COSMIC Measurement Manual for ISO 19761

Part 3a: Examples

Version 5.0

March 2020

Editing February 2021

Foreword.

The COSMIC Measurement Manual for ISO/IEC 19761:2011 consists of the Parts:

Part 1: Principles, definitions & rules.

Part 2: Guidelines.

Part 3: Examples of COSMIC concepts and measurements, consisting of:

Part 3a	Examples
Part 3b	Real-time Examples
Part 3c	MIS Examples (under construction).

This Part 3a Examples of the COSMIC Measurement Manual provides examples of measurements and concepts of the COSMIC Functional Size Measurement method (the 'COSMIC Method'). The examples are presented per measurement phase and where needed grouped per topic. The topics are visible in the Table of Contents for ease of search. This Part does not provide examples of the real-time domain - see Part 3b Real-time Examples.

A public domain version of the COSMIC Measurement Manual and other technical reports, including translations into other languages, can be found at the Knowledge base of www.cosmic-sizing.org.

February 2021 editing of Part 3, to become Part 3a:

- All Real-time examples moved to Part 3b.
- The short introduction to each section removed, as it repeats the text of Parts 1 and 2.
- In section 3.1 'Functional processes' example added on Identifying functional processes of installed software.
- In section 3.2, 'Data groups and objects of interest', Example 2 removed.
- In section 3.6 'Control commands' the clause 'Functions that enable a user to control the display (or not) of a header or of sub-totals' extended with 'or to sort a column with values in ascending or descending order'.

Editors:

Alain Abran, Ecole de technologie supérieure – University of Quebec (Canada),
Peter Fagg, Pentad (UK),
Arlan Lestherhuis (The Netherlands).

Other members of COSMIC Measurement Practices Committee:

Jean-Marc Desharnais, Ecole de technologie supérieure – University of Quebec (Canada),
Dylan Ren, Measures Technology LLC (China),
Bruce Reynolds, Tecolote Research (USA),
Hassan Soubra, German University in Cairo (Egypt),
Sylvie Trudel, Université du Québec à Montréal - UQAM (Canada),
Frank Vogelezang, Metri (The Netherlands).

Table of Contents.

1 INTRODUCTION TO COSMIC. 5
1.1 Functional User Requirements. 5
1.2 Non-Functional Requirements (NFR). 5
1.3 The COSMIC Software Context Model. 5
1.4 The Generic Software Model. 5
1.5 Types versus occurrences. 5
1.5.1 The Software Context Model. 5
1.5.2 The Generic Software Model. 5
1.6 The COSMIC measurement process. 5

2 THE MEASUREMENT STRATEGY PHASE. 5
2.1 Purpose of a measurement. 5
2.2 Scope of a measurement. 6
2.3 Functional users. 7
2.3.1 Common examples. 7
2.3.2 Incompatibility of functional users. 7
2.4 Measurement Strategy Patterns. 7
2.5 Layers. 8
2.5.1 A layered business applications software architecture. 8
2.5.2 Layers and the 'view' of the architecture. 8
2.6 Levels of decomposition. 9
2.7 Context diagrams. 9
2.8 Levels of granularity. 9

3 THE MAPPING PHASE. 12
3.1 Functional processes. 12
3.1.1 Starting a functional process. 12
3.1.2 Separate decisions, separate functionality. 12
3.1.3 Separate responsibilities. 13
3.1.4 Batch processing as on-line processing. 13
3.1.5 Organization of data input. 13
3.1.6 Functional processes of installed software. 14
3.1.7 Various processing paths. 14
3.2 Data groups and objects of interest. 14
3.2.1 Data groups. 14
3.2.2 Objects of interest. 14
3.2.3 The functional user as object of interest. 15
3.3 Data attributes. 15
3.4 Data movements. 15
3.4.1 Data movement(s) and obtaining date and/or time. 15
3.4.2 Data that are not related to an object of interest. 15
3.4.3 Data movements, the common case. 15
3.4.4 Different functional users, different data groups, same object of interest. 15
3.4.5 Persistent storage, different data groups, same object of interest. 15
3.4.6 Repeated occurrences, same object of interest. 16

3.4.7 Data movement when an enquiry outputs fixed text. 16
3.4.8 Exchanging data between pieces of software. 16
3.4.9 Data movement(s) and different rights of access to stored data. 17
3.5 Data manipulations associated with data movements. 17
3.6 Control commands. 18
3.7 Error/confirmation messages and other indications of error conditions. 18
3.7.1 General examples 18
3.7.2 Error/confirmation messages with additional data. 18
3.7.3 Error conditions to human users which are not specified in the FUR. 19
3.8 Measuring the components of a distributed software system. 19
3.9 Re-use of software. 19
3.9.1 Functional processes that include common functionality. 19
3.9.2 Functionality that is not in the FUR. 19
3.10 Measurement of the size of changes to software. 19
3.10.1 Changes and their sizes. 19
3.10.2 Sizing a deletion of a part of an application. 20
3.10.3 Changing an error/confirmation message 20
3.10.4 Changing control commands and application-general data. 20
3.11 Extending the COSMIC measurement method. 20

1 INTRODUCTION TO COSMIC.

1.1 Functional User Requirements.

A large number of examples of Functional User Requirements (FUR) are documented in the COSMIC Case studies (freely available at cosmic-sizing.org).

1.2 Non-Functional Requirements (NFR).

BUSINESS EXAMPLE: The requirements for a new software system include the statement 'the user needs the option to secure files by encryption'. The project to develop the system is at the stage of estimating effort and cost. Two options are considered:

- Develop some proprietary encryption software. For project estimating purposes it may be necessary to measure the size of the FUR for the encryption software.
- Purchase an existing Commercial Off-The-Shelf (COTS) package. For project estimating purposes it may be necessary to measure only the size of the software functionality needed to integrate the COTS package. The cost of the package and the effort to integrate and test the file encryption package will also need to be considered in the project cost estimate.

1.3 The COSMIC Software Context Model.

See the Case studies at cosmic-sizing.org.

1.4 The Generic Software Model.

See the Case studies at cosmic-sizing.org.

1.5 Types versus occurrences.

1.5.1 The Software Context Model.

BUSINESS EXAMPLE: The system supporting a Call Centre has 100 employees who answer customer questions. As the employees are subject to the same requirement ('answer customer questions') a context model of the system includes the one functional user type: 'Call Centre employee' of which there are 100 occurrences.

1.5.2 The Generic Software Model.

BUSINESS EXAMPLE: Suppose a functional process that enables data to be entered and validated for a new customer. The Entry data movement will be executed, i.e. it will occur once, each time a human functional user registers data for a new customer. During its execution, the functional process must validate the entered data by searching to check if the customer already exists in the database. Hence the Read data movement for this validation will occur one or more times (depending on the database design). However, one Entry and one Read of the customer are counted when measuring this functional process.

1.6 The COSMIC measurement process.

See the Case studies at cosmic-sizing.org.

2 THE MEASUREMENT STRATEGY PHASE.

2.1 Purpose of a measurement.

EXAMPLE: The following are typical measurement purposes.

- To measure the size of the FUR as they evolve, as input to a process to estimate development effort.
- To measure the size of changes to the FUR after they have been initially agreed, in order to manage project 'scope creep', caused by addition of and changing requirements.
- To measure the size of the delivered software as input to the measurement of the development organization's performance.
- To measure the size of the total software delivered, and also the size of the software which was developed, in order to obtain a measure of functional re-use.
- To measure the size of the existing software as input to the measurement of the performance of the group responsible for maintaining and supporting the software.
- To measure the size of some changes to an existing software system as a measure of the size of the work-output of an enhancement project team.
- To measure the size of the sub-set of the total software functionality that must be developed, that will be provided to the software's human functional users.

2.2 Scope of a measurement.

BUSINESS EXAMPLE: Figure 2.1 shows all the separate pieces of software – the 'overall scope' - delivered by a project team:

- the client and the server components of an implemented application software package
- a program that provides an interface between the server component of the new package and existing applications
- a program that is used once to convert existing data to the new format required by the package. This program was built using a number of re-usable objects developed by the project team
- the device driver software for new hardware on which the package client component will execute

Each individual piece of software for which a measurement scope was defined is shown as a solid rectangular box.

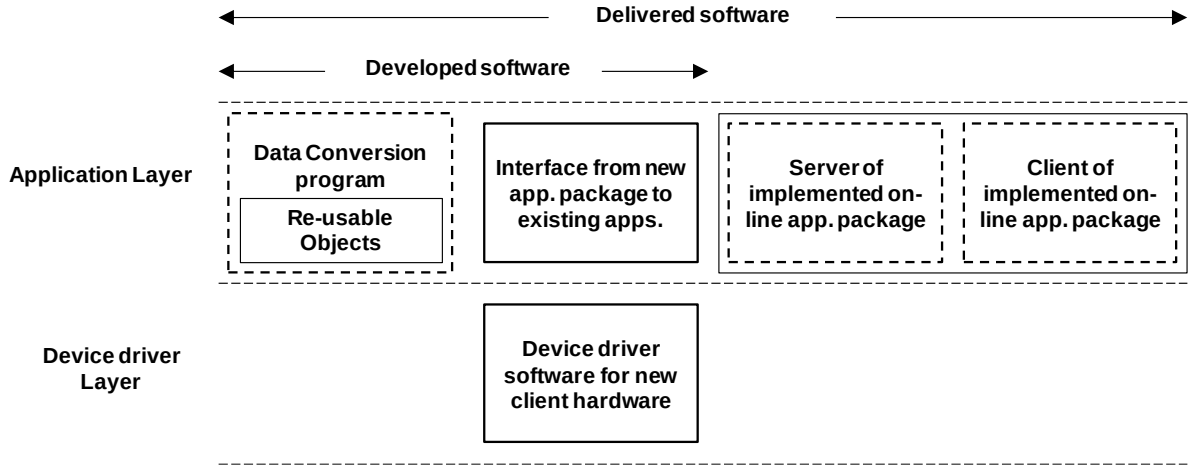

Figure 2.1 - Overall scope and the individual measurement scopes.

The diagram shows that the 'delivered' pieces of software consisted of some that were newly developed and some that were implemented by the project team. The purpose is to measure

the FUR of the individual pieces of delivered software to be added to the size of the organization's software portfolio, considering the software package as a whole, i.e. ignoring the client-server component structure.

The size of the implemented package was added with that of the interface program to update the total size of the organization's application portfolio. The size of the data conversion program was not of interest as it was used once and thrown away. But the size of each of the re-usable objects was recorded in the organization's infrastructure software inventory, as well as that of the new device driver. Again these were classified separately.

Due to the diverse nature of the deliverables it would not be sensible when measuring the overall project team's performance to add together the sizes of all the delivered software. The performance of the teams that delivered each piece of software should be measured separately.

Note also that it is normally only of interest to measure the software resulting from implementing a package, not the package itself. The latter is perhaps only of interest to the package supplier.

2.3 Functional users.

2.3.1 Common examples.

BUSINESS EXAMPLE 1: In an order system a number of employees (human functional users) maintain the order data. An employee's ID is added to all data groups they enter. Identify one 'employee' functional user type because the order system FUR are the same for all these employees.

BUSINESS EXAMPLE 2: A software system has functionality to maintain basic personal data accessible to all staff of the Personnel Department, and more sensitive salary data accessible to only a sub-set of these staff. Therefore, this software has two types of functional users. The purpose of a measurement of this software should define whether the measurement scope applies to the functionality accessible to all staff or is restricted to the functionality available to one of the two types of staff.

2.3.2 Incompatibility of functional users.

EXAMPLE: Consider the embedded software of a copier. The software's functional users could be defined in one of two ways. They could be either (a) the human user who wants to make copies, or (b) the copier's hardware devices i.e. the control buttons, a screen on which messages are displayed to the human user, the paper transport mechanism, the paper jam sensors, the ink controller, indicator lights, etc., with which the software interacts directly. These two types of functional users, humans or the set of hardware devices, will 'see' different functionality. The human user, for example, will be aware of only a sub-set of the total copier software functionality. The developers of the embedded software that drives the copier will need to define the hardware devices as its functional users. Alternatively, a marketing person may find it useful to measure a size of the functionality of his own company's copier as seen by a human functional user versus that of a competitor's product in order to compare their price/performance[1]. Do NOT try to mix the two views; a size measurement from a 'mixed' human/hardware view would be very difficult to interpret.

2.4 Measurement Strategy Patterns.

See the examples in the Guideline for Measurement Strategy Patterns.

[1] Toivonen, for example, compared the size of the functionality of mobile phones available only to human users in 'Defining measures for memory efficiency of the software in mobile terminals', International Workshop on Software Measurement, Magdeburg, Germany, October 2002.

2.5 Layers.

2.5.1 A layered business applications software architecture

BUSINESS EXAMPLE 1: The physical structure of a typical layered software architecture supporting business applications software is given in Figure 2.2:

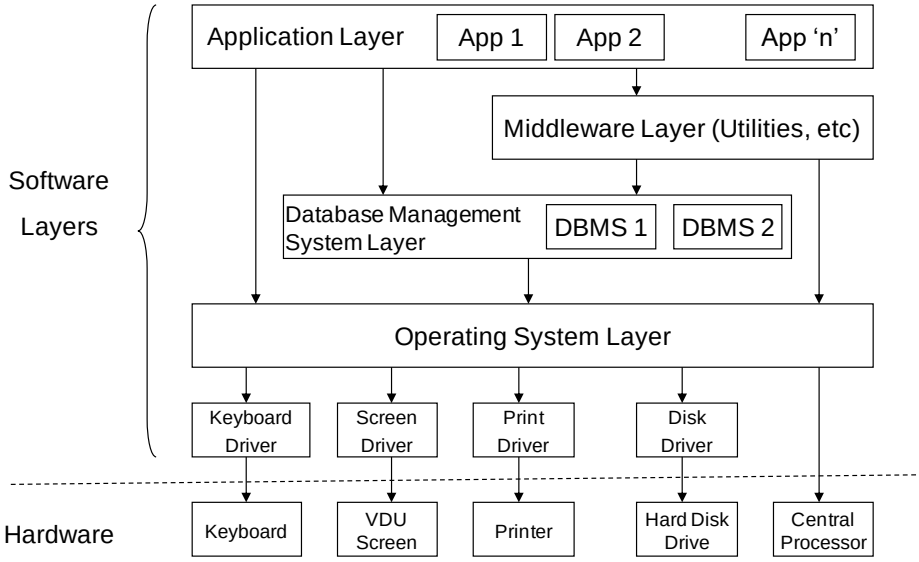

Figure 2.2- Typical layered software architecture for a Business/MIS system.

EXAMPLE 1: The pieces of software in the application layer of Figure 2.2 are all peers of each other.

EXAMPLE 2: Normally in software architectures, the 'top' layer, i.e. the layer that is not a subordinate to any other layer in a hierarchy of layers, is referred to as the 'application' layer. Software in this application layer relies on the services of software in all the other layers for it to perform properly. Software in this 'top' layer may itself be layered, e.g. as in a 'three-layer architecture' of User Interface, Business Rules and Data Services components (see the example in section 2.5.2).

2.5.2 Layers and the 'view' of the architecture.

BUSINESS EXAMPLE: Consider an application A situated in a layered software architecture, as in Figure 2.3 below, which shows three possible layer structures a), b) and c) according to different architecture 'views'.

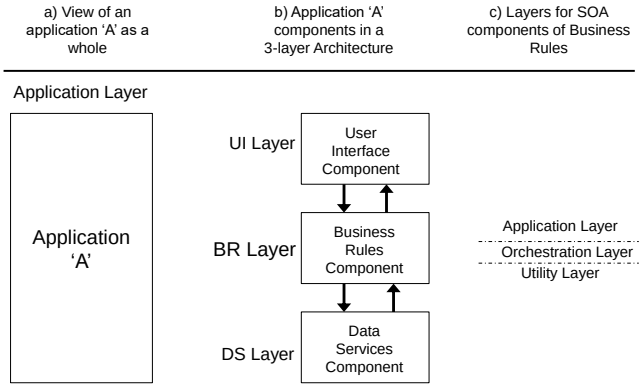

Figure 2.3 - Three views of the layers of an application.

Purpose 1 is to measure the functional size of application A ‘as a whole’, as in View a). The measurement scope is the whole of application A, which exists entirely within the one ‘application’ layer

Purpose 2. Application A has been built according to a ‘three-layer’ architecture comprising a User Interface, Business Rules and Data Services components. Purpose 2 is to measure the three components separately as in View b). Each component is situated in its own layer of the architecture and the measurement scope must be defined separately for each component.

Purpose 3. The Business Rules component of the application has been built using re-usable components of a Service-Oriented Architecture, which has its own layer structure. Purpose 3 is to measure an SOA component of the Business Rules component as in View c). Each SOA component is situated in a layer of the SOA architecture and the measurement scope must be defined separately for each SOA component. (Note that SOA terminology also uses ‘application layer’ within its own architecture.)

2.6 Levels of decomposition.

EXAMPLE: The Guideline for 'Measurement Strategy Patterns' recognizes three standard levels of decomposition: ‘Whole application', 'Major Component' and 'Minor component'. The three levels are shown in Figure 2.3.

2.7 Context diagrams.

BUSINESS EXAMPLE: Figure 2.4 shows the context diagram for the client/server software of the implemented application package as in the example shown in Figure 2.1, to be measured as a ‘whole’, i.e. the fact that the application package has two components (client and server) is to be ignored for this measurement of the package.

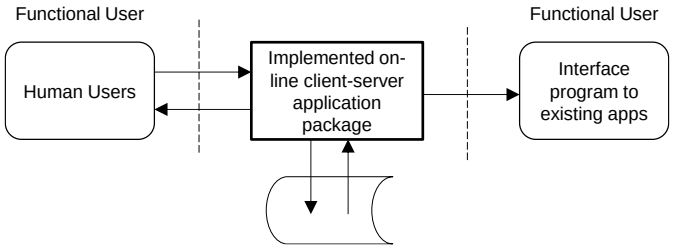

Figure 2.4 - Context diagram for the client-server application.

2.8 Levels of granularity.

EXAMPLE: A group of functional users might be a ‘department’ whose members handle many types of functional processes, or a ‘control panel’ that has many types of instruments, or ‘central systems’.

A group of events might be indicated in a statement of functional requirements at a high level of granularity by an input stream to an accounting software system labelled ‘sales transactions’ or by an input stream to an avionics software system labelled ‘pilot commands’.

EXAMPLE: For an example of sizing at varying levels of granularity and of decomposition, see the telecoms system example in the *Early Software Sizing with COSMIC: Experts Guidelines*.

BUSINESS EXAMPLE: The example, from the domain of business application software, is part of a well-known system for ordering goods over the Internet, which we will call the ‘Everest Ordering Application'. The purpose of this example is to illustrate different levels of

granularity and that functional processes may be revealed at different levels'. The description below is highly-simplified for the purposes of this illustration of levels of granularity.

If we wished to measure this application, we might assume the purpose of the measurement is to determine the functional size of the part of the application available to the human customer users (as 'functional users'). We would then define the scope of the measurement as 'the parts of the Everest application accessible to customers for ordering goods over the Internet'. Note, however, that the purpose of this example is to illustrate different levels of granularity. We will therefore explore only some parts of the system's total functionality sufficient to understand this concept of levels of granularity. This example is about levels of granularity of the FUR; it says nothing about any possible decomposition of the underlying software.

At the highest 'Level 1 (Main Function)' of this part of the application a statement of the requirements of the Everest Ordering Application would be a simple summary statement such as the following.

'The Everest Ordering Application must enable customers to enquire upon, select, order, pay for and obtain delivery of any item of Everest's product range, including products available from third party suppliers.'

Zooming-in on this highest-level statement of the requirements we find that at the next lower level 2 the Everest Ordering Application consists of four sub-functions, as shown on Figure 2.5 (a).

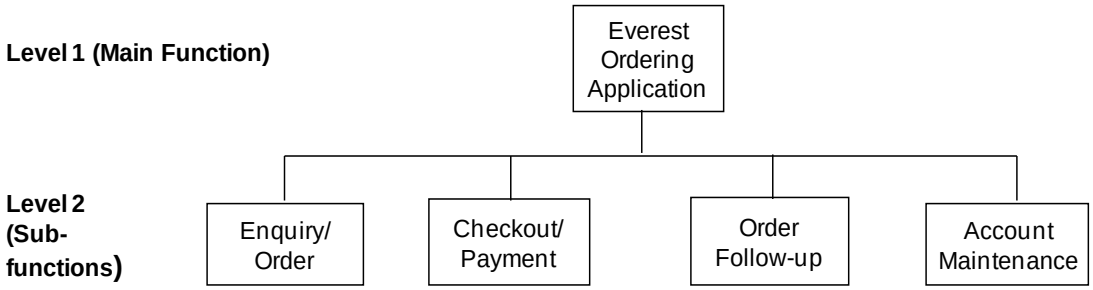

Figure 2.5 (a) - Analysis of the Everest Ordering System: the first two levels of granularity.

The requirements of the four sub-functions are:

- The Enquiry/Order sub-function which enables a customer to find any product in the Everest database, as well as its price and availability and to add any selected product to a 'basket' for purchase. This sub-function also promotes sales by suggesting special offers, offering reviews of selected items and enabling general enquiries such as on delivery terms, etc. It is a very complex sub-function. We therefore do not analyze this sub-function in any further detail below level 2 for the purposes of this example.
- The Checkout/Payment sub-function which enables a customer to commit to order and pay for the goods in the basket.
- The Order Follow-up sub-function that enables a customer to enquire how far an existing order has progressed in the delivery process, to maintain their order (e.g. change delivery address) and to return unsatisfactory goods.
- The Account Maintenance sub-function that enables an existing customer to maintain various details of his/her account such as home address, means of payment, etc.

Figures 2.5 (b) and (c) show some details revealed when we zoom-in on the requirements, down one further level of granularity on the Checkout/Payment sub-function, the Order Follow-up sub-function and the Account Maintenance sub-function. In this zooming-in process it is important to note that:

- we have not changed the scope of the functionality to be measured, and
- all levels of the description of the Everest application show the functionality available to the customers (as functional users). A customer can 'see' the functionality of the application at all these levels of granularity.

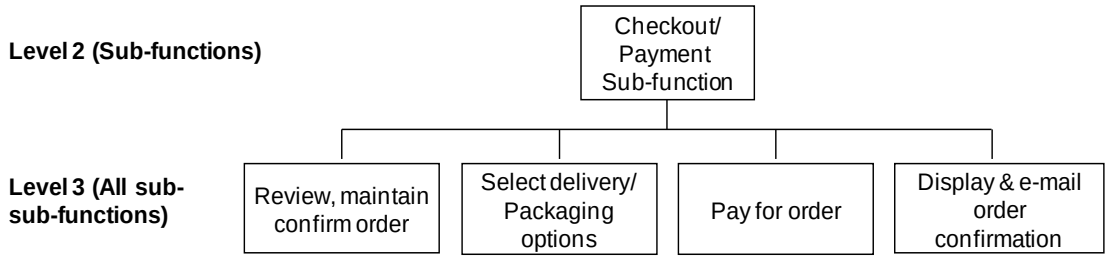

Figure 2.5 (b) - Analysis of the Checkout/Payment Sub-function.

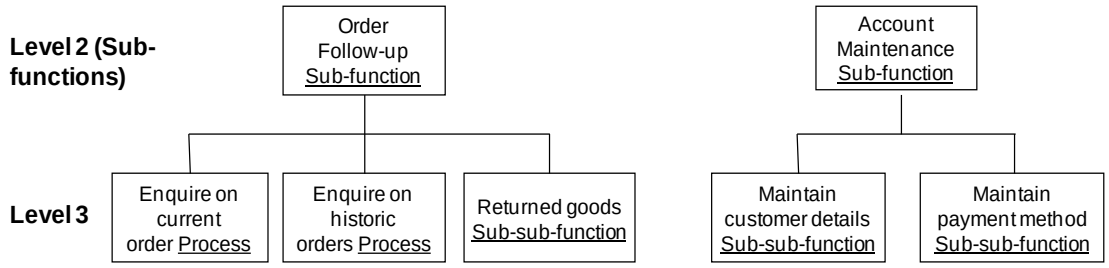

Figure 2.5 (c) - Analysis of the Order Follow-up and Account Maintenance Sub-function.

Figure 2.5 (c) now reveals that when we zoom-in to the lower level 3 of this particular analysis of the Order Follow-up sub-function, we find two individual functional processes[2] at level 3 (for two enquiries of the Order follow-up sub-function). More functional processes would be revealed if we were to continue the refinement of the Level 3 sub-sub-functions to lower levels. This example demonstrates, therefore, that when some functionality is refined in a 'top-down' approach, it cannot be assumed that the functionality shown at a particular 'level' on a diagram will always correspond to the same 'level of granularity' as this concept is defined in the COSMIC method. (This definition requires that at any one level of granularity the functionality is 'at a comparable level of detail'.)

Furthermore, other analysts might well draw the diagrams differently, showing other groupings of functionality at each level of the diagram. There is not one 'correct' way of zooming in on the functionality of such a complex system.

Given these variations that inevitably occur in practice, a Measurer must carefully examine the various levels of an analysis diagram to find the functional processes that must be measured. Where in practice this is not possible, for example because the analysis has not yet reached the level where all functional processes have been revealed, an approximate method must be applied. To illustrate this, let us examine the case of the 'Maintain customer details sub-sub-function' (see Figure 2.5 (c) above), in the branch of the Account Maintenance sub-function.

To an experienced Measurer, the word 'maintain' almost invariably suggests a group of events and thus a group of functional processes. We can therefore assume that this 'Maintain' sub-sub-function must comprise three functional processes, namely an 'enquire on customer details', 'update customer details' and 'delete customer details'. (The 'create customer details' process must also obviously exist, but this occurs in another branch of the

system, when a customer orders goods for the first time. It is outside the scope of this simplified example.)

An experienced Measurer should be able to 'guestimate' a size of this sub-sub-function in units of COSMIC Function Points by taking the assumed number of functional processes (three in this case) and multiplying this number by the average size of a functional process. This average size would be obtained by calibration in other parts of this system or in other comparable systems. Examples of this calibration process are given in the document 'Early Software Sizing with COSMIC: Experts Guide' which also contains other examples of other approaches to approximate sizing.

Clearly, such approximation methods have their limitations. If we apply such an approach to the Level 1 statement of requirements as given above ('The Everest application must enable customers to enquire upon, select, order, pay for and obtain delivery of any item of Everest's product range ….'), we could identify a few functional processes. But more detailed analysis would reveal that the real number of functional processes in this complex application must be much greater. That is why functional sizes usually appear to increase as more details of the requirements are established, even without changes in scope. These approximation methods must therefore be used with great care at high levels of granularity, when very little detail is available.

3 THE MAPPING PHASE.

3.1 Functional processes.

3.1.1 Starting a functional process.

BUSINESS EXAMPLE 1: In a company, an order is received (triggering event), causing an employee (functional user) to enter the order data (triggering Entry conveying data about the object of interest 'order'), as the first data movement of the 'order entry' functional process.

BUSINESS EXAMPLE 2: Suppose that on receipt of an order in Business Example 1, the order-processing application is required to send the details of any new client to a central client-registration application, which is being measured. The order-processing application is thus a functional user of the central application. The order-processing application, having received data about a new client, generates the client data group which it sends to the central client-registration application which triggers a functional process to store these data.

BUSINESS EXAMPLE 3: A functional process of a personnel software system may be started by the triggering Entry that moves a data group describing a new employee. The data group is generated by a human functional user of the personnel software who enters the data.

3.1.2 Separate decisions, separate functionality.

BUSINESS EXAMPLE: A functional user enters a customer order for an item of complex industrial equipment and later confirms acceptance of the order to the customer. Between entering the order and accepting it, the user may make enquiries about whether the new order can be delivered by the requested delivery date, and about the customer's credit-worthiness, etc. Although acceptance of an order must follow entry of an order, in this case the user must make a separate decision to accept the order. This indicates separate functional processes for order entry and for order acceptance (and for each of the enquiries).

3.1.3 Separate responsibilities.

BUSINESS EXAMPLE: In a personnel system where the responsibility for maintaining basic personal data is separated from the responsibility for maintaining payroll data indicating separate functional users each with their own separate functional processes.

3.1.4 Batch processing as on-line processing.

BUSINESS EXAMPLE 1: Suppose orders are entered by an 'off-line' process in some way, e.g. by optical scanning of paper documents, and are stored temporarily for automatic batch processing. How is this order-entry functional process analyzed if it is to be batch-processed? The functional user is the human who causes the order data to be entered off-line ready to be processed in batch mode; the triggering Entry of the functional process that will process the orders in batch mode is the data movement that moves the order data group into the process. The off-line process, if it must be measured, involves another, separate functional process that loads the orders to temporary storage.

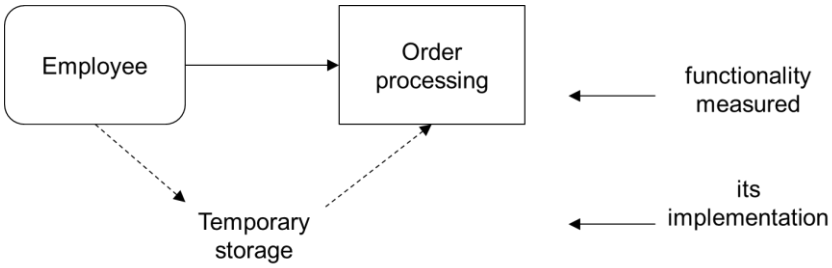

Figure 3.1 – Batch processing of off-line entered orders

BUSINESS EXAMPLE 2: Suppose FUR for an end-of year batch-processed application is to report the outcome of business for the year, and to reset positions for the start of the next year. Physically, an end-of-year clock tick generated by the operating system causes the application to start processing. Logically, however, each functional process of the application takes its input data from the stream of data to be batch-processed. This should be analyzed in the normal way (e.g. the input data for any one functional process comprises one or more Entries, the first of which is the triggering Entry for that process).

However, assume there is a particular functional process of the batch-processed application that does not require any input data to produce its set of reports. Physically, the (human) functional user has delegated to the operating system the task of triggering this functional process. Since every functional process must have a triggering Entry, we may consider the end-of-year clock tick that started the batch stream as filling this role for this process. This functional process may then need several Reads and many Exits to produce its reports. Logically, the analysis of this example is no different if the human functional user initiates the production of one or more reports via a mouse click on an on-line menu item, rather than delegating the triggering of the report production in batch mode to the operating system.

3.1.5 Organization of data input.

BUSINESS EXAMPLE: Suppose there is a functional user requirement for two types of social benefits, first for an additional child and second a 'working tax credit' for those on low income. These are requirements for the software to respond to two events that are separate in the world of the human functional users. Hence there should be two functional processes, even though a single tax form may have been used to capture data for both cases.

3.1.6 Functional processes of installed software.

BUSINESS EXAMPLE: If functional processes of installed software must be identified, examine its menus. Menu choices that are associated with triggering events are functional processes. Note that there may be batch processing functional processes as well.

3.1.7 Various processing paths.

BUSINESS EXAMPLE 1: A functional process that provides a general search capability against a database may be required to accept up to four search parameters (attributes of its triggering Entry). But the same functional process will function if the values of only one, two or three search parameters are entered.

BUSINESS EXAMPLE 2: For a functional process to register a new customer for a car rental company, it is mandatory to enter data for most data attributes, but some (e.g. some contact details) are optional and may be left blank. Regardless of whether all or a sub-set of these attributes are entered there is only one functional process for registering a new customer.

BUSINESS EXAMPLE 3: Continuing from Business Example 2, for the functional process to make a car rental reservation in the same company, there are several options which may or may not be taken up, e.g. for extra insurance, additional drivers, requests for child seats, etc. These different options lead to different processing paths within the car rental reservation functional process, but there is still only one functional process for reserving a car rental.

3.2 Data groups and objects of interest.

3.2.1 Data groups.

EXAMPLE: In practice, a data group can have many origins, e.g.:

a) A data structure on a hardware storage device (file, database table, ROM memory, etc.).

b) A data structure within the computer's volatile memory (data structure allocated dynamically or through a pre-allocated block of memory space).

c) A clustered presentation of functionally-related data attributes on an input/output device (display screen, printed report, control panel display, etc.).

d) A message in transmission between a device and a computer, or over a network, etc.

3.2.2 Objects of interest.

BUSINESS EXAMPLE 1: In the domain of business application software, an object of interest could be 'employee' (physical) or 'order' (conceptual). In the case of 'order', it commonly follows from the FUR of multi-line orders that two objects of interest are identified: 'order' and 'order-line'. The corresponding data groups could be named 'order data', and 'order-line data'.

BUSINESS EXAMPLE 2: A common data structure represents objects of interest that are mentioned in the FUR, which can be maintained by functional processes, and which is accessible to most of the functional processes found in the measured software.

BUSINESS EXAMPLE 3: Assume FUR for an ad hoc enquiry functional process against a personnel database to find out the number of employees older than a given age, which must be input. The input parameter (the age limit) is a data group that defines an object of interest 'the set of employees aged over the given limit'. The output data group, comprising the count of employees aged over the given limit describes the same object of interest as the input.

BUSINESS EXAMPLE 4: Assume the same ad hoc enquiry against a personnel database as in Business Example 3, but in addition to outputting the total number of employees over

the given age limit, the enquiry must also list the names of all employees aged over the given limit. The analysis is as for Business Example 3 but the functional process must now output two data groups: the count of employees and the list of names of employees over the given age limit (derived from persistent data). These must be two separate data groups because they have different frequencies of occurrence (one count of employees for zero, one or more employee names.

3.2.3 The functional user as object of interest.

BUSINESS EXAMPLE 1: A human functional user enters an ID and a password in a logon process to identify himself/herself to a system. The object of interest of the data group entered is the human user.

3.3 Data attributes.

BUSINESS EXAMPLE: An 'employee' object of interest may be described by a data group called 'Employee master data', which contains the data attributes 'Employee ID', 'Name', 'Address', 'Date of birth', 'Sex', 'Marital status', 'National Insurance number', 'Grade', 'Job title', etc.

3.4 Data movements.

3.4.1 Data movement(s) and obtaining date and/or time.

BUSINESS EXAMPLE: When a functional process adds a time stamp to a record to be made persistent or to be output no Entry is identified. By convention, obtaining the system's clock value is functionality that is made available by the operating system to all functional processes.

3.4.2 Data that are not related to an object of interest.

BUSINESS EXAMPLE 1: Do not identify data movements for moving application-general data such as headers and footers (company name, application name, system date, etc.) appearing on all screens.

BUSINESS EXAMPLE 2: Do not identify data movements for moving control commands (a concept defined only in the business application domain) that enables a functional user to control their use of the software, rather than to move data, e.g. page up/down commands, clicking 'OK' to acknowledge an error message, etc., see further section 3.6.

3.4.3 Data movements, the common case.

BUSINESS EXAMPLE: The common case is that one Write would be identified that moves a data group containing <u>all</u> the data attributes of an object of interest required to be made persistent in a given functional process.

3.4.4 Different functional users, different data groups, same object of interest.

BUSINESS EXAMPLE: Suppose FUR exist for a single functional process to produce two or more Exits moving different data groups describing the same object of interest, intended for different functional users: e.g., when a new employee joins a company a report is produced to be passed to the employee to sign off his personal data as valid, and a message is sent to Security to authorize the employee to enter the building. Identify two Exits.

3.4.5 Persistent storage, different data groups, same object of interest.

BUSINESS EXAMPLE 1: Suppose FUR for a single functional process A to store two data groups derived from a bank's current account files for later use by separate programs.

The first data group is ‘overdrawn account’ details’ (which includes the negative balance attribute). The second data group is ‘high value account’ details’ (which only has the account holder’s name and address, intended for a marketing mail-shot). Functional process A will have two Writes, one for each data group, both describing the same object of interest ‘account’.

BUSINESS EXAMPLE 2: Suppose FUR for a program to merge two files of persistent data describing the same object of interest, e.g. a file of existing data about an object of interest ‘X’ and a file with newly-defined attributes describing the same object of interest ‘X’. Identify two Reads, one for each file, for the functional process.

3.4.6 Repeated occurrences, same object of interest.

BUSINESS EXAMPLE 1: Suppose a Read of a data group is required in the FUR, but the developer decides to implement it by two commands to retrieve different sub-sets of data attributes of the same object of interest from persistent storage at different points in the functional process. Identify one Read.

BUSINESS EXAMPLE 2: Suppose a Read is required in the FUR that in practice requires many retrieval occurrences, as in a search through a file. Identify one Read.

3.4.7 Data movement when an enquiry outputs fixed text.

BUSINESS EXAMPLE: For the result of pressing a button for ‘Terms & Conditions’ (e.g. on a shopping web-site) identify one Exit for the fixed text output.

3.4.8 Exchanging data between pieces of software.

EXAMPLE: The pieces of software to be measured are assumed to have a ‘client/server’ relationship, i.e. where one piece, the client, obtains services and/or data from the other piece, the ‘server’, in the same or a different layer. Figure 3.2 shows an example of such a relationship, in which the two pieces are major components of the same application. In any such client/server relationship, the FUR of the client component C1 would identify the server component C2 as one of its functional users, and vice versa. The same relationship would exist and the same diagram would apply if the two pieces were separate applications, or if one of the pieces were a component of a separate application.

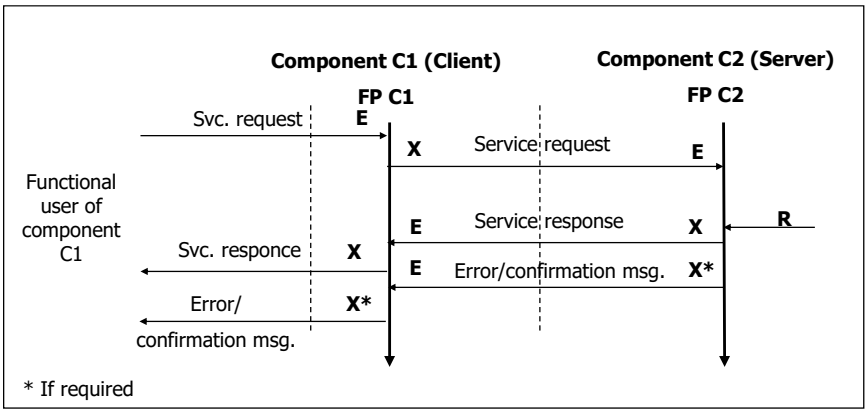

Figure 3.2 – Data exchanges between client and server components.

Physically, the two components could execute on separate processors; in such a case they would exchange data via the respective operating systems and any other intermediate layers of their processors in a software architecture such as shown in Figure 2.1. But logically,

applying the COSMIC models, the two components exchange data via an Exit followed by an Entry data movement. All intervening software and hardware is ignored in this model.

Figure 3.2 shows that a functional process FP C1 of the client component C1 is triggered by an Entry from a functional user (such as a human) which consists, for example, of the parameters of the enquiry. The FUR of component C1 will recognize that this component must ask the server component C2 for the required data, and must tell it what data group is required.

To obtain the required data group, FP C1 issues an Exit containing the enquiry request parameters to component C2. This Exit data movement crosses the boundary between C1 and C2 and so becomes the triggering Entry of a functional process FP C2 in the component C2. The functional process FP C2 of component C2 is assumed to obtain the required data group via a Read from its own persistent storage, and sends the data back to C1 via an Exit. Functional process FP C1 of component C1 receives this data as an Entry. FP C1 then passes the data group on as an Exit to satisfy the enquiry of its functional user.

Taking into account the possible error/confirmation message issued by the client, this enquiry therefore requires 6 data movements (i.e. 6 CFP) to satisfy the enquiry request for component C1 and 4 CFP for component C2.

3.4.9 Data movement(s) and different rights of access to stored data.

EXAMPLE: See Figure 3.3. Suppose a piece of software A to be measured is allowed to retrieve certain stored data Z, but it is not allowed to maintain (i.e. create, update or delete) this same data Z directly. The piece of software B is required to ensure the integrity of the data Z by ensuring consistent validation, so it processes all data maintenance requests for the data Z. When A is required to maintain the data Z, A must pass its request to B via an Exit followed by an Entry.

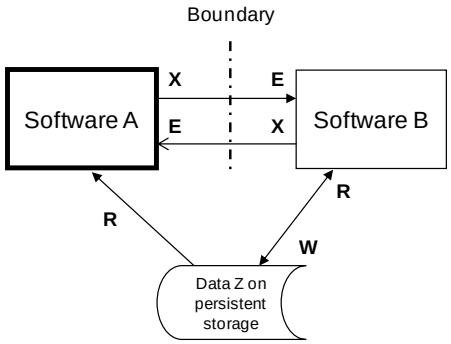

Figure 3.3 – Persistent data Z within the boundary of both software A and B for a Read.

3.5 Data manipulations associated with data movements.

BUSINESS EXAMPLE 1: An Entry includes all manipulation needed to format a screen to enable a human user to enter data and to validate the entered data EXCEPT any Read(s) that might be required to validate some entered data or codes, or to obtain some associated code descriptions.

BUSINESS EXAMPLE 2: An Exit includes all manipulation to format output and prepare some data attributes for printing (or output on a screen), including the human-readable field

headings[3] EXCEPT any Read(s) or Entries that might be required to supply the values or descriptions of some of the printed data attributes.

EXAMPLE: Consider two occurrences of a given functional process. Suppose that in the first occurrence the values of some attributes to be moved by a given data movement lead to a data manipulation sub-process A and that in another occurrence of the same functional process the attribute values lead to a different data manipulation sub-process B. In such circumstances, both data manipulation sub-processes A and B should be associated with this same one data movement and hence only the one data movement should be identified and counted in that functional process.

3.6 Control commands.

EXAMPLE of control commands:

- Commands to 'page up/down' or between physical screens.
- Hitting a Tab or Enter key, or pressing a button to continue.
- Clicking on an 'OK' button to confirm or cancel a previous action, or to acknowledge an error message or to confirm some entered data, etc.
- Functions that enable a user to control the display (or not) of a header or of sub-totals that have been calculated, or to sort a column with values in ascending or descending order.
- Menu commands that enable a user to navigate to one or more specific functional processes but which do not themselves initiate any one functional process.
- Commands to display a blank screen for data entry.

3.7 Error/confirmation messages and other indications of error conditions.

3.7.1 General examples

EXAMPLE: Error/confirmation messages may convey successes or failures of validation of entered data or for a call to retrieve data or to make data persistent, or for the response from a service requested of another piece of software.

BUSINESS EXAMPLE 1: In a human-computer dialogue, examples of error messages occurring during validation of data being entered could be 'format error', 'customer not found', 'error: please tick check box indicating you have read our terms and conditions', 'credit limit exceeded', etc. All such error messages should be considered as occurrences of one Exit in each functional process where such messages occur (which could be named 'error messages').

BUSINESS EXAMPLE 2: Functional process A can potentially issue 2 distinct confirmation messages and 5 error messages to its functional users. Identify one Exit to account for all these (5 + 2 = 7) error/confirmation messages. Functional process B can potentially issue 8 error messages to its functional users. Identify one Exit to account for these 8 error messages.

3.7.2 Error/confirmation messages with additional data.

BUSINESS EXAMPLE: A functional process of a bank's ATM (i.e. an 'automatic teller machine', or 'cash dispenser') can issue five types of messages in response to a request to withdraw a specific amount of cash:

[3] This example applies when measuring application software for use by humans, regardless of the domain. It would obviously not apply when measuring the size of re-usable objects which support the display of individual field headings on input or output screens.

- Error: machine has no available cash
- Error: the amount requested must be a multiple of $10
- Withdrawal refused. Account blocked. Contact the bank.
- Withdrawal refused (credit limit would be exceeded by $139.14)
- Withdrawal accepted; your remaining balance is $756.25

The first four messages describe an error condition and the first part of the fifth message is a confirmation. According to the rules on indications of error conditions, for all of this output, count one Exit. The last two messages also include data attributes related to the customer's account. Count one Exit for this data, to account for moving the customer's account data. In total, identify two Exits for this functional process, i.e. 2 CFP for its output.

3.7.3 Error conditions to human users which are not specified in the FUR.

BUSINESS EXAMPLE: A message passed on from the operating system could be 'printer X is not responding', ignore such messages.

3.8 Measuring the components of a distributed software system.

See the client/server example in section 3.4.8.

3.9 Re-use of software.

3.9.1 Functional processes that include common functionality.

BUSINESS EXAMPLE 1: Several functional processes in the same software being measured may need the same validation functionality for e.g. 'date of order', or may need to access the same persistent data, or may need to carry out the same interest calculation. Measure the common functionality as part of each functional process that requires it.

3.9.2 Functionality that is not in the FUR.

BUSINESS EXAMPLE 1: For a business application, the user that starts the application may be a scheduler component of the operating system, a computer operator, or any other human user (e.g. when a PC user launches a browser or word-processing software). The data conveyed by these users is not processed by the application as stated in the FUR, so must be ignored.

BUSINESS EXAMPLE 2: An application to process the input data for a variety of functional processes in batch mode may be started by a scheduler of the operating system. If the purpose is to measure the FUR of the batch application, the 'start-the-system' functionality should be ignored. The triggering Entries for the functional processes of the batch-processed application and any other Entries that may be required will form the input data for the batch application.

BUSINESS EXAMPLE 3: Exceptionally, a batch-processed application to produce summary reports at the end of a time-period may be started without needing any input data provided directly from the functional user. For the analysis see Business Example 2 in section 3.1.4.

3.10 Measurement of the size of changes to software.

3.10.1 Changes and their sizes.

EXAMPLE 1: A data manipulation is modified for instance by changing the calculation, the specific formatting, presentation, and/or validation of the data. 'Presentation' can mean, for example the font, background colour, field length, field heading, number of decimal places, etc.

EXAMPLE 2: Suppose a change request for a functional process requires three changes to the data manipulation associated with its triggering Entry and two changes to the manipulation associated with an Exit, as well as two changes to the attributes of the data group moved by this Exit. Measure the size of the change as 2 CFP, i.e. count the total number of data movements whose attributes and associated data manipulation must be changed. Do NOT count the number of data manipulations or data attributes to be changed.

EXAMPLE 3: Suppose a requirement to add or to modify the data attributes of a data group D1, such that after modification it becomes D2. In the functional process A where this modification is required, all data movements affected by the modification should be identified and counted as modified. So, if the changed data group D2 is made persistent and/or is output in functional process A, identify one Write and/or one Exit data movement respectively as modified.

If other functional processes Read or Enter this same data group D2, but their functionality is unaffected by the modification because they do not process the changed or added data attributes. These functional processes continue to process the data group moved as if it were still D1. So, these data movements in the other functional processes that are not affected by the modification to the data movement(s) of functional process A must NOT be identified and counted as modified.

3.10.2 Sizing a deletion of a part of an application.

EXAMPLE 1: Often, an obsolete part of an application is deleted ('disconnected' would be a better description) by leaving the program code in place and by just removing the link to the obsolete functionality. Suppose the functionality of the obsolete part amounts to 100 CFP but the part can be disconnected by changing, say, 2 data movements. The size of the functional change depends on the purpose. If the purpose is to use the size as input for project estimating, the size is 2 CFP. If the purpose is to determine the overall application size, the size of the change is 100 CFP.

EXAMPLE 2: If in the previous example the 'project size' of 2 CFP is measured, this should be clearly documented and distinguished from a measurement of the FUR which require that the application should be reduced in size by 100 CFP.

3.10.3 Changing an error/confirmation message

BUSINESS EXAMPLE: If an error/confirmation message is required to be changed (i.e. texts added, modified or deleted) it should be identified for measurement, regardless of whether or not the changed text is a consequence of a requirement to change another data movement.

3.10.4 Changing control commands and application-general data.

BUSINESS EXAMPLE: When the screen colour for all screens is changed, this change should not be measured.

3.11 Extending the COSMIC measurement method.

EXAMPLE 1: The COSMIC method could be extended to measure separately and explicitly the size of the FUR of data manipulation sub-processes.

EXAMPLE 2: The method could be extended to measure separately the influence of the number of data attributes per data movement on software size.