



**The COSMIC Functional Size Measurement Method**

**Version 3.0**

# **Guideline for Sizing Business Application Software**

**VERSION 1.1**

**May 2008**

# ***ACKNOWLEDGEMENTS***

<b>Version 1.0 authors and reviewers 2005 (alphabetical order)</b>		
Alain Abran, École de Technologie Supérieure, Université du Québec, Canada	Ton Dekkers, Sogeti, The Netherlands	Jean-Marc Desharnais, Software Engineering Lab in Applied Metrics – SELAM, Canada
Peter Fagg,	Arlan Lesterhuis*, Sogeti, The Netherlands	Roberto Meli, Data Processing Organization, Italy
Pam Morris, Total Metrics, Australia	Serge Oigny, Bell Canada	Marie O'Neill, Software Management Methods, Ireland
Tony Rollo, Software Measurement Services, United Kingdom	Grant Rule, Software Measurement Services, United Kingdom	Luca Santillo, Agile Metrics, Italy
Charles Symons*, United Kingdom	Hannu Toivonen, Nokia Siemens Networks, Finland	Frank Vogelesang, Sogeti, The Netherlands

<b>Version 1.1 authors and reviewers 2008 (alphabetical order)</b>		
Alain Abran, École de Technologie Supérieure, Université du Québec, Canada	Arlan Lesterhuis*, Sogeti, The Netherlands	Marie O'Neill, Software Management Methods, Ireland
Luca Santillo, Agile Metrics, Italy	Charles Symons*, United Kingdom	Hannu Toivonen, Nokia Siemens Networks, Finland

\* Authors of versions 1.0 and 1.1

In addition to the above, comments arising from work on translating v1.0 of the Business Application Guideline into Japanese were used to help produce v1.1. These comments were submitted by:

Shin-ichi Nagano, NTT East, Japan  
 Taku Fujii, OGIS-RI Co., Ltd., Japan  
 Noboru Hirabayashi, Fujitsu, Japan  
 Yasunori Nagatani, Kinkei System Corp., Japan.

The authors would like to express their appreciation to Rabobank Nederland and to Sogeti Nederland b.v. for their assistance and contribution to the development of the Guideline, and to Software Measurement Services Ltd for the use of some of their case study examples.

Copyright 2008. All Rights Reserved. The Common Software Measurement International Consortium (COSMIC). Permission to copy all or part of this material is granted provided that the copies are not made or distributed for commercial advantage and that the title of the publication, its version number, and its date are cited and notice is given that copying is by permission of the Common Software Measurement International Consortium (COSMIC). To copy otherwise requires specific permission.

A public domain version of the COSMIC documentation and other technical reports, including translations into other languages can be found on the Web at [www.gelog.etsmtl.ca/cosmic-ffp](http://www.gelog.etsmtl.ca/cosmic-ffp)

# ***VERSION CONTROL***

---

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

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

## **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' [3] on how to apply the COSMIC method v3.0 of Functional Size Measurement (FSM) to size software from the domain generally referred to as 'business application' software'.

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 19761:2003 standard [5]. 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. This software domain is particularly important since it is the domain for which '1<sup>st</sup> generation' FSM methods, such as the IFPUG, MkII and NESMA methods, were designed.

## **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 therefore not tied to any particular development methodology or life-cycle.

Measurers who have used a 'first generation' functional size measurement method and who wish to migrate 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 an existing method may be familiar with a detailed 'recipe book' with examples of how to measure functional size in many different situations, whereas the COSMIC Measurement Manual concentrates on defining general principles and rules, and deliberately avoids many domain-specific examples.

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

Readers of this Guideline are assumed to be familiar with the COSMIC Measurement Manual, version 3.0 and any associated 'Method Update Bulletins' (all obtainable from [www.gelog.etsmtl.ca/cosmic-ffp](http://www.gelog.etsmtl.ca/cosmic-ffp)). For ease of maintenance, there is little duplication of material between the Measurement Manual and this Guideline.

## Scope of applicability of this Guideline

The content of this Guideline is primarily intended to apply to 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'.

For a full description of what characterizes 'business application software', see section 1.1.<sup>1</sup>

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, except general-purpose software tools (sometimes also described as 'applications') 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.

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

## Introduction to the contents of the Guideline

This Guideline focuses on examples that illustrate the principles and rules of the Measurement Manual and that help interpret them in the business application software domain. These principles and rules are not duplicated in the Guideline, except where necessary to support a new rule or example in the Guideline.

For definitions of the terms of the COSMIC method in general, please refer to the glossary in the Documentation Overview and Glossary [1]. Terms specific for the business application software domain can be found in the glossary at the end of the present Guideline.

Chapters 1 and 2 of the Guideline provide background material designed to assist measurers to identify functional user requirements and especially data requirements that are needed for measurement from software artifacts that are encountered in practice.

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

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

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

---

<sup>1</sup> An ISO Technical Report [6] gives two formal models that aim to distinguish the business application software domain from other software domains.

## **Introduction to the new version 1.1 of this Guideline**

This new version 1.1 brings this Guideline into line with version 3.0 of the COSMIC functional size measurement method, as defined in the 'COSMIC Method v3.0: Measurement Manual', [3]..

Version 3.0 of the COSMIC method introduced several simplifications. For example the suffix '-FFP' was dropped from the name of the method, and the unit of measure was changed from 'Cfsu' to 'CFP' (COSMIC Function Points). The concept of the 'functional user' was introduced, a simplification which meant that the concept of the 'End User Measurement Viewpoint' could be eliminated.

These changes and other suggestions for editorial improvements made it necessary to publish an updated version of this Guideline. For a summary of the main changes made in producing version 1.1 from version 1.0 of the Business Application Guideline, see Appendix B.

It must be emphasized that in updating to version 3.0 of the COSMIC method, no changes were made to the basic functional sizing rules. Consequently, in this version 1.1 of the Business Application Guideline, all the answers to the examples are identical to those given in version 1.0 of the Guideline (though the description of some examples has been changed slightly to make them clearer, as a result of comments received from COSMIC practitioners).

# TABLE OF CONTENTS

<b>1</b>	<b>THE FUNCTIONALITY OF BUSINESS APPLICATION SOFTWARE .....</b>	<b>9</b>
1.1	Characterization of business application software .....	9
1.2	Functional User Requirements.....	10
1.2.1	<i>The meaning of 'functional' .....</i>	<i>10</i>
1.2.2	<i>The evolution of FUR in a typical software development project life-cycle.....</i>	<i>11</i>
1.2.3	<i>System technical and quality requirements that evolve into software FUR.....</i>	<i>12</i>
1.2.4	<i>When the FUR lack sufficient detail to apply the COSMIC method.....</i>	<i>13</i>
1.2.5	<i>Measurement as a quality control on FUR .....</i>	<i>13</i>
1.2.6	<i>Who can specify Functional User Requirements?.....</i>	<i>14</i>
<b>2</b>	<b>INTRODUCTION TO DATA ANALYSIS .....</b>	<b>15</b>
2.1	Data modeling 'levels' .....	15
2.2	Data analysis principles.....	16
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 objects of interest .....	19
2.6.1	<i>Input, output and transient data structures .....</i>	<i>20</i>
2.6.2	<i>Parameter (code) tables and objects of interest.....</i>	<i>21</i>
2.6.3	<i>Additional criteria for identifying objects of interest.....</i>	<i>23</i>
2.6.4	<i>Summary: types of objects of interest.....</i>	<i>23</i>
<b>3</b>	<b>THE MEASUREMENT STRATEGY PHASE.....</b>	<b>25</b>
3.1	The purpose and scope of the measurement .....	25
3.1.1	<i>Examples of purposes and scope.....</i>	<i>25</i>
3.1.2	<i>Software in different layers .....</i>	<i>26</i>
3.2	Identifying the functional users.....	27
3.2.1	<i>The functional users of business application software .....</i>	<i>27</i>
3.2.2	<i>The boundary.....</i>	<i>27</i>
3.3	Identifying the level of granularity.....	28
<b>4</b>	<b>THE MAPPING AND MEASUREMENT PHASES .....</b>	<b>29</b>
4.1	Identifying functional processes .....	29
4.1.1	<i>CRUD(L) transactions.....</i>	<i>29</i>
4.1.2	<i>Elementary parts of FUR and functional processes .....</i>	<i>29</i>
4.1.3	<i>Screen design style and functional processes.....</i>	<i>30</i>
4.1.4	<i>Physical screen limitations.....</i>	<i>30</i>
4.1.5	<i>Separate functional processes owing to separate functional user decisions .....</i>	<i>30</i>
4.1.6	<i>Retrieve and update of data in a single functional process .....</i>	<i>31</i>
4.1.7	<i>Drop-down lists arising in functional processes.....</i>	<i>32</i>
4.1.8	<i>Measurement of apparently inter-dependent functional processes.....</i>	<i>32</i>
4.1.9	<i>The 'many-to-many' relationship between event-types and functional process types .....</i>	<i>33</i>
4.2	Identification of objects of interest, data groups and data movements.....	33
4.2.1	<i>Introduction .....</i>	<i>33</i>
4.2.2	<i>Identification of objects of interest, data groups and data movements.....</i>	<i>34</i>
4.2.3	<i>Examples .....</i>	<i>35</i>
4.3	Sizing components of business applications .....	41
4.4	Other measurement conventions .....	44

4.4.1	<i>Control commands and application-general data</i>	44
4.4.2	<i>Menus and the triggering Entry</i>	44
4.4.3	<i>Applications processed in batch mode</i>	45
4.4.4	<i>Multiple sources, destinations and formats of a data movement – applications of the ‘data uniqueness’ rule</i>	47
4.4.5	<i>GUI elements</i>	48
4.4.6	<i>Authorization, help and log functionality</i>	48
4.4.7	<i>Error and confirmation messages</i>	49
4.5	<i>Measurement of the size of functional changes to software</i>	50
4.5.1	<i>Examples of functionally changed functional processes</i>	50
4.5.2	<i>Data conversion software</i>	51
4.5.3	<i>Size of the functionally changed software</i>	51
<b>REFERENCES</b>		<b>52</b>
<b>APPENDIX A - COSMIC CHANGE REQUEST AND COMMENT PROCEDURE</b>		<b>53</b>
<b>APPENDIX B - MAIN CHANGES IN V1.1 FROM V1.0 OF THE BUSINESS APPLICATION GUIDELINE</b>		<b>54</b>
<b>GLOSSARY</b>		<b>55</b>

---

## THE FUNCTIONALITY OF BUSINESS APPLICATION SOFTWARE

### 1.1 Characterization of business application software

'Business application software' is distinguished by the following characteristics:

- 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 peer 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.
- Data are usually stored historically, i.e. after events happened in the real world, with on-line response times that are suitable for human interaction. 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 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' (as defined in the COSMIC method), 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.
- A piece of software that is regarded by its human functional users as a single 'business application' may be composed of several major 'peer components' which may be distributed over different computer processors (where each peer component resides in the application layer of its respective processor). Although this underlying structure will not be visible to the human functional users, if the purpose of the measurements is to provide sizes as input to an estimating process, it may be necessary to define separate measurement scopes for each peer component, for example, when different development or processor technologies are used for each peer component.

## 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 artifacts' 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 in [13] 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 is also an ISO definition of 'non-functional requirement', which is "a requirement that constrains the design of software, but does not describe a service the software is to provide."

These two definitions are not sufficient for all functional size measurement purposes. An example of a difficulty that might be encountered in practice would be a requirement for 'ease of use' of a business application. This might be interpreted:

- as a quality requirement (and therefore not a FUR) or as a constraint (and therefore 'non-functional'), or
- as an implied software requirement for a graphical user interface (with implied FUR), and hence also as 'a service the software is to provide' (and therefore not 'non-functional')

With the first interpretation, any functionality provided purely for ease of use would be ignored in the software functional size measurement but with the second interpretation some functionality resulting from the GUI interface might need to be considered in the measurement, depending on the FSM method's rules.

Each FSM method therefore has to set its own rules and give its own guidance on what functionality should be measured. For business application software, there is general agreement that the 'primary' business data processing functions of the software (e.g. enter, store and extract data about customers and orders, pay out on insurance policies, enquire on bank balances, etc) result from true FUR and these must be measured. However, uncertainty may arise on whether or not to measure what might

be termed the 'secondary' or 'overhead' functions of the software such as some functions of GUI interfaces, control functions such as security access or data logging, functions provided for future ease of maintenance of the software, etc.

In the COSMIC method, such uncertainty on whether some requirement is 'functional' or not must and always can be resolved by using the definitions of the method's basic concepts and its principles and rules:

The measurer must always apply the COSMIC model which requires FUR to be broken down into 'functional processes', each consisting of 'data movements', where a data movement moves a 'data group' containing attributes of a single 'object of interest'.

- Data movements either
  - enter a 'data group' from a functional user to the software (and may trigger a functional process), or exit a 'data group' from the software to the functional user, or
  - move a 'data group' to persistent storage or retrieve it from persistent storage.

Any such data movement is 'functional' for the COSMIC method and should be measured.

- Conversely, any movement
  - of data attributes of something that is not an object of interest to the functional user
  - or of any data attributes as an intermediate step in a functional process where those attributes neither cross the boundary between the software and its functional users, nor move to or from persistent storage

cannot be recognized as a data movement in the COSMIC method and must be ignored for measurement purposes.

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

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

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

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

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

This means that at any point in the life-cycle, the measurer must be able to derive the FUR from whatever artifacts of the software are available. Very often, initially there will only be an outline SOR. Later when the software is being built there will only be an agreed 'specification' and/or 'design document' that is sufficiently up-to-date to be used. For a piece of software that was implemented

years ago, there may be little up-to-date documentation available at all for measurement, just the installed system plus a user guide.

None of this matters in principle to FSM. Whatever the state of the software and its artifacts, it is always possible to derive a corresponding statement of the *implied* FUR, though the difficulty of doing this may be very variable in practice. This is the task of the measurer who must have a thorough understanding of the underlying concepts of the FSM method and of a process to analyze the artifacts 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 functional processes, e.g. create and update. FSM measures the separate logical functions of the software, not its physically implemented artifacts.

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

### 1.2.3 *System technical and quality requirements that evolve into software FUR*

As stated above, a typical system statement of requirements contains both functional user requirements for the software, and technical and quality (or 'non-functional') requirements for the software and other system resources, and they are often inter-leaved. FSM is only interested in software FUR, so we have to separate the FUR from the technical and quality requirements in the statement of requirements. Further, as a software development project progresses, some system technical and quality requirements will remain unchanged, but some may evolve into software FUR. Examples will illustrate both cases.

- a) Technical requirements to write the software in a given programming language or to execute it at a particular data centre will always remain technical requirements. A quality requirement that the software should have zero major defects in the first month of operation will always remain a quality requirement
- b) A statement of FUR may define (in a lot of detail) that a new system must enable a stockbrokers' clients to enquire on their portfolio of investments and their current value over the Internet. Early in the project, a target response time is specified as a technical requirement. Further study shows that the response time requirement can only be met by developing the system to run on a certain hardware platform and also by providing continuous feeds of stock market prices to the portfolio enquiry system. The original technical requirement is still valid, but it now results in an additional technical 'constraint' (specified hardware) *plus* the FUR of some new application software to receive the feeds of stock market prices. When such a change occurs, the number of software functions to be built will increase and hence the functional size of the software that must be built will inevitably increase<sup>2</sup>

---

<sup>2</sup> We assume in this example that the solution to the problem of how to achieve the response time target has been agreed with the sponsoring user and hence the additional FUR to provide continuous feeds of stock-market prices are true software FUR, and not just a developer's implementation decision. The size of the additional FUR must then be included in the functional size of the application. The same assumption applies to the following example c) about GUI features.

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.

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

So although requirements evolve as projects progress and some original technical and quality requirements may evolve into or result in additional FUR, *at any time when a measurement is needed* we must aim to be able to say "these are the FUR of this software", or "this is its logical functionality which implies certain FUR at this point in time". It does not matter to FSM that some software FUR happened to start their life as system technical and quality requirements.

Where a measurement is being undertaken in the context of a software contract, and when a technical or quality requirement evolves resulting in additional, agreed, measurable software functionality within the defined scope of the measurement, the circumstances leading to the increase in functional size should be documented as part of the measurement.

#### 1.2.4 *When the FUR 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. Without the detail it may not be possible to apply the COSMIC method as it is defined in the Measurement Manual, but an estimate of functional size may still be needed, for example for an investment decision.

To deal with this situation, it is possible to use approximation variants of the COSMIC method. Possible approaches for approximate sizing using COSMIC are given in the 'COSMIC Method v3.0: Advanced and Related Topics' document [4].

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 quote 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 not written very clearly; they may be ambiguous, there may be inconsistencies or omissions and there may be plain errors. Most often, software eventually gets built to the customer's satisfaction only because of informal verbal agreements between developers and the customer, and the documentation will probably never be properly corrected to reflect reality.

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 the COSMIC method is that the discipline of applying the method helps identify inconsistencies, ambiguities, errors and omissions. In other words the measurement process is also a very good quality-control process. If a specification cannot be measured it is almost certainly unreliable in some way. The measurer can then add great value pointing out the defects (inconsistencies, anomalies lack of clarity, etc), where they occur and why.

Frequently, lack of clarity or anomalies in the documentation, or uncertainties 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 measured size should be quoted with an upper and lower limit of uncertainty, just as described above for approximate sizing.

### 1.2.6 Who can specify Functional User Requirements?

Any of the following categories of people can potentially generate FUR for a business application.

- 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. The physical media are the objects about which data is to be maintained. Other examples would be FUR about functional processes for backup or conversion and FUR about technical (error/confirmation) messages that are relevant to business users<sup>3</sup>.

---

<sup>3</sup> Note that the FUR of a business application should not encompass functionality that may be 'of interest' to non-business staff but which provide a technical solution to a business user's requirement. An example of this kind would be a functional process maintaining a file with intermediate counts, set up to accelerate the process of reporting to the business users

---

## 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 references to 'object of interest', etc. refer to *types* of these, not occurrences (or instances). 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') describing a single object of interest. It is therefore almost inevitable that some data analysis will have to be used to identify the objects of interest and hence the data movements that have to be measured in any given functionality.

### 2.1 Data modeling 'levels'

Data analysis methods usually define and distinguish 'data models' at different 'levels'<sup>4</sup> [7]. For our purposes it is sufficient to define the three modeling levels, as follows:

- 'Conceptual'. This level shows things<sup>5</sup> in the real-world that are important for a piece of software and the relationships between them
- 'Logical'. This level shows the things in the real world and the relationships between them that are relevant to the software and the data groups describing these things and their relationships
- 'Physical'. This level shows the actual data records and their relationships, of the files or databases that will be managed by the actual software that will hold data about the things in the real world.

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

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

---

<sup>4</sup> 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'

<sup>5</sup> 'Thing' is the word we use to represent literally anything for which software must process data; it may be a physical thing or a conceptual thing

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

## 2.2 Data analysis principles

We will discuss three of the most widely-used data analysis methods, namely Entity-Relationship Analysis (E/RA), Class diagrams as proposed by the Unified Modeling Language (UML) and Relational Data Analysis (RDA). Readers who normally use only one of these methods are nevertheless encouraged to read the whole of this chapter.

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

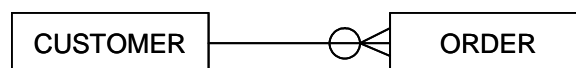
As a general rule, these ‘things’ correspond to ‘objects of interest’ in the COSMIC method, but it is not always so and it is vital to understand why there can be differences. As an example, both E/RA and UML Class diagrams allow the data analyst to show only the ‘things’ that are of interest and only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

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

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

## 2.3 E/R Analysis

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



**Figure 2.3.1 - E/R diagram showing data about customers and orders with a one-to-many relationship**

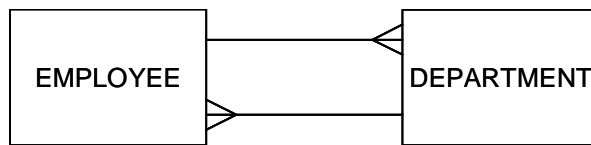
The symbol on the line showing the relationship between the two entity-types Customer and Order indicates that a Customer may be associated with zero, one or more Orders at any given time. Conversely, an Order can be associated with one and only one Customer. Both entity-types will be objects of interest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

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

Consider the following functional user requirements:

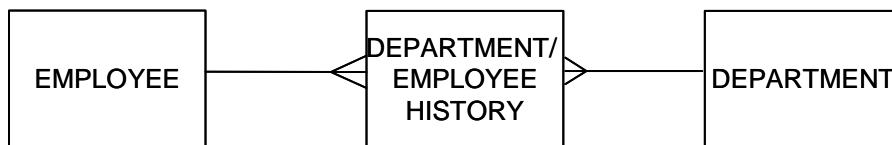
“Data must be stored and maintained about employees and the departments in the organization in which they have worked. From this data the software must be able to supply the employment history of any given employee, that is, the list of departments in which this employee has worked in the past and the 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 2.3.2 - 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’ 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 2.3.3 - 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 some thing in the real-world and typically has its own attributes. For the purpose of measuring the functional size of this FUR, the COSMIC method would identify three distinct objects of interest in this example.

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

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. A case might be a requirement for a matrix to record that there is a relationship between certain columns and certain rows, for example to keep track of which pupils have completed which assignments in a school course by entering a tick as each pupil completes each assignment. In this example, the ‘pupil/assignment’ relationship is an object of interest.

## 2.4 UML class diagrams (and use cases)

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

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

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

Contrast the above with an object of interest (type) which is 'a thing ... in the world of the functional user about which data is processed and/or stored'. In the COSMIC Generic Software Model, an object of interest is only described by its data attributes.

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

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

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

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

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

In this context, as UML 'use cases' are widely used, it is important to point out that a use case does not necessarily correspond to a COSMIC functional process. A use case construct is 'used to define the behaviour of a system or other semantic entity without revealing the entity's internal structure.

Each use case specifies a sequence of actions, including variants, which the entity can perform, when interacting with actors<sup>6</sup> of the entity' [10].

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

## **2.5 The normalization process of RDA**

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

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

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

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

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

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

In almost all cases, entity-types that may be identified from E/R analysis, classes shown on UML class diagrams and the subjects of relations identified from RDA will be COSMIC objects of interest. But experience shows that cases arise in practice where data analysis methods do not consistently lead to

---

<sup>6</sup> 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.

the correct identification of objects of interest. Other criteria are needed in addition to the rules given in the Measurement Manual. The cases are described in the next two sections and the additional criteria in section 2.6.3.

### 2.6.1 *Input, output and transient data structures*

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

However, for COSMIC purposes, we also need to apply these same principles to the data structures of the input and output components of functional processes to identify the objects of interest that are the subjects of the data groups of the Entry and Exit data movements. For many functional processes, the Entries and Exits will be entering or displaying data that will become or is already persistent (e.g. enter data about a customer, or enquire on an employee, respectively). So in such cases the data analysis of the input and output data structures is the same as for the persistent data structures.

But many enquiry functional processes, e.g. in management information reporting systems, generate derived data that is not stored. The Entry (occasionally Entries) of the input and the one or more Exits that make up typical outputs of such functional processes will form structures of transient data groups, i.e. data groups that do not survive the functional process in which they occur.<sup>7</sup>

Consider the following example. Suppose there is a Functional User Requirement to develop an enquiry to calculate and display:

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

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

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

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

As a cross-check on this conclusion, E/RA would say that ‘the goods sold to a customer of any one type in a period’ is a different ‘thing’ from ‘the goods sold to all customers in the period’. In this example, both ‘things’ are clearly really different ‘physical .... objects .... in the world of the user about which the software is required to process ... data’, as per the definition of an object of interest.

The same conclusion is reached by applying RDA to the enquiry output. The result will show three occurrences of sales value by customer-type in the period (for P, R and W customers) and one value of sales for all customers in the period. Step 3 of the RDA normalization process will conclude that the data about goods sold by the three customer-types is a repeating group and is therefore in a different relation from that of data about ‘all customers’.

---

<sup>7</sup> To clear up one possible source of confusion: the output report or the output screen displaying the result of an ad hoc enquiry is NOT the object of interest that we are discussing here. It is the data on the report or on the screen that must be analysed to find out the objects of interest. In the business application software domain, the measurer should focus on the data of the FUR, not on the data about physical devices, documents or media.

We must also remember that the Entry to this enquiry conveys the parameters of the enquiry, e.g. the start and end dates, which also comprise a transient data group. These are the attributes of a 'time-period' object of interest.

### 2.6.2 Parameter (code) tables and objects of interest

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

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

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

- a) Suppose an order-processing application that stores a lot of data about customers, e.g. customer-ID, customer-name, customer-address, customer-contact telephone, customer-credit-limit, customer-type-code, date-of-last-order, etc., all of which must be entered.

'Customer' is clearly an object of interest to many functional users, e.g. to the staff on the order desk, in this system. So the simplest functional process to enter data about a customer would have one Entry and one Write for the data entered about the Customer object of interest, and probably one Exit for error/confirmation messages<sup>8</sup>. Total size: 3 CFP.

- b) Notice the attribute 'customer-type-code' in the above list. It can have several code values, e.g. P, R, W, etc., standing for Personal, Retailer, Wholesaler, etc respectively. Suppose this same order-processing application also stores other data about this customer-type 'thing' e.g. 'customer-type-description', 'customer-type-order-value-discount-%', 'customer-type-payment-terms, etc.

Customer-type is now also a 'thing' with its own attributes and in this system it is an object of interest to functional users who set the policy on commercial terms and maintain the attributes of each customer-type, as well as being an object of interest to the functional users on the order desk.

In this case, therefore, when entering data about a new customer, the customer-type-code is not just an attribute of Customer, but it provides a link to other attributes such as 'customer-type-order-value-discount-%', etc., that are highly relevant (i.e. *of interest*) for any functional user concerned with customer relations. These attributes of the customer-type object of interest become indirect attributes of the Customer object of interest.

So the simplest functional process to enter data about a new customer described in case a) above would now, in case b), have to include a Read of the data of the customer-type object of interest, in order to validate that a correct customer-type-code has been entered.

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

---

<sup>8</sup> Almost all business application software for which human users enter data or prepare data for entry is required to output messages to inform the user when entered data has failed a validation test and/or to confirm that data has been processed satisfactorily. Such messages are all occurrences for an Exit that we refer to as 'error/confirmation messages' in this Guideline. See section 4.4.7 for a discussion of such messages.

an attribute of (i.e. data *about*) a Customer (we are not entering data *about* customer-type in this functional process).

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

- c) As an alternative to case b), suppose that the customer-type 'thing' only has two attributes 'customer-type-code' and 'customer-type-description'. These two attributes might have pairs of values P, Personal; R, Retailer, etc., respectively.

When data must be entered about a new customer, and the point is reached in this data entry functional process where 'customer-type-code' must be entered, suppose the system displays a list of valid 'customer-type-descriptions'. The user then selects the appropriate description and the corresponding customer-type-code is entered and stored as an attribute of customer.

In this case, in the functional process to enter data about a new customer, there is no reason for the functional user to consider customer-type as an object of interest, as in case b). 'Customer-type-code' is only required to be stored as an attribute of customer and in this customer data entry functional process, no data is processed about *'customer-type'*. 'Customer-type-code' is involved only in data being processed about *customer*.

The software does, of course, need to store values of the attribute customer-type-code for the computer's needs and the equivalent customer-type-description for the human user's needs, but these are needed only to ensure that a valid customer-type-code value is entered for each customer. (It is immaterial whether values of these attributes are stored in e.g. a code table maintenance software system or are hard-coded.) Customer-type is not an object of interest for business users of this functional process in this case c) as it is in case b).

For functional processes such as this customer data entry process, no data movement should be identified for the references to data attributes of customer-type and the total size is the same as for case a), i.e. 3 CFP.

- d) Continuing with example c), suppose now that values of the two attributes 'customer-type-code' and 'customer-type-description' are 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'.)

Such parameter table maintenance software would have functional processes to create new customer-type codes and descriptions, and to update, delete and read them.

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

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

Note: the conclusion that customer-type is not an object of interest for the FUR of case c), but is for the FUR of case d) holds valid whether or not these two sets of FUR happen to be within the same measurement scope or not.

In summary, we see from these examples that:

- To the business functional user, customer-type is or is not an object of interest, simply depending on whether it has its own attributes (as in case b) or not (as in cases a) and c). In cases a) and c), customer-type-code is simply an attribute of Customer.
- To the non-business functional user, when attributes of customer-type must be maintained as system parameters by functional processes, then customer-type is an object of interest to those non-business users, in those functional processes.
- As a consequence, different types of functional users may 'see' different objects of interest in their respective functional processes of the same application. See also the example 7 in section 4.4.3.

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 functional processes to be in separate scopes. In an old cliché of E/RA, 'one man's entity is another man's attribute'.

### 2.6.3 Additional criteria for identifying objects of interest

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

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

- a) to determine whether the thing is really 'of interest', i.e. that it would be specified in a Functional User Requirement (FUR) as a thing about which the software is required to process and/or store data, and
- b) to recognize that whether a 'thing' is an object of interest or not may depend on the (type of) functional user for whom the FUR are specified. A 'thing' may be an object of interest in some FUR for certain functional processes for certain types of functional users, but may not be an object of interest in other FUR for certain functional processes for other types of functional users. (Note: it is possible that these different views on what are the objects of interest can exist within one collection of FUR covered by the same measurement scope.)

Entity-types identified in entity-relationship analysis and the subject of relations in Third Normal Form arrived at from Relational Data Analysis are almost always candidates for objects of interest. But the transient data of the input and output parts of functional processes must be analyzed as well as the persistent data to identify all the objects of interest. And only an examination of the FUR (or, in the absence of any documented FUR, the functional processes that have been implemented, carefully taking into account the intended meaning of an object of interest) can determine if some 'thing' is an object of interest or not to the functional users.

### 2.6.4 Summary: types of objects of interest

In the following it must be kept in mind that some 'thing' is an object of interest only if it satisfies the definition of an object of interest and the criteria of section 2.6.3. The purpose of this section is to draw the attention of measurers to the many different types of objects of interest that may be found in business application software. The discussion so far has been illustrated by examples that are all clearly 'data' objects of interest. We have discussed:

- The '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 attributes change very frequently, as a result of the normal business processes
- The 'secondary' (or 'non-core') objects of interest about which business applications need to maintain persistent data in support of the 'primary' objects of interest. We have seen an example in section 2.6.2, case b), where 'customer-type' is an object of interest with a few important attributes, e.g. 'customer-type-order-volume-discount-%'. Other examples would be:

- in a payroll application, tables of salary bands by grade, or of income-tax bands with validity dates
- in an accounting system, tables of budgeted exchange rates by country
- in a manufacturing control system, tables of work-centres and their capacity, or of job-types and required skill levels
- in a banking system, savings interest rates by product, by level of deposit, with validity dates

'Secondary' objects of interest tend to have low volumes of data that change relatively infrequently.

- Objects of interest about which only transient data ever exists. Data about these are either input by a functional user or are derived by the application; they are not stored persistently, and appear only in the input and output of enquiries
- 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). But 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 types 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. 'Country' may be secondary, or not an object of interest at all to a manufacturing company, but primary to a freight-forwarding company. 'Currency' and 'Currency exchange rates' will be primary for a bank and either secondary or of no interest at all to the applications of a city administration. 'Department' may be primary for an application that maintains an organizational structure but only a parameter for the asset register. We mention these distinctions only because such classifications are often used and it helps us understand the extent of what may be considered as objects of interest.

When discussing objects of interest for which 'data' are held in normal language, 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.

---

## THE MEASUREMENT STRATEGY PHASE

The reader is assumed to be familiar with the chapter on Measurement Strategy of the Measurement Manual.

The importance of agreeing the 'Measurement Strategy', before starting an actual measurement cannot be over-emphasized. In summary, the Measurement Strategy considers four key parameters of the measurement that must be addressed, namely the purpose and scope of the measurement, the identification of functional users and the level of granularity of the FUR that should be measured. These parameters will be treated in the following sections.

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

- a) If the purpose is to measure the work-output of a project that must develop some application software and also some software that will reside in other layers, then a separate measurement scope must be defined for each piece of software in each layer.
- b) If the purpose of the measurement is to determine the software project productivity (or other performance parameter) and/or to support estimating, and the business application is one piece of software running on one technical platform, then the scope will be the whole application.

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

- c) If, as in the previous example, the purpose of the measurement is related to performance measurement or estimating, *and the application must be distributed over different technical platforms*, then it will be desirable to measure separately the size of each part of the application that runs on a different platform. (Each part is called a 'peer component' of the application). This can be achieved by defining a separate measurement scope for each of the peer components and the performance measurement or estimating can be carried out separately for each peer component on each platform. (The reason for this is that it is much easier to interpret performance data, or to estimate, for a piece of software that is (to be) developed on a single technical platform than for one that is (to be) developed for multiple platforms.) Note, however, that measuring the performance per platform pre-supposes that the effort per platform as well as the size will be or has been registered separately

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

- d) If the purpose is to measure the total size of an application portfolio (excluding any infrastructure software) in order to establish its financial value, e.g. at replacement cost, then it may be sufficiently accurate to measure sizes ignoring any functionality arising from any distributed

architecture. A separate scope should be defined for each application to be separately measured. (Sizes may also be measured to sufficient accuracy using an approximation variant of the COSMIC method to speed up this task<sup>9</sup>.) An average productivity for replacing the whole portfolio can then be used to determine the replacement cost

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

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

### 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.

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. For software to reside in different layers, all the aspects of the definition, principles and rules for distinguishing layers, must be satisfied (see the Measurement Manual). The two most important of these are that for two pieces of software to reside in different layers, there is a 'superior/subordinate' hierarchical dependency between both pieces of software and both pieces of software must interpret the data they exchange differently.

Examples:

- a) A file copy utility may interpret the data of an application differently from that of the application, but there may not be a hierarchical relationship between them. If the application can function without the file copy utility and vice versa, then both pieces of software would be in the same application layer
- b) A piece of general security access control software may determine which users can access which applications, and even which transactions within each application. An application may only be able to execute if control is passed to it by the security access control software, so in a sense there is a hierarchical relationship between the two pieces of software. But if the data exchanged between the security access control software and each application is viewed identically, then they must be regarded as in the same application layer
- c) 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. The logging function can operate without any specific application, but not vice versa – an application set up to use logging cannot execute without the logging function. So there is a hierarchical relationship between them and they will inevitably interpret differently the data that is logged. So the logging function is in a different layer to the application.

For examples a) and b) above, it is an entirely separate question as to whether these support utilities should or should not be within one measurement scope, together with the business application. The scope depends only on the purpose of the measurement. The purpose might be to measure only the

---

<sup>9</sup> See the document 'COSMIC Method v3.0. Advanced & Related Topics' for approximate sizing variants of the COSMIC method.

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

### 3.2.1 The functional users of business application software

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

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

Note that staff who maintain software programs or who maintain stored data or rules via utilities only available to those with computer software knowledge are NOT considered as 'functional users' of the software.

### 3.2.2 The boundary

When carrying out measurements on software in the business application software domain we identify the data movements across the input/output ('I/O') interfaces – the boundary – which lie between the application and its functional users, as illustrated in the figure below. The I/O devices and any supporting infrastructure software that enable the functional users to communicate with the application being measured are 'invisible' to the functional users and are ignored.

Data movements to and from persistent storage are also identified from the perspective of these functional users but these take place 'inside' the software, i.e. they do not cross a boundary. The functional users are not aware of any persistent storage devices; functionally, all the functional user wants is for data to be made persistent, or to be retrieved from persistent storage.

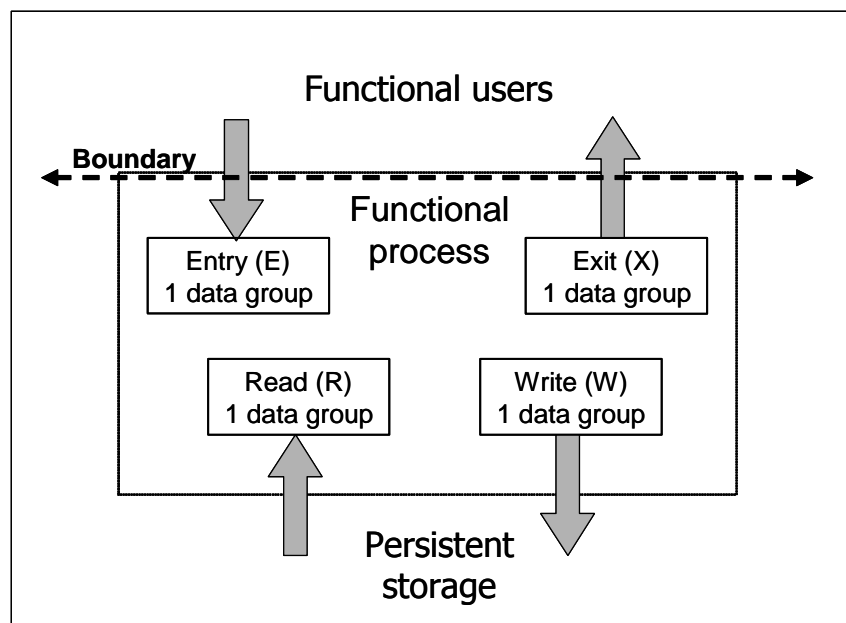


Figure 3.2.1 - The components of a functional process and some of their relationships

### 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. Furthermore, the FUR, as far as they exist, may be at varying levels of granularity.

Consequently, the first measurement task is to check the level or levels at which the FUR exist. Then, if the FUR 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.

The Measurement Manual [3] provides a detailed example (the 'Everest' case) of situations that may be found in practice when a measurement must be made on FUR that have been specified at varying, 'high' levels of granularity. The 'Advanced and Related Topics' document [4] describes various approaches for approximate sizing of FUR at high levels of granularity.

---

## THE MAPPING AND MEASUREMENT PHASES

In the Mapping Phase, the FUR of the software to be measured must be mapped to three main concepts of the COSMIC method, namely to functional processes, to objects of interest and to data groups. In the next Measurement Phase, knowing the functional processes and data groups, the individual data movements of the functional processes can then be identified, as the basis for measuring the functional size. As almost all examples treat both the functional processes and data movements together, both phases are considered in this one chapter.

### 4.1 Identifying functional processes

The reader is assumed to be familiar with the section 'Identification of functional processes' of the Measurement Manual [3].

Various cases can help understand how to distinguish the functional processes of business application software.

#### 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. This rule is summarized by the acronym 'CRUD' where C = Create, R = Read, U = Update and D = Delete (sometimes known as 'CRUDL' where L = List). Data about every object of interest must be created, is invariably read, and will usually be updated and deleted, and maybe listed.

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

In practice, for some objects of interest, the FUR might specify several Update functional processes corresponding to different stages in the life-cycle of the object of interest. For example the FUR might specify separate functional processes to change the data of a person as he/she moves between the states of single, married, widowed and divorced because of the need to add, modify or delete various relationships. Alternatively, the FUR might, of course, 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, of course, also has requirements for enquiry or management reporting functional processes that produce (or create) transient data via Reads of the persistent data.

#### 4.1.2 *Elementary parts of FUR and functional processes*

To identify the functional processes in the Functional User Requirements 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 Functional User Requirements corresponds to a functional process. For an elementary part of a FUR to be a functional process it is necessary

- to be independently executable and
- to be triggered by an event in the world of the functional users and
- that when the elementary part has been executed, the software has 'executed all that is required to be done in response to the triggering event'

To be 'independently executable' is not enough by itself. If two parts of the FUR, A and B, appear to be independently executable, but A is triggered by an event outside the boundary, whereas B can only be triggered by A, then A+B forms *one* functional process. 'Triggered' means that a functional user – by definition outside the boundary – can be identified that detects an event and then triggers the functional process. See also section 4.1.9 'Measurement of (apparently) interdependent functional processes'.

#### 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 screen could account for the implementation of several functional processes. A separate functional process must be identified for any of these logically distinct functions that are mentioned in the Functional User Requirements as triggered by separate events.

#### 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.

#### 4.1.5 *Separate functional processes owing to separate functional user decisions*

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

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

Logically, the enquire-before-update and the update are two separate, self-contained functional processes. The user may or may not decide to proceed with the update functional process, depending on the outcome of the enquire-before-update. The update is entirely independent of the enquiry.

The FUR for an enquire-before-update can even require two separate functional processes, as follows. The task is to update an employee record, where the user knows the employee's name but not the unique employee ID. First in FP1, the user is invited to list all employees by name:

E	Request 'list employees'
R	Employee data
X	Employee data (e.g. only name and ID, sufficient to choose the desired employee)
X	Error/confirmation messages

Size of FP1 = 4 CFP

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

E	Employee ID (= selecting the desired employee)
R	Employee data
X	Employee data (detailed form, all attributes)
X	Error/confirmation messages

Size of FP2 = 4 CFP

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

E	Updated employee data
W	Employee data
X	Error/confirmation messages

Size of FP3 = 3 CFP

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

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

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

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

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

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

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

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

#### 4.1.7 Drop-down lists arising in functional processes

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

- a) The drop-down list displays the valid values of the attribute Z of the object of interest A of the Entry, without needing to refer to the data of any other object of interest to obtain these values. In this case the preparation and display of the drop-down list does not involve any other data movements and the existence of the drop-down list is ignored when analyzing the Entry.
- b) The drop-down list displays the valid values of the attribute Z of the object of interest A of the Entry. These valid values are obtained from an attribute of another object of interest B. Use of the drop-down list for value selection is mandatory. In this case identify a Read and an Exit for the retrieval and display of the list, i.e. a total of 2 additional CFP in this functional process FP1.
- c) The functional user is offered a choice of two ways of entering the value of the attribute Z of the Entry A, depending on whether the functional user knows the valid value to be entered or needs to select a value from a valid list. Hence (i) either the functional user enters the value directly and the value is validated against an attribute of an object of interest B, or (ii) the functional user may optionally choose to display the valid values of the attribute in a drop-down list and then select the correct value, where these valid values are obtained from an attribute of another object of interest B. In this case for alternative (i) identify one additional Read (of object of interest B) in the functional process FP1 for the validation of the direct entry of the value of the attribute of object of interest A of the Entry AND for alternative (ii) recognize a separate functional process FP2 for the optional display of the list of valid values of the attribute. This separate functional process FP2 has size 3 CFP (1 Entry, 1 Read, 1 Exit)

Note 1: The object of interest B must be a 'genuine' object of interest. See sections 2.6 and 4.2 for more guidance and examples on distinguishing objects of interest.

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

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

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

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

#### 4.1.8 Measurement of apparently inter-dependent functional processes

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 Functional User Requirements (FUR) for two functional processes FP1 and FP2.

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

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

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

Two functional processes are identified:

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

### Discussion

- a) The above solution results in part of the functionality of functional process FP1 being measured twice. This is correct because when measuring a functional size of software, we measure the size of the FUR, not the size of the physical transactions that the developer has chosen to implement. This is an example of the developer taking advantage of functional re-use, which is very common in software design.
- b) If the scope of the measurement in the above example is defined to include functional process FP2, but not functional process FP1, then when sizing functional process FP2, measure only the data movements of functional process FP2 within the scope plus any Exit/Entry messages across the boundary to maintain the creditworthiness indicator in the software 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 type may be triggered by one or more event-types.

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

The converse of this situation is also possible. A single event-type may trigger more than one functional process. This situation occurs in real-time systems where, for example, detection of an emergency condition may trigger separate multiple functional processes to execute in parallel to counter the emergency. This type of situation is uncommon in the domain of business application software, but cannot be excluded.

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

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

### 4.2.1 Introduction

We assume that for the given (stated or implied) FUR, we have made a first pass at identifying the functional processes using the guidance of section 4.1 and that we must now size each functional process by identifying the data movements of its input, process and output phases. In principle this

can be achieved by applying your preferred data analysis method to the input, process and output phases of each functional process in turn, to identify the various data groups that are moved. In practice, however, it is usually most effective to proceed as follows.

- First, construct a logical data model of the objects of interest of the persistent (stored) data within the scope of the measurement, using your preferred data analysis method.
- Then analyze the input, process and output phases of each functional process for the data groups to be sent to, or obtained from, persistent storage and for the transient data groups in the input and output
- Be prepared to iterate around these two steps, since analyzing the data of the input, process and output phases may result in identifying new objects of interest, and identifying a new object of interest may result in identifying new functional processes for its maintenance (see 4.1.1 on 'CRUDL transactions')

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

### **Warning on misleading attribute names**

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

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

In this example, the input parameters should really be named as e.g. 'ID of customer to whom sales made in period', 'ID of product sold to customer in period', 'Start date of period of sales', 'End date of period of sales'. There is one Entry for the enquiry.

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

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

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

### **The logical data model of the persistent data**

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

Then for each functional process

### **For the input phase of each functional process**

Identify one Entry for each unique data group in the input, regardless of whether some or all of the entered data is intended for persistent storage or not, or whether it is transient. (Remember we are seeking data group types; differing frequencies of occurrence always indicate differing data group types.)

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

Every functional process must have at least one Entry.

### **For the processing phase of each functional process**

For each object of interest for which persistent data is required to be retrieved or for which data is required to be made persistent, identify a Read or Write respectively (also bearing in mind the data movement uniqueness and possible exception rules; see the Measurement Manual).

### **For the output phase of each functional process**

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

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

Functional processes of business applications very commonly include an Exit for error/confirmation messages, though this is not always required.

#### *4.2.3 Examples*

(Key attributes are underlined)

#### **Example 1a – Simple enquiry**

We have a database with two objects of interest:

Client (Client ID, client name, address, ...).  
Order (Order ID, client ID, product ID, order date, ...)

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

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

E      Period start and end dates  
R      Order data  
X      Client ID, enquiry period start and end dates, number of orders.

In this example, the Entry moves a transient data group with the attributes 'enquiry\_period\_start\_date' and 'enquiry\_period\_end\_date' of one object of interest, 'time-period'. The Exit moves one transient data group, about an object of interest which could be defined as 'the business of a given client in the given time-period'. The key of this object of interest is (enquiry\_period\_start\_date, enquiry\_period\_end\_date, Client ID) and its only non-key attribute is 'the number of the client's orders placed in the period'. There is one occurrence of the Exit for every client that placed orders in the period.

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 3 CFP.

### Example 1b – More complex enquiry

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

The solution for this functional process is now:

E	Period start and end dates
R	Order data
R	Client data
X	Client ID, client name, client address
X	Client ID <sup>10</sup> , enquiry period start and end dates, 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. Moreover, applying RDA 3NF to the data required on the output tells us that we now have two Exits, one moving persistent data about the client (an object of interest) and one moving transient data about ‘the business of a given client in a given time-period’ (another object of interest). Hence there are two Exits each occurring once for every client.

The same solution applies even if the developer decides to lay out the data into a different printed form such as:

Period start and end dates (as a report heading)

Client ID, client name, client address, number of orders (one line for each occurrence of Client)

The size of this functional process is 5 CFP.

### 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 natural way to lay out the report would now be:

Client ID, client name, client address

Period 1 start and end dates, No. of orders placed  
Period 2 start and end dates, No. of orders placed  
Period 3 start and end dates, No. of orders placed

(These blocks are repeated for each client)

Now, in this example, nothing has fundamentally changed from example 1b. The objects of interest of both the Entries and Exits are the same as in example 1b. We simply have multiple occurrences of the same data. The size of this functional process is 5 CFP.

---

<sup>10</sup> In these examples, for convenience we use commonly-used, short-hand names for the data attributes. This results in the same attribute name ‘Client ID’ being used in the two Exits in this example which is confusing and technically incorrect, since they have different meaning and are therefore different attributes. ‘Client ID’ is a good name in the first Exit since the attribute is the key to this data group. But if we were being really precise, a better name for this attribute in the second Exit would be e.g. ‘ID of Client that has ordered’. See the ‘Warnings’ in section 4.2.1 for other examples that cause similar difficulties.

## Example 2 – A data entry functional process

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

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

Solution for this functional process:

E	Order header
R	Client data
E	Order item
R	Product data
W	Order header
W	Order item
X	Error/confirmation messages

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

## Example 3 – Simple enquiry with entered condition

Consider the following Functional User Requirement (FUR1):

“The software must support an ad hoc enquiry against a personnel database to extract a list of names of all employees older than a certain age, where that age must be entered.”

Solution for the functional process of FUR1:

E	The employee age limit (of the ad hoc enquiry)
R	Employee data
X	Employee names (for all employees within the given age limit)

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

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

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

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

If there were now a further requirement (FUR3) to output the total number of employees that satisfy the age-limit criteria in addition to the age limit (of FUR2), this would be a second attribute of the same data group that contains the age limit, and the size of the functional process satisfying FUR1 + FUR2 + FUR3 would be unchanged at 4 CFP.

#### **Example 4 – Report with multi-level aggregations**

Consider the following set of Functional User Requirements:

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

Solution for the functional process that satisfies these FUR:

E	Selection of the report
R	Employee data
R	Department data
X	Employee names (grouped by department within division)
X	Department employee subtotal
X	Division employee subtotal
X	Company employee total

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

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

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

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

#### **Example 5 – Functional processes with object of interest sub-types**

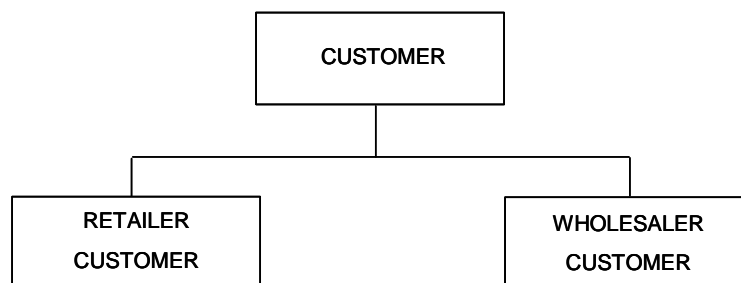
Sometimes, separate objects of interest should be recognized for sub-types of a particular object of interest.

- a) Suppose (FUR1) the object-class or entity ‘Customer’ has an attribute that indicates its type as ‘Personal’, ‘Retailer’, or ‘Wholesaler’. If the rules governing the processing of the data of all customers are the same, and all customers have the same data attributes, then these three types are NOT regarded as sub-types of Customer. ‘Customer-type’ is simply an attribute of Customer.
- b) Alternatively suppose the following FUR2.

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

Given that Customers have some common attributes, but also have different attributes according to their Customer-type, it follows that Personal, Retailer and Wholesaler are sub-types of Customer. Logically, therefore, the object of interest ‘Customer’ has three sub-type objects of interest (‘Personal Customer’, ‘Retailer Customer’ and ‘Wholesale Customer’).

The physical database structure would probably be as shown below. 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 4.2.3 – Object of interest type with its sub-types**

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

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

FUR4. “A functional process is required that enables entry of orders for a Personal Customer.” This functional process would only need to Read the object of interest ‘Personal Customer’ in order to validate the existence of the particular Personal Customer placing the order.

FUR5. “A single functional process is required to enable entry of orders for any Retailer or Wholesaler Customer.” This functional process would need to distinguish two separate objects of interest (Retailer Customer and Wholesaler Customer) in order to validate that the Customer exists and that orders can be accepted, and then to apply the correct discount rules for pricing the order. So this functional process FP3 would have two Reads for these sub-types.

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

The general principle is that where there is a need to distinguish more than one sub-type in the same functional process, each sub-type must be treated as a separate object of interest.

## Example 6 – Maintenance of parameter tables

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

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

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

Several situations may arise.

- a) Parameter tables are typically provided with a set of ‘CRUD’ (Create, Read, Update, Delete) functional processes to maintain the parameter values, which must be available to the non-business user.

In the simplest possible case, one set of CRUD functional processes might be provided to maintain all parameters in one table. For example, to change any one existing parameter occurrence, the non-business user would have to display the parameter table contents and page through them to find the particular parameter occurrence and its attributes to be maintained. Effectively there is only one object of interest to the non-business user, namely ‘parameter’. Each of the four CRUD functional processes would have size 3 CFP (1 Entry, 1 Write or Read, 1 Exit), making a total of 12 CFP for the CRUD set, assuming no error/confirmation message.

- b) 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. The size of this utility would then be at least (no. of parameter-types x 4 functional processes x 3 CFP per functional process).

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.

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, one for the code/description

data entry), 1 Write and 1 Exit. 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 requirements for the 'Read' functional processes, there are endless possibilities, 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.

### 4.3 Sizing components of business applications

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

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

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

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

#### Example 1

Suppose a simple two-component, client-server application, for example component A executes on a PC front-end that communicates with component B on a main-frame computer holding some 'legacy' data describing one object of interest. The (human) functional user triggers a functional process on the PC that requires this data to be read from the main-frame and displayed on the PC.

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

The (human) functional users of the application have no knowledge of how the application is physically distributed over the PC and the main-frame. The split of the application into two components is invisible, as also are the data movements between the two components. Similarly, any additional functionality in lower layers needed to achieve the communications between the PC and the main-frame is invisible and is ignored. The data movements of this functional process are then as shown in the figure 4.3.1 below (total 3 CFP):

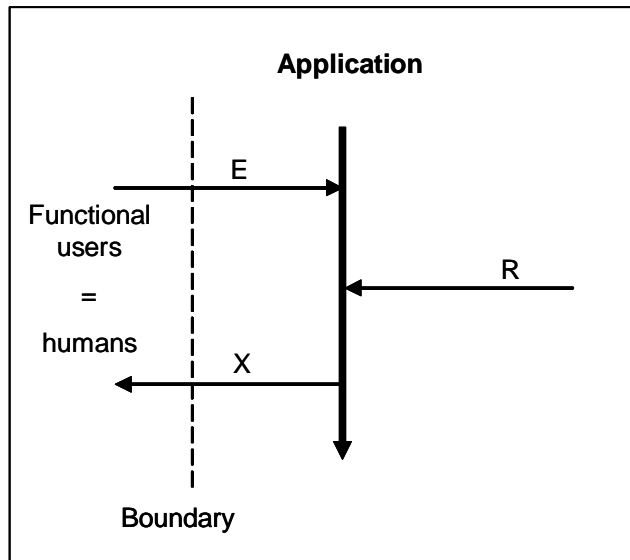


Figure 4.3.1 - Scope of the measurement is the whole application

*Case (b) Scope of the measurement is component A*

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

But limiting the scope of the measurement to component A results in the functionality being revealed that is actually needed to obtain the data from component B rather than by a local Read - see figure 4.3.2 below. Component A must issue a 'request to obtain' (or 'get' command) with the data selection criteria as an Exit to component B and receive back the required data from component B as an Entry. (Component A has no knowledge of how component B obtains the requested data; it could be via a Read, or by local calculation or from some other software.)

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

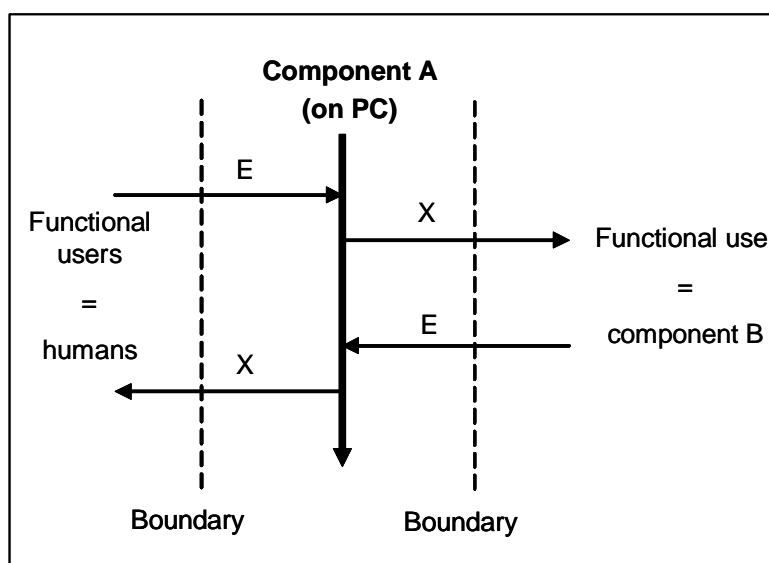
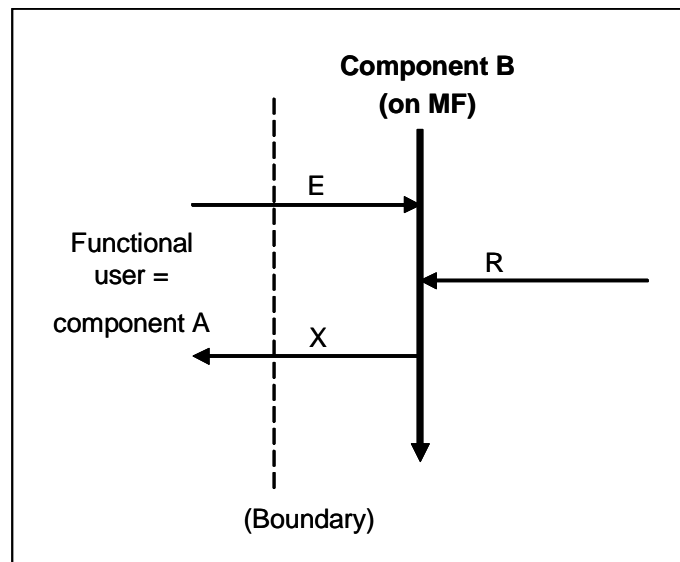


Figure 4.3.2 - Scope of the measurement is component A

*Case (c) Scope of the measurement is component B*

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

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



**Figure 4.3.3 - Scope of the measurement is component B**

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

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

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

**Example 2**

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

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

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

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

- An Entry from the clock to trigger the process (physically on the server)
- One or 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 (physically on the laptops)

*Case (b) Scope of the measurement is component A*

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

*Case (c) Scope of the measurement is component B*

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

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

## **4.4 Other measurement conventions**

### *4.4.1 Control commands and application-general data*

'Control commands' are defined as 'commands that enable the functional user to control their use of the software but which do not involve any movement of data about an object of interest'. Control commands must be ignored in the business application software domain.

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

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

### *4.4.2 Menus and the triggering Entry*

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

But often a menu command will launch a specific functional process, with or without input data for that process, for example an enquiry functional process. When a user presses a button on a menu to

launch a specific enquiry functional process P, a message goes to the software saying 'start this specific enquiry P'. If it happens that this enquiry also needs selection parameters which the user must input before pressing 'Enter', these are additional attributes of the same data group (but now the message is: 'start this specific enquiry P with these parameters'). In both cases this message is the one triggering Entry for this functional process. The object of interest is 'the subject of enquiry P'.

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

#### 4.4.3 Applications processed in batch mode

Fundamentally, when measuring the size of functional processes, it should make no difference whether the functional process is required to be processed on-line or in batch mode. But in practice the Functional User Requirements for processing in batch mode do sometimes necessarily differ from their equivalents for on-line processing. For example GUI features are unique to on-line processing, whilst 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.

The key tasks of the measurer in analyzing batch application streams are to decide what are the separate triggering Entries and hence the separate functional processes and who are the functional users involved in the batch processes.

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

- a) The input data to the batch process has been entered on-line by humans and has been accumulated in a file for batch processing. The processing of the file may be analysed as if the humans are the functional users
- b) A data file 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. The batch process may be analysed as if a human functional user provides the triggering Entry.

Note: a clock 'tick' or operator command that triggers the processing of a batch stream is a control command and is never measured in the business application software domain. Such a command launches the batch stream, not an individual functional process.

EXAMPLE 1 for Case a) The input data for a batch 'job' may typically consist of the Entries for several different functional processes. For example, the batch stream for an overnight order-processing system might contain functional processes to add new clients, add new and delete obsolete products, enter new orders, enter order cancellations, etc. Each of these different functional processes should be analyzed 'end-to-end' and independently of any other functional process in the same stream. Each functional process which runs in sequence once the job has started is triggered by its own Entry(-ies), and should be analyzed as if the data was being entered on-line by the human functional user.

EXAMPLE 2 for Case b) Application A, the software being measured, is required to transmit a file of 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 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.

Note: This functional process 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.)

EXAMPLE 3 of 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. 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. As an alternative to a single summary report, it might be that each functional process issues its own report. In general this should not affect the size measured.

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

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

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

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

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

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

For processing the header:

E	Interface file data	
R	Interface file history	(file already processed?)
W	Interface file history	(store result of processing)
X	Messages	

For processing the employee data:

E	Employee data	
W	Employee data	(data about an existing employee is overwritten)
X	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
- each functional process that enters and writes a specific file-type should be counted, i.e. there are as many of these functional processes as there are interface file-types to be entered in the application (assuming the validation and processing is different for each file-type).

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

The Measurement Manual states<sup>11</sup> that, exceptionally, different data group types describing a given object of interest may be required (in the FUR) to be moved in a data movement of the same type in the same functional process. Alternatively, and again exceptionally, the same data group may be required to be moved in the same data movement type in the same functional process, but with different associated data manipulation.

As a consequence of this rule, requirements for different formats or different manipulations of a given data group when it is sent to multiple destinations or received from multiple sources must be treated as different data movements. It is not the multiplicity of destinations or sources, as such, that determine if we have different data movements, but the multiplicity of formats or manipulations required (typically by different functional users) in the FUR. For instance, if the same identical Exit is sent to two physical devices or to two destinations, only one Exit is identified. But if the Exits to the two devices or destinations differ beyond the completely trivial (i.e. differences that cause some extra analysis or design), two Exits are identified.

#### Example 1

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

#### Example 2

As per Example 1. The output contains the same attributes as the screen display, but the printed layout is significantly different. 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, so measure 2 CFP.

---

<sup>11</sup> See the ‘Data Uniqueness’ rule in the Measurement Manual for v3.0 of the COSMIC Method; previously known as the ‘Data De-duplication’ rule in v2.2

### Example 3

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

### Example 4

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

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

*Functional process 1, showing details and/or list:*

E	Client name
R	Client data
X	Client data (one is found)
X	Client list data (more are found)
X	Error/confirmation messages

*Functional process 2, for selecting from the list + showing details:*

E	Client-id
R	Client data
X	Client data
X	Error/confirmation messages

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

#### 4.4.5 GUI elements

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

For GUI elements conveying data related to objects of interest, the rules for identification of functional processes and data movements apply. See especially section 4.1.8 'Drop-down lists arising in functional processes'.

#### 4.4.6 Authorization, help and log functionality

Authorization, help and log functionality may be measured if their functionality or changes to their functionality are explicitly described in the FUR, the functionality is in the application layer and it is agreed they should be within the scope of the measurement. It would be normal to ignore from the measurement scope existing authorization, help or log functionality that is exploited by the application being measured, but is not specific to it.

A Help button that is provided on each screen and which provides the same service from wherever it is pressed should be measured as one functional process for the whole application. (This assumes that the functional process type is the same for all screens of the application and that the context-dependent output simply represents different occurrences of the same output-type. Of course the Help function, once invoked, may offer other functional processes for more searching enquiries. In

this case each of these functional processes must be analyzed according to the usual rules, assuming the Help sub-system is within the measurement scope of the business application)

Depending on the FUR, help and log functionality may be required as separate functional processes for the application (in which case these functional processes are measured), or as part of some or all of the functional processes of the application. For example, logging may be required in the FUR as a separate function, or as an integral part of each functional process. In the latter case, identify one Write data movement per object of interest for which the log data are written.

#### *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'. All application error messages are considered to be different occurrences of a data movement called 'error message type'. So identify only one Exit for all application error messages in any one functional process. (See also the rule for Exits in the Measurement Manual).

The argument for this rule is that an error message results from a user requirement for some alternative output (one Exit), when a functional process fails to produce the normally expected output. Error messages result from a data movement that has failed (e.g. 'customer not present', or 'invalid code') and so there is only need to count one additional CFP for the error message in any one functional process. The error message is triggered by the data movement that failed. When, therefore for example, the error message results from the failure of a Read, no additional Read is necessary for generating the error message.

Following the same reasoning, 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 an occurrence of an error message, that is it should be regarded as accounted for by the single Exit 'error/confirmation messages'.

Ignore error messages arising from non-application software, e.g. 'server not responding'.

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. Example: in an order-entry functional process, the automatic production of a letter when an order is not accepted due to a credit-check failure. Identify 1 Exit.

## 4.5 Measurement of the size of functional changes to software

The reader is assumed to be familiar with the section 'Identifying objects of interest and data groups'; the section 'Identifying data attributes' and the section 'Measurement of the size of changes to software' of the Measurement Manual [3]. The approach of sizing changes to a functional process is illustrated by two examples.

### 4.5.1 Examples of functionally changed functional processes

#### Example 1

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

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

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

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

E	Employee data
W	Employee data
X	Error/confirmation messages

Total size 3 CFP

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

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

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

E	Employee data	data movement modified (attribute removed)
E	Dependent data	data movement added
W	Employee data	data movement modified (attribute removed)
W	Dependent data	data movement added
X	Error/confirmation messages	(no change) <sup>12</sup>

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

#### Example 2

A bank statement shows the interest payable each month on positive balances. The algorithm to calculate the interest must be modified in some detail although there is no functional change in the data needed as input to 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.

---

<sup>12</sup> There may be new or modified error/confirmation message occurrences, but the data movement type is unchanged.

#### 4.5.2 *Data conversion software*

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

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

#### 4.5.3 *Size of the functionally changed software*

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

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

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

## REFERENCES

- [1] The COSMIC Method v3.0: 'Documentation Overview and Glossary', 2007.
- [2] The COSMIC Method v3.0: 'Method Overview', 2007.
- [3] The COSMIC Method v3.0: 'Measurement Manual', 2007, The COSMIC Implementation Guide for ISO/IEC 19761:2003, COSMIC Measurement Practices Committee.
- [4] The COSMIC Method v3.0: 'Advanced and Related Topics', 2007.
- [5] ISO/IEC 19761:2003, Software Engineering – COSMIC – A functional size measurement method.
- [6] ISO/IEC TR 14143/5 'Information technology – Software measurement – Functional size measurement – Determination of Functional Domains for use with functional size measurement'.
- [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] Software Engineering, Sommerville, I., Addison-Wesley, 1996.
- [13] ISO/IEC 14143/1:2007 'Information technology – Software measurement – Functional size measurement – Definition of concepts'.

## APPENDIX A - COSMIC CHANGE REQUEST AND COMMENT PROCEDURE

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

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

mpc-chair@cosmicon.com

### Informal general feedback and comments

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

### Formal change requests

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

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

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

A form for submitting a CR is available from the [www.cosmicon.com](http://www.cosmicon.com) site.

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

### Questions on the application of the COSMIC method

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

## APPENDIX B - MAIN CHANGES IN V1.1 FROM V1.0 OF THE BUSINESS APPLICATION GUIDELINE

Note. The nature of a change is indicated by

- 'Method' when the *content* of the COSMIC method was changed, or by
- 'Editorial' when the *description* of the method (but not the method itself) was changed.

References in version 1.0 / 1.1	Nature of change	Comment
-	Editorial	The Guideline has been re-structured so that all the general background material is dealt with first, followed by material in the sequence of the phases of the Measurement Manual
-	Method	References to 'viewpoint' as an abstraction principle and to the End User Measurement Viewpoint have been removed where appropriate
-	Method	'End user' replaced by 'functional user' where appropriate.
-	Editorial	References to the Method Update Bulletins 1 and 2 have been removed as they have been incorporated into the COSMIC method v3.0 documents
Foreword	Editorial	Characterization details of the 'business application software domain' have been moved from the Foreword to become the first section of chapter 2.
4.1 / 1.2	Editorial	The definition of 'Functional User Requirements' has been updated to that of ISO/IEC 14143/1:2003 [13]
5.4.5 / 4.1.5	Method	The section 4.1.5 'Separate functional processes' has been renamed as 'Separate functional processes owing to separate functional user decisions'
5.4.6 / 4.1.6	Editorial	The section 'Functional process and same data' has been renamed to 'Retrieve and update of data in a single functional process. The former name does not clarify what the issue is, nor that it is a sequel to the section before
5.6.3 / 4.4.3	Editorial	The text on batch processing has been brought in line with the text in the Measurement Manual. The cases and examples are now more comprehensive
5.6.4 / 4.4.4	Editorial	The text on multiple sources etc. has been extended, brought in line with the text on the 'data uniqueness rule' in the Measurement Manual and an example added.
5.7 / 4.5	Editorial	The definition of a 'change', the criteria for distinguishing a modification to a data movement and the rules for identifying and counting modifications to a data movement have all been removed as these are described in detail in the Measurement Manual v3.0

## **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 'The COSMIC Method v3.0: Documentation Overview and Glossary' (see References [1]).

### **E/RA**

Abbreviation for 'Entity Relationship Analysis' – see section 2.3 of this Guideline.

### **Entity-type**

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

### **RDA**

Abbreviation for 'Relational Data Analysis' – see section 2.5 of this Guideline

### **Relation**

Any set of data attributes.

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

### **SOR**

Abbreviation for 'Statement of requirements'.

### **Statement of requirements**

A document containing all the requirements for a piece of software, that is, Functional User Requirements and Non-Functional Requirements, (i.e. technical requirements and quality requirements).

### **UML**

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