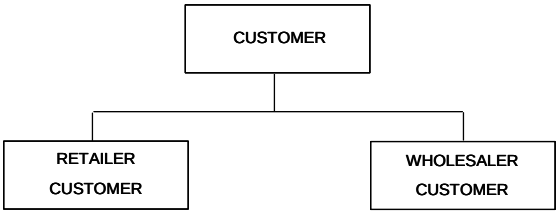


Figure 4.6 (b) – The probable corresponding physical database record structure

This physical data model is not relevant to the COSMIC measurement but it does illustrate how Measurers must sometimes abstract from a physical to a logical data 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. We consider only how the Entries, Reads and Writes vary due to the effect of the three sub-types on the FUR.

FUR 3. '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'.

FUR 4. '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.

FUR 5. '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 refer to (i.e. Read) the three separate objects of interest when validating the entered data, namely:

- *'Customer' in order to validate that the Customer exists and that orders can be accepted;*
- *'Retailer Customer', or 'Wholesaler Customer' in order to apply the correct discount rules for pricing the order.*

So this functional process FP3 would have three Reads, for the Customer and for the two sub-types, i.e. a total of six CFP for these sub-processes.

FUR 6. '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 as well as a Read of Customer to check that the Customer does not already exist, i.e. a total of seven CFP for these sub-processes.

The general principles are:

- where the FUR do not require any 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 3: Maintenance of parameter tables

In order to ease the task of maintaining data, it is common practice to store attributes of coding systems in tables and to provide software to help maintain these data. Examples are:

- codes and names, e.g. 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.

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 2.6.3, 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, but who is not a normal, authorized 'business user'.) For any such users who must maintain the parameter tables, the attributes of the parameters do describe objects 'of interest' to that user.

Several situations may arise.

EXAMPLE 3a: 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, where each parameter-type has just two attributes, namely 'code' and 'description'. 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 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 Entry could be either a menu selection to display all parameters, or the entry of a specific parameter code. In the latter case an error/confirmation message would probably be 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, or 13 CFP if an error/confirmation message is needed for an enquiry on a specific code.

EXAMPLE 3b: 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 3c: 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 4: 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. 'Any of the above' 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 understanding, maintenance and explanation, the expert system shall be rule-based. The expert system shall explain why it has posed a question 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 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, if any, 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 questions and to 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, for a client user who has not previously used the system is 'Show holidays'. We assume there will be one functional process to enable the client to enter his/her 'wishes' in answer to all questions, which has the following data movements. We assume the software allocates a unique ID to the set of client enquiry wishes.

DM	***Key Attributes***	***Data Group***
Entry	*Client enquiry ID*	*Client holiday requirements*
Read	*Holiday knowledge*	*Holiday rules*
Read	*Holiday ID*	*Holiday data*
Exit	*Client enquiry ID*	*Client holiday requirements*
Exit	*Holiday ID*	*Selected holiday details*
Exit	*Error*	*Error/confirmation message* *(In case no holiday satisfies the requirements)*

We assume in the above that the functional process must display both the client's requirements and one or more 'Selected holiday details', so that the client may subsequently repeat the functional process to see the effect of changing the input requirement parameters.

This functional process obviously has rule-processing logic which we assume accesses a set of persistently-stored rules 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 'Selected holiday details' Exit.

If the client wishes to save this set of questions and answers for future use, he/she initiates a separate functional process immediately following the first process, and allocates a name to his set of questions and answers, with the following data movements.

DM	***Key Attributes***	***Data Group***
Entry	*Client_Questions & answers set name*	*Client_Questions & answers details*
Write	*Client enquiry ID*	*Client holiday requirements (including the set name)*
Write	*Holiday ID*	*Selected holiday ID (for each holiday that meets the client's r~~we~~quirements)*
Exit	*Errors/Confirmations (S)*	*Confirmation message (The client needs assurance that the set has been saved.).*

4.3 Sizing components of distributed 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, 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; e.g., 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 or 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 two 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. We assume in both examples that the communications between the components of the distributed system take place synchronously. For the effect that asynchronous communications would have on these examples, see section 4.3.2.

4.3.1 Examples

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 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 1a** Measurement scope is the whole application.*

In this case, the (human) functional users of the application have no knowledge that the application is physically distributed over the PC and the main-frame. The data movements between the two components are 'invisible' to them. Similarly, any additional functionality in lower layers needed to achieve the communications between the PC and the main-frame is invisible and outside the scope when measuring the application.

The data movements of the functional process to obtain the required data are then as shown in the Message Sequence Diagram of Figure 4.7 below (total 3 CFP):

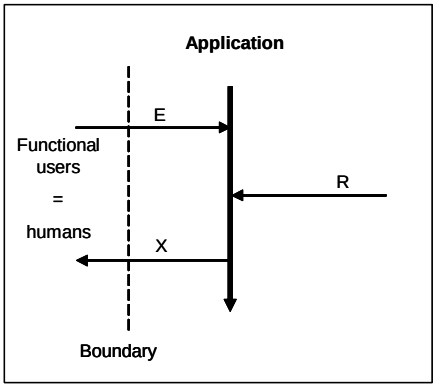


Figure 4.7 –Measurement scope is the whole application

***Case 1b** Measurement scope is component A*

For this case, the FUR of component A must describe its communications with the server B (even though the (human) functional user of the PC component A sees no difference to that of case 1a.

Figure 4.8 shows the data movements that component A needs in order to obtain the required data from component B. 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. The data movements of case 1b are therefore as shown below (4 CFP).

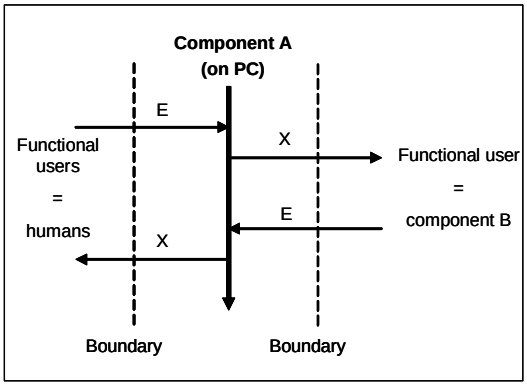


Figure 4.8 –Measurement scope is component A

***Case 1c** Measurement scope 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. The data movements of component B are shown in Figure 4.9.

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.

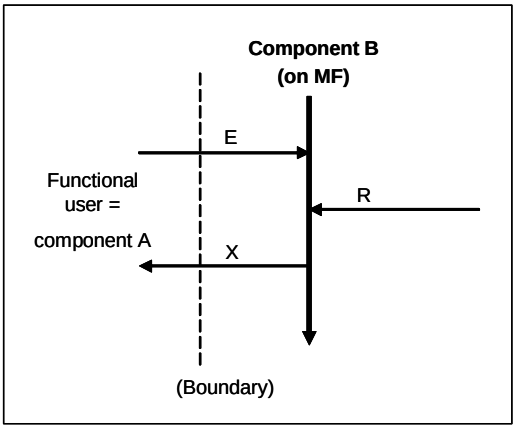


Figure 4.9 - Measurement scope is component B

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

It follows that the increase in size of the functionality of this process when the purpose is to measure the size of the two application components separately is (4 + 3 – 3) = 4 CFP compared with case 1a.

As another cross-check on this conclusion, and following the 'aggregation rules' in the Measurement Manual [1], section 4.3.1, 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 as in Example 1, 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 2a** Measurement scope is the whole application*

This case now differs from Example 1a 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.

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 to the sales staff functional users (both data movements occurring physically on the laptops)*

***Case 2b** Measurement scope is component A*

Figure 4.10 shows the context diagram for Cases 2b and 2c

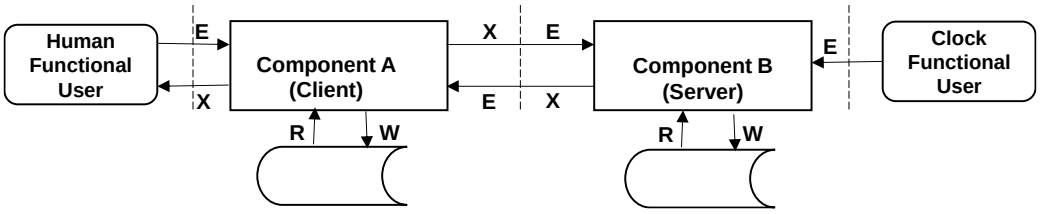


Figure 4.10 – Context diagram for cases (b) and (c) of Example 2

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 analysed in the normal way.

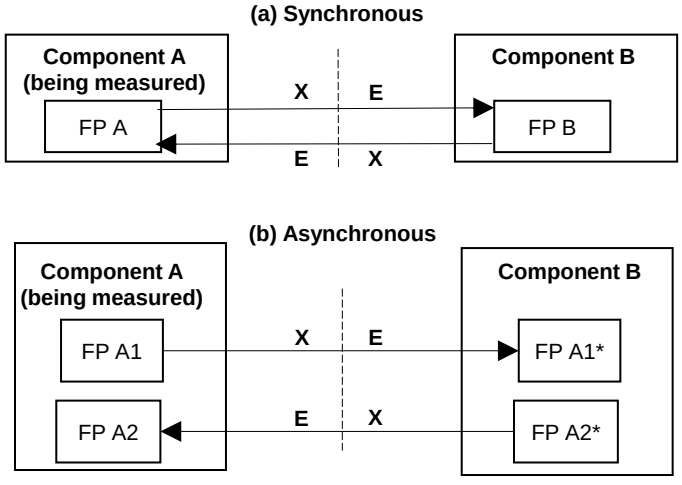
***Case 2c** Measurement scope is component B*

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 analysed 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 2b and 2c is likely to be a significant number of additional CFP, due to the inter-component data movements, compared to case 2a where the purpose was to size the application ignoring its split into separate components.

4.3.2 Synchronous versus asynchronous communications

The COSMIC model of the communications between components of a distributed software system depends on whether the communications will be synchronous or asynchronous. To describe the difference from a functional point of view, assume a system of two components A and B that communicate with each other (See Figure 4.11).



* Note: It does not matter to component A how Component B handles the response to the message from Component A (i.e. 1 or 2 FP's?)

Figure 4.11 – COSMIC functional model of synchronous and asynchronous communications

- In synchronous communications, if a functional process FP A needs to send data to or receive data from component B, then FP A issues an Exit to component B and waits for the response, if expected. A functional process FP B of component B sends the response as an Exit which is received by functional process FP A as an Entry.

  (When FP A is required only to send data to B, the response from B might be that the data has been received OK, or there might be no response.)

- In asynchronous communications, functional process FP A1 issues its Exit to component B and then terminates. Component B issues its response as an Exit, which is received as the triggering Entry of another functional process FP A2 of component A.

Note that the number of data movements between the two components (i.e. the Exits and Entries) is unaffected by the nature of the communications (and we are not interested in this model in how the communications are achieved technically).

EXAMPLE: Asynchronous communications. Suppose in Figure 4.11(b) the component A is in a hotel reservation system and component B is in the system of a credit card supplier. When a hotel customer wants to reserve a hotel room, the credit card system may not be available or may be heavily loaded. In order not to keep the hotel customer waiting, the communication between the two systems takes place asynchronously.

4.4 Other measurement conventions

4.4.1 Control commands

'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, e.g. via 'page up' and 'page down', or by scrolling,
- 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 4.4.3.

4.4.2 Authorization, help and log functionality

Authorization, help and log functionality may be measured if their functionality is:

- explicitly described in the FUR of the application being measured;
- in the application layer, and
- agreed to be within the overall 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.

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

EXAMPLE 1: Assume a Help button is provided on each screen, independent of the function of the screen. Measure one Help functional process for the whole application if the functional process provides the same service from wherever it is pressed, i.e. it is of the same functional process type for all screens of the application. (The output from the Help function may well be context-dependent but such output simply represents different occurrences of the same output-type.)

A Help function, once invoked, may offer other functional processes for more searching enquiries. In this case each of these functional processes must be analysed according to the usual rules, assuming the Help sub-system is within the measurement scope of the business application.

EXAMPLE 2: 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, with a date/time stamp. Assuming the movement of all logged records is identical, identify one Write data movement for all log data groups that are written in each functional process where logging is required. (For the logging functionality, the object of interest of the log file data group that is written is, for example 'changed record in database X' for all records that must be logged.)

4.4.3 GUI elements

For GUI elements conveying control data, e.g. navigation buttons, or 'close screen', the rules for control commands apply (see section 4.4.1), i.e. these elements 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 and functional processes'.

4.4.4 Menus and the triggering Entry

Ignore 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). Such a menu command should be regarded as just a control command.
- 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.

A menu command should be measured as the triggering Entry of a functional process if the menu command:

- launches a specific functional process that does not require any input data. Example: a process to launch a standard report from a menu. (When a user presses a button on a menu to launch such a specific enquiry functional process P, a message goes to the software saying 'start this specific enquiry P'.)
- needs selection parameters that the user must input before pressing 'Enter'. These are additional attributes of the same data group (but now the message to the software is: 'start this specific enquiry P with these selection parameters').

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 clicking on a menu button (so this may appear 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.

4.4.5 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 FUR for processing in batch mode do sometimes necessarily differ from their equivalents for on-line processing. Examples:

- GUI features are unique to on-line processing.
- 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.
- In batch mode, a functional process to update an existing record usually needs to read the existing record, update it and write back the updated record, all in the same one process. In contrast, in on-line processing, usually the data to be updated must first be retrieved and displayed by an enquiry functional process. This is then followed by a separate update functional process. Similarly, in batch mode, normally a delete functional process needs only a Write data movement to delete the record. In contrast, in on-line processing mode a delete functional process is normally preceded by a functional process to retrieve-and-display the data to be deleted.

- Batch-processed applications may need sub-processes or program steps in the middle of processing, e.g. a data sort, or a checkpoint / re-start 'save' (which is a 'control' feature, implementing roll-back technical requirements). The Measurer should ignore such sub-processes if they do not read or write persistent data and/or are provided by standard utilities outside the scope of the measurement.

All data that has been entered as *input* for batch processing must be either:

- transmitted in across the boundary as a batched file of data to be processed, or
- temporarily stored somewhere before the process can start. See Figure 4.12. (Note 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'.

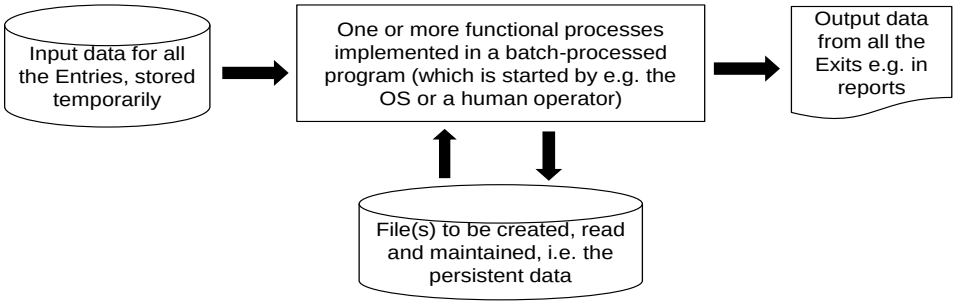


Figure 4.12 – A batch-processed 'job', implementing a collection of functional processes

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 analysed 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 main tasks of the Measurer in analyzing batch application streams are to identify the separate triggering Entries and hence the separate functional processes, and 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 analysed 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 analysed 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 analysed '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 analysed 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 of application A 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.to application B.*

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: 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 the file to the importing application.*
- *The functional process that is specific to the importing application and to the file 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:

DM	*Key Attributes*	*Data Groups*
Entry	*File ID*	*Interface file data*
Read	*File ID, Processing date*	*Interface file history (file already processed?)*
Write	*File ID, Processing date*	*Interface file history (store result of processing)*
Exit	*Errors*	*Error/Confirmation Messages*

For processing the employee data, assuming a 'create' functional process:

DM	*Key Attributes*	*Data Groups*
Entry	*Employee ID*	*Employee data*
Write	*Employee ID*	*Employee data*
Exit	*Errors ID*	*Error/Confirmation Messages*

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

4.4.6 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 different data groups to be entered into one functional process from functional users that must be separately identified by that functional process, where each data group describes the same object of interest. One Entry shall be identified for each of these different data groups.

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

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

Note that it is both the multiplicity of source or destination functional users and the multiplicity of different data groups required to be entered or sent according to the FUR that determines if we have different data movements. (In this sentence, data groups having the same identifying key attributes and thus describing the same object of interest should be considered as 'different' if the FUR specify different non-key attributes and/or different formats of the same attributes.)

EXAMPLE 1: 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.

EXAMPLE 2: A data group is displayed as output on a screen, and printed in the same format. One Exit is identified.

EXAMPLE 3: As per EXAMPLE 2. The printed 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.

---

[20] See the 'Data Uniqueness' rule in section 3.5.7 of the Measurement Manual for v4.0.1 of the COSMIC Method.

[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.

EXAMPLE 4: A file of 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, and 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. But an Exit that is identical when included in different files for different functional users is counted only once for the application.

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 data in addition to 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 an Exit for each object of interest about which data is found in the letter.

- 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 in that same functional process, i.e. 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 and other fixed information

'Fixed text' is text that:

- is available to the software being measured, either hard-coded in the software, or available from persistent storage or from other software;
- may be selected but not changed by the normal functional user. (It may be maintainable by a system administrator.)

'Other fixed information' could include diagrams, photos, audio or video output.

Fixed text may be output on request, or is made available 'automatically' by the software to help a human functional user to understand what data must be entered, or to understand output data. The two categories must be analysed differently.

Fixed text or other fixed information that is output on request.

Each function that may be triggered independently to output fixed text should be treated as a separate functional process.

EXAMPLE 1: An enquiry which outputs fixed text, as the result of pressing a button for 'Terms & Conditions' on a shopping web-site, should be modelled as a separate functional process having one Entry for pressing the button, one Read to obtain the text and one Exit for the output.

EXAMPLE 2: A functional process to assemble and print a medical prescription for drugs will have one Exit for the fixed text output of the prescription (plus one Exit to account for the prescription items).

EXAMPLE 3: 'Help' text is fixed text. Another form of Help information could be an explanatory video. See section 4.4.2 for how to analyse and measure Help functionality.

EXAMPLE 4: Menus are fixed text. See section 4.4.4 for how to analyse and measure Menus.

EXAMPLE 5: Error and confirmation messages are fixed text. See section 4.4.7 for how to analyse and measure error and confirmaiton messages.

This category of fixed text may or may not be maintainable via functional processes (Where the fixed text is maintainable, it should be considered as 'parameters' of the software being measured – see section 2.6.3).

Fixed text or other fixed information that is made available 'automatically' to help a human functional user.

Examples are:

- 'Application-general' data, i.e. any data related to the application in general, including headers and footers (company name, application name, system date, etc.), that appears on all screens and reports, that is not related to objects of interest on those screens or reports.
- Field headings to guide data entry, or to help interpret output.
- The fixed text headings of the output of enquiries and of reports.

All these types of fixed text should be ignored unless they must be changed – see section 4.5. (It is the variable data on the input and output that must be analysed and measured.)

Note: enquiry output and report headings may contain data describing an identifiable object of interest. For an example, see Figure 4.4 i), where the heading contains the year of the 'time-period definition' data group. Such variable data should obviously not be ignored.

For examples of the measurement of functional processes that access maintainable and non-maintainable fixed text, see the Web Advice Module case study [15].

4.4.9 Time-out functionality

On-line business applications used by humans may provide automatic 'timeout' functionality for security reasons in case the human user leaves the application inactive for a system-defined maximum period. The user may receive a 'timeout' message and will have to log in again to resume activity.

For an account of how such timer functionality may work and may be measured, see the 'Guideline for sizing real-time software' [20].

4.4.10 Date/time functionality

Normally business applications needing the current date or time obtain these parameters from the operating system. No data movement should be measured for this functionality.

However, if FUR require the attributes date and/or time to be output, they should be analysed and measured in the normal way.

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 'number 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 'number of dependents' is removed from the Employee object of interest.

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

Employee (Employee-ID, …, employee name. address, date of birth, ...no. of dependents, …)

The 'create employee' functional process is

DM	Key Attributes	Data Group
Entry	*Employee ID*	*Employee data*
Read	*Employee ID*	*Employee data (To check if the Employee already exists)*
Write	*Employee ID*	*Employee data*
Exit	*Errors*	*Error/confirmation messages*

Total size 4 CFP

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

Employee (Employee-ID, …) *('no. of dependents' removed)*

Dependent (Employee ID, Dependent-name, date of birth, etc …)

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

DM	Key Attributes	Data Movement changes
Entry	*Employee ID*	*Data movement modified (attribute removed)*
Read	*Employee ID*	*(Not changed)*
Entry	*Employee ID, Dependent name*	*Data movement added*
Write	*Employee ID*	*Data movement modified (attribute removed)*
Write	*Dependent name*	*Data movement added*
Exit	*Errors*	*Data movement modified)*[22]

The size of the functional change to the 'create employee' functional process is 2 Entries + 2 Writes + 1 Exit = 5 CFP. Probably many more functional processes that must deal with the modified or added data groups must also be functionally changed. 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.

4.5.2 Changes to fixed text

If a change is required to a data group containing 'Fixed text or other fixed information that is output on request' (see section 4.4.8), then the Entries, Exits, Read and Writes that move the data group should be measured as changed according to the normal rules.

The fixed text of field headings is measured differently from the headings of input or output screens or reports and application-general data (see section 4.4.8).

[22] There may be new or modified error/confirmation message occurrences. If so, the data manipulation associated with the data movement type will be changed.

Normally, fixed text field headings on input or output only need to change if the data associated with the headings must be changed. As any required change to the data accounts for any changes to the associated headings, the latter should be ignored. It is the change to the data that must be measured.

However, it may arise that there is a requirement to change the 'presentation'[23] of an attribute, including its field heading appearing on input or output, when there has been no change to the associated data. This might be needed, for example, to improve ease of understanding. In these circumstances, one Entry or one Exit may be counted for any input or output data group respectively in which one or more field headings must be changed.

Any required change to the fixed text heading of an input or output screen, or to a report heading, or to application-general data should be ignored, i.e. it should not be measured.

4.5.3 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.

(Implementing the change of Example 1 in section 4.5.1 would probably require a conversion program to be written and used once to allow dependents to be added for existing employees.)

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 and maybe the size of the change, then data conversion software would be excluded from the scope.

4.5.4 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 (6 – 4 = 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).

[23] See the Measurement Manual, section 4.4.1 which states that 'Presentation' can mean, for example the font, background colour, field length, field heading, number of decimal places, etc.

References

All COSMIC publications are available for free download from the portal of www.cosmic-sizing.org, except where another web address is given

(Version numbers and publication dates of COSMIC publications given below are correct at the time of publication of this Guideline. However, these publications are updated from time to time as necessary. The 'cosmic-sizing' website will always have the latest version available.)

[1] Measurement Manual. (The COSMIC implementation guide to ISO/IEC 19761:2011), v4.0.1, April 2015

[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, v1.1, January 2016

[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', www.iso,org.

[6] COSMIC 'Guideline for Early or Rapid Functional Size Measurement using approximation approaches', v1.0, July 2015

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

[13] COSMIC 'Guideline on Non-Functional & Project Requirements', v1.0, November 2015

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

[15] 'Web Advice Module: COSMIC case study', v3.0.1, January 2014.

[16] COSMIC 'Guideline for sizing Data Warehouse Application Software'. v1.1, April 2015.

[17] COSMIC 'Guideline for sizing Service-Oriented Architecture Software', v1.1, March 2015.

[18] Mobile (various articles and case studies on the application of COSMIC FSM)

[19] 'Guideline for the use of COSMIC FSM to manage Agile projects', v1.0, September 2011

[20] COSMIC 'Guideline for sizing real-time software'; v1.1.1, November 2016

[21] COSMIC 'Proposals: to refine definitions and add rules to help improve the identification of objects of interest and data groups', Method Update Bulletin 13, June 2016

[22] COSMIC 'Proposal: to improve the definition of a Triggering Event ', Method Updare Bulletin 14, June 2016

Appendix A

Appendix A – Main changes in v1.3 from v1.2 of this guideline

Note. The nature of a change is indicated by

- 'Method' when a definition or rule of the COSMIC method has been changed
- 'Editorial' when the description of the guidance was changed to improve ease of understanding. Many minor editorial improvements have also been made in addition to those listed below.
- 'Correction' when an error in the previous version v1.2 of this Guideline has been corrected.

References in version 1.2	Nature of change	Comment
-	Editorial	The desription of the analysis and measurement of all functional processes has been transformed to one standard table format.
1.1	Editorial	The characterization of business application software has been expanded to cover application packages and to refer to the various COSMIC publications on specialized types of business application software.
1.2.2	Editorial	In the fourth and fifth paragraphs, the text has been improved to make clear that an approximate size of 'functional requirements' can be measured at any stage as a project evolves but a precise size can be measured only when all the FUR details are known.
1.2.3	Method	The definition of Non-Functional Requirements has been updated in line with the new Guideline [13], to exclude project requirements.
1.2.3	Editorial	Example 1 has been improved.
2.0 (Intro.)	Method	Definition of an 'object of interest' has been updated in line with Method Update Bulletin 13.
2.2	Editorial	Some new text has been added to explain the expression 'frequency of occurrence' as used in the new rule of 2.6.2 and its relation to other related expressions
2.4	Editorial	A remark has been added that due to the close mapping of UML and COSMIC concepts, automation of COSMIC sizing from FUR expressed in UML is now possible. A footnote has been added making clear that it is the part of a 'concrete' object class, not an 'abstract' object class, which corresponds to a COSMIC object of interest.
2.5	Editorial	A remark has been added that the RDA normalization process is now not commonly needed.
2.6.1	Editorial	The Example has been explained in more detail and a Figure 2.4 added to improve ease of understanding.
2.6.1 – 2.6.3	Editorial	In the examples in these sections, the term 'customer-type' was used to indicate three 'types' of customers (personal, retail and wholesale). However, it became evident that the difference in meaning between 'customer' and 'customer type' was not clear. 'Customer' could be interpreted an occurrence of the object of interest type 'customer-type'. The term 'customer-type' was therefore changed to 'customer-category' throughout these sections. 'Customer' and 'customer-category' both now represent different object of interest types.

(New section)	Method & Correction	Section 2.6.2. A new rule for distinguishing objects of interest and data groups due to differing frequencies of occurrence or differing identifying key attributeshas been added. (This rule will be added to the next release of the Measurement Manual.) The analysis of the example of section 2.6.1 is completed in this new section and it has been corrected from how it previously appeared, in section 4.2.3 as Example 8 by deleting one Exit which should not have been counted according to the new rule in this section.
2.6.2	Editorial	In the v1.2 version of Examples a), b) and c) – now re-numbered 1, 2 and 3 in section 2.6.3, an assumption was made in the FUR that it was not necessary to have a Read of Customer to check it does not already exist. This was an unlikely assumption. Changing the assumption results in the size of all three examples increasing by 1 CFP.
2.6.2	Correction	Example c) (now Example 3 in section 2.6.3) has been corrected by adding an Exit for valid customer-category codes. This defect was reported in a Notice issued in September 2015.
2.6.3	Editorial	A final paragraph has been added to draw attention to the new section 4.2.2 on the mapping between the logical analysis of the data movements and the physical movements of data attributes of a functional process
2.6.5	Editorial	An example has been added of objects of interest for which only transient data exists in input and output.
3	Correction	In the fourth bullet point of the introduction to Chapter 3 ‘Measurement Strategy’, the phrase ‘or the physical software’ has been removed, Level of granularity applies only to the requirements of software.
3.1.1	Editorial	A new example 6 has been added concerning size measurement in the case where the FUR will be largely implemented by an application package.
3.2	Correction	In the last paragraph of this section, the phrase ‘staff who maintain software programs’ might be taken to mean computer programmers, which was never intended. ‘Software programs’ has been removed.
3.4	Editorial	A new section has been added describing standard Measurement Patterns for business application software.
4.1.1	Correction	‘State transition’ has been changed to ‘state inspection’ in the context of enquiry or reporting functional processes.
4.1.2	Editorial	The second bullet point has been changed to bring it in line with the current definition of a functional proecss, as introduced in Method Update Bulletin 11 and v4.0.
4.1.5	Editorial	The example of enquire-before-update and update functional processes has been partially re-written to make it much cleaerer. There is no change ot the analysis or to the measured size
4.1.7	Editorial	The paragraph title has been changed and the text substantially re-written to improve ease of understanding and to bring it in line with the examples in section 2.6.3 (formerly 2.6.1 in v1.2).
4.2.1 & 4.2.3	Editorial	In v1,2. There was considerable overlap between sections 4.2.1 and 4.2.3. Section 4.2.1 has been re-written to improve ease of understanding. The old section 4.2.3 of v1.2 has been eliminated.
(New section)	Editorial	New section 4.2.2 has been added to illustrate that the sequence of physically entering data into an on-line application does not necessarily map directly to a COSMIC model of the data movements.
(New section)	Editorial	New section 4.2.3 has been added ‘Data movements needed to validate input data’.
(New section)	Editorial	New Section 4.2.4 has been added ‘Data movements in enquiries and reports’.

		Examples 1, 3, 4 from the old section 4.2.3 of v1.2 are included here, as well as Example 4 from section 4.4.6 (see further below). Example 4 has been modified to require the report date to be output, which makes it more realistic. A new example 5 has been added for a functional process to output a complex report with multi-level, 2-dimensional aggregations of data aiming to show how to apply the new rule in section 2.6.2 to measure such processes. A new example 6 has been added to illustrate measurement of graphical output.
(New section)	Editorial	New section 4.2.5 'Other examples' added which includes Examples 2, 5, 6 and 7 from section 4.2.3 in v1.2.
4.2.5	Editorial	Example 2. A definition of an object of interest sub-type has been added. FUR 3 to FUR 6 have been partly re-written and expanded to improve clarity.
4.2.5	Editorial	Example 3a. The possibility of needing an error/confirmation message has been added for a functional process to enquire on a specific code, to make the example more realistic.
4.2.5	Editorial	Example 4. This has been changed significantly to make it more realistic. Both the client requirements and selected holiday details are now saved as a named set.
4.3.1	Editorial	The description of Examples 1 and 2 has been simplified to improve ease of understanding. A new Figure 4.10 has been added to show the context diagram for Example 2, cases (b) and (c).
(New Section)	Editorial	A new section 4.3.2 has been added to show how the COSMIC model of a distributed software system is affected by whether the exchange of data between components is by asynchronous or asynchronous communications.
Section 4.4 has been re-structured into a more logical order. Section numbers given in the first column below are for this version 1.3 of this Guideline.		
4.4.1	Editorial	The topic of 'application-general data' has been moved from this section to a new section 4.4.8 on 'Fixed text'
4.4.2	Editorial	This was section 4.4.6 in v1.2 of this Guideline. It has been re-written to make it clearer
4.4.3	Editorial	This was section 4.4.5 in v1.2 of this Guideline
4.4.4	Editorial	This was section 4.4.2 in v1.2 of this Guideline. It has been re-written to make it clearer.
4.4.5	Correction	This was section 4.4.3 in v1.2 of this Guideline. Text shown as Examples 5 and 6 in v1.2 has been abstracted and grouped with other text that describes differences of batch processing from on-line processing. The introduction has been corrected to acknowledge that a batch process may have to process a file of transmitted-in data, not just a file of temporarily-stored data.

4.4.6	Editorial Correction	This was section 4.4.4 in v1.2 of this Guideline. A new Example 1 has been added. The measurement of Example 4 in section 4.4.4 in v1.2 of this Guideline was and is correct, but the explanation of the measurement that the 'data uniqueness' rules apply to this Example was irrelevant and has therefore been removed. The example has been moved to become Example 3 in section 4.2.4. In this example, an Exit for an Error/confirmation message has been removed as it is unnecessary in the context.
4.4.8	Editorial	The definition of 'fixed text' has been broadened and several examples given to deal with various cases of fixed text that must or must not be measured A reference has been added to the 'Web Advice Module' case study [15] for more examples of the analysis of fixed text.
(New section)	Editorial	A new section 4.4.9 has been added on 'timeout' functionality, referring to the 'Guideline on sizing real-time software' [20].
(New section)	Editorial	A new section 4.4.10 has been added on how to measure 'Date/time' functionality obtained from the operating system.
4.5.1	Editorial / Correction	Example 1 has been changed to improve ease of understanding. Also a footnote has been corrected. An Exit must be counted for an error-confirmation message that has been changed in any way. This defect was reported in a Notice issued in September 2015.
4.5.2	Editorial/ Correction	A new section 4.5.2 has been added on how to measure changes to fixed text. The text is now consistent with the Measurement Manual, section [1]. Field headings that must be changed without changing the associated data may now be accounted for when measuring a change to a data group.
Glossary	Editorial	The term 'Key (attribute)' and its definition have been added. Other uses of the term 'key', e.g. as in 'the key to analyzing ….' have been eliminated from this Guideline to avoid confusion.

Appendix B

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]).

DM. Abbreviation for 'Data Movement'.

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.

Key (attribute) The one or more data attribute types whose values uniquely identify an occurrence of a data group type.

NFR. Abbreviation for 'Non-Functional Requirement' – see section 1.2.3 of this Guideline.

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@.cosmic-sizing.org

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.cosmic-sizing.org 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.

You can use the forum on cosmic-sizing.org/forums to post your questions and receive answers from our world-wide community. The quality of any answers will depend on the knowledge and experience of the community member that writes the answer; the MPC cannot guarantee the correctness. Commercial organizations exist that can provide training and consultancy or tool support for the method. Please consult the www.cosmic-sizing.org web-site for further detail.