COSMIC Measurement Manual for ISO 19761

Part 3: Examples

Version 5.0

March 31 2020

Foreword.

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

Part 1: Principles, definitions & rules.

Part 2: COSMIC Group Guidelines.

Part 3: Examples of COSMIC concepts and measurements.

This Part 3 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, with an introduction to and characterization of the intention of examples in italics.

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.

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:

Diana Baklizky, (Brazil),
Jean-Marc Desharnais, Ecole de technologie supérieure – University of Quebec (Canada),
Cigdem Gencel, Free University of Bozen-Bolzano (Italy),
Dylan Ren, Measures Technology LLC (China),
Bruce Reynolds, Telecote 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.

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).

The COSMIC method is used only to measure the size of FUR. Requirements often include systems non-functional requirements which cannot be measured by COSMIC. However, systems requirements initially appearing as NFR may evolve into software FUR as a project progresses. It is therefore important to distinguish the two types of requirements and the evolution of the requirements.

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.

REAL-TIME EXAMPLE: Dependability or fault tolerance requirements for aerospace systems are achieved mostly through a combination of redundancy and backup of the physical systems. A function, such as engine monitoring, is implemented on two or more separate embedded computers. This function has a strict timing constraint stated as a NFR: 'each separate computer must respond within a specific time. If any one of the computers repeatedly responds later than the required time, or its results disagree with the others, it must be out-voted' (by a mechanism specified as a functional requirement). A requirement for fault tolerance that when initially stated may appear as non-functional therefore evolves into FUR that can be measured. The timing mechanism can also be partly implemented in software and this functionality can also be measured (see for example the Guideline for sizing real-time software, section 3.2).

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.

The COSMIC method is concerned only with types of things, not with occurrences of things of the same type. As it has big influence on size, it is of critical importance to apply this to all things used in measurement, such as functional users, objects of interest, data groups etc..

Examples of the Software Context Model.

BUSINESS EXAMPLE 1: 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.

REAL-TIME EXAMPLE 1: The embedded software of a digital radio sends its output to a pair of stereo loudspeakers. The software sends separate signals of the same type to each of the two loudspeakers. They each convert the received electrical signal into sound in the same way. A context model of the software would show one functional user type 'loudspeaker' of which there are two occurrences.

Examples of the Generic Software Model.

BUSINESS EXAMPLE 2: 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.

REAL-TIME EXAMPLE 2: Suppose a functional process that must control the temperature of an oven once every ten seconds. The functional process will be executed, i.e. it will occur once every 10 seconds. During its execution, the Exit data movement of the process to switch the heater on or off may or may not be executed i.e. it may or may not occur at all in any cycle, depending on whether the heater must be switched on or off, or left in its current state. The Exit data movement is counted once in the functional process, regardless of whether or not it occurs in a particular execution.

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.

A measurement is performed to satisfy the information needs of a stakeholder. It is important for the measurer to understand this purpose in order that the stakeholder needs are satisfied.

EXAMPLES: 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.

The scope of a measurement may be the whole or part of the software, and depends of the purpose of the 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.

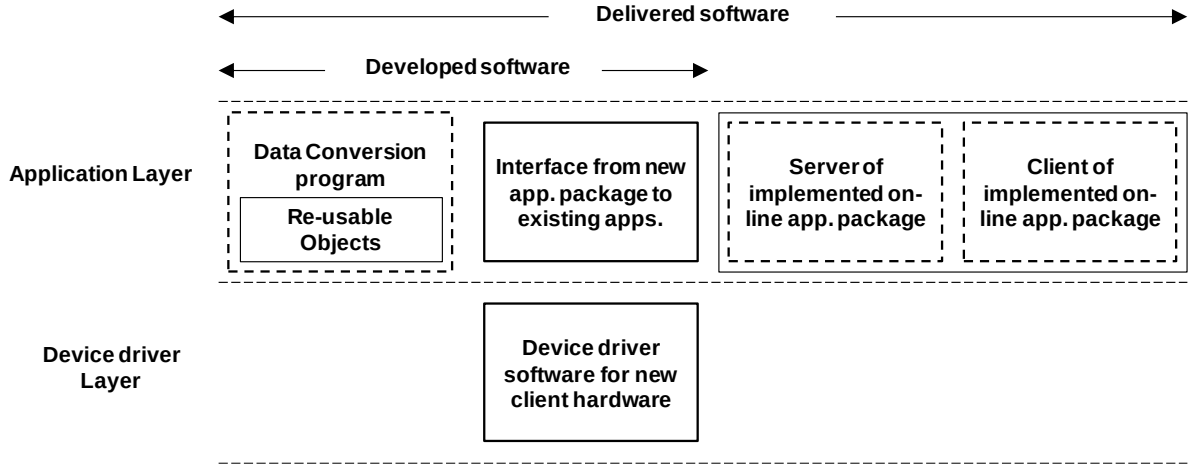


Figure 2.1 - The overall scope of the deliverables of a software project and the individual measurement scopes.

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.

Multiple occurrences of a functional user may be responsible for entering data to the same functional process. However, the COSMIC method is concerned only with types, not with occurrences (see 1.5).

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.

REAL-TIME EXAMPLE 1: Each wheel of a car has a sensor that obtains the pressure of its tire. At regular intervals, a functional process must obtain the pressure of all four tires. If the pressure is too low or too high - the range of safe pressures is in the software - the software shows which tire has a pressure problem indicated on a diagram of the four wheels on a display screen at the dashboard. The functional users are the four sensors and the four indications on the display screen. However, the four sensors are subject to the same requirement (and idem for the indications on the screen), so identify one functional user type 'sensor' and one functional user type for the indications on the display screen.

Different functional users requiring different functionality.

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.

FUR are written from the viewpoint of its functional users. As each functional user may perceive a system in different ways, requirements will be expressed depending on the functional users. It is important that all requirements to be measured are expressed at the same functional user viewpoint if their sizes are to be compared.

REAL-TIME EXAMPLE 2: 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

[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.

two views; a size measurement from a ‘mixed’ human/hardware view would be very difficult to interpret.

2.4 Measurement Strategy Patterns.

See also examples in the Guideline for Measurement Strategy Patterns.

2.5 Layers.

Software often has an architecture whereby the functionality is distributed across several components. The components communicate with one another using messages following rules set out in their requirements. In turn, components addressing a similar concern may be included in a layer. Within a layer, the components may be peers, clients or servers. COSMIC states that only the sizes of the components in the same layer are comparable and can be aggregated.

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 Business Example 2 below).

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

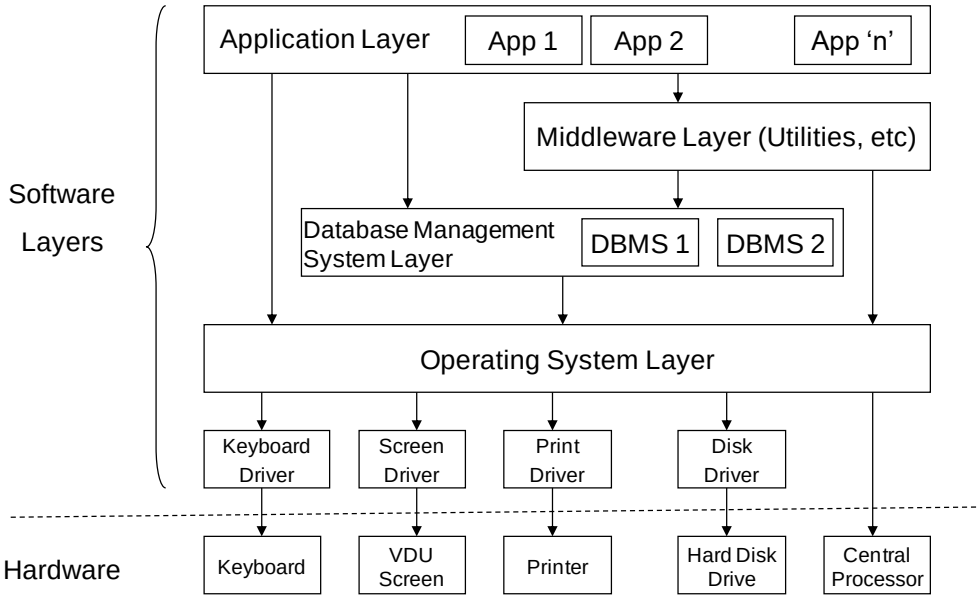


Figure 2.2- Typical layered software architecture for a Business/MIS system.

REAL-TIME EXAMPLE 1: The physical structure of a typical layered software architecture supporting a piece of embedded real-time software is given in Figure 2.3. (Note: simple uni-tasked real-time embedded software may not need a real-time operating system.)

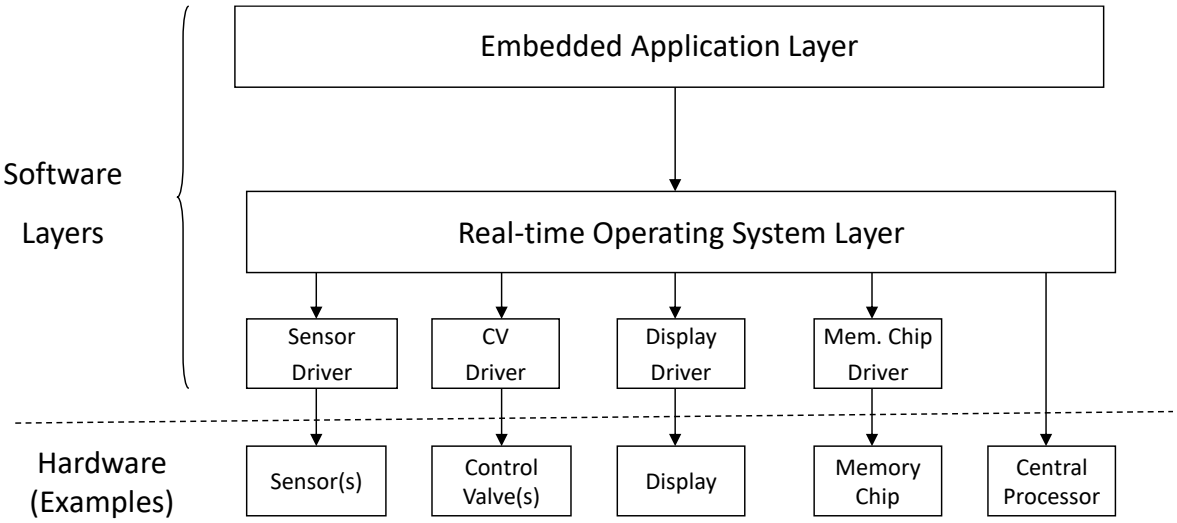


Figure 2.3 - Typical layered architecture for a real-time embedded-software system.

REAL-TIME EXAMPLE 2: The ISO 7-layer (OSI) model for telecommunications. This defines a layered architecture for which the hierarchical correspondence rules for the layers of the message-receiving software are the inverse of the rules for the layers of the message-transmitting software.

REAL-TIME EXAMPLE 3: The 'AUTOSAR' architecture of the auto industry that exhibits all the different types of correspondence rules between layers now described in the principles for a layer.

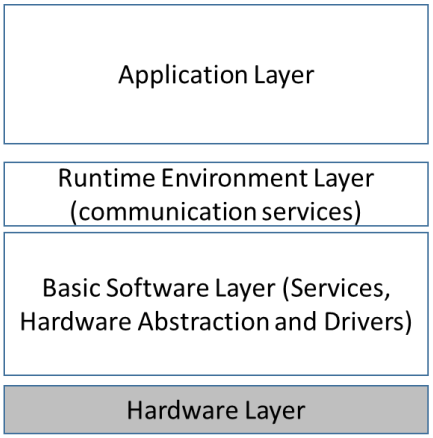


Figure 2.4 – The structure of the AUTOSAR architecture

A software architecture that exhibit different layers depending on the 'view' of the architecture.

BUSINESS EXAMPLE 2: Consider an application A situated in a layered software architecture, as in Figure 2.5 below, which shows three possible layer structures a), b) and c) according to different architecture 'views'.

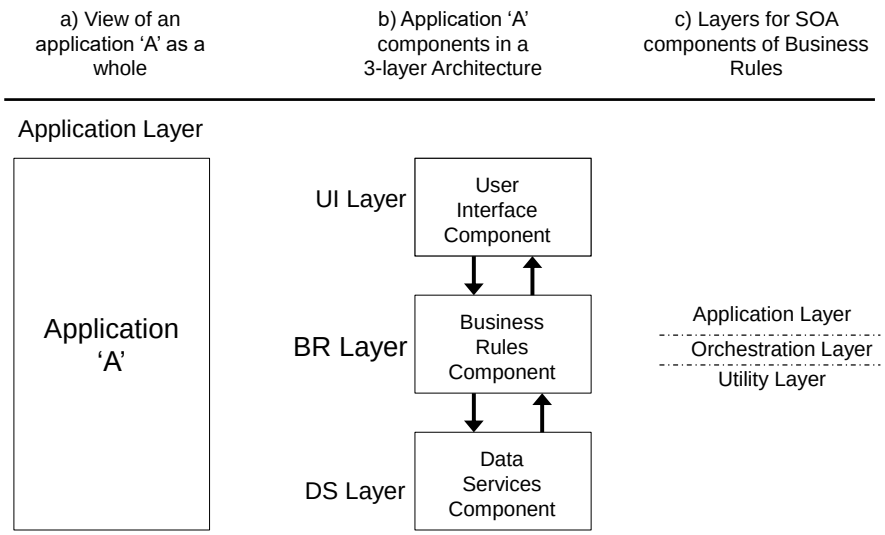


Figure 2.5 - Three views of the layers of an application.

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.

Software may consist of various levels of components, known in COSMIC as levels of decomposition. It is important to measure at the level of decomposition appropriate to the purpose of the measure, as sizes of components of a piece of software are only directly comparable for components at the same level of decomposition.

EXAMPLE: The Guideline for 'Measurement Strategy Patterns' recognizes three standard levels of decomposition: 'Whole application', 'Major Component' and 'Minor component'. See business example 2 above, where the three levels are shown in Figure 2.5.

2.7 Context diagrams.

A context diagram is useful to visualize the software to be measured in the context of its functional users and persistent storage.

BUSINESS EXAMPLE: Figure 2.6 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.

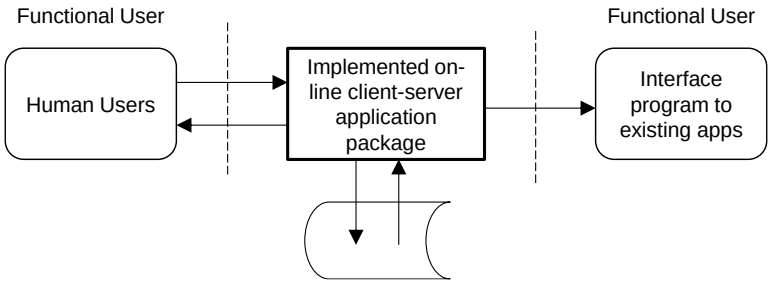


Figure 2.6 - Context diagram for the client-server application.

REAL-TIME EXAMPLE: Figure 2.7 shows the context diagram for a simple intruder alarm embedded software system.

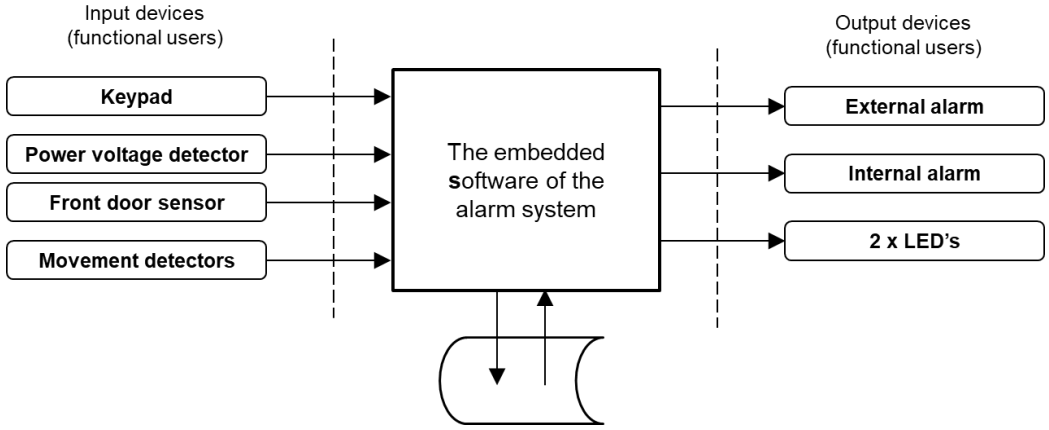


Figure 2.7 - Context diagram for the embedded software of an intruder alarm system.

2.8 Levels of granularity.

In the early stages of development, requirements evolve as information is gathered and understanding increases. In the early stages individual functional processes, and the information needed to identify and measure them may not be there. In this case the size can be estimated or assessed using a variety of methods, see the ‘Early Software Sizing with COSMIC: Guidelines’. Functional requirements at a higher level of granularity describe e.g. groups of functional users, rather than individual humans or individual engineered devices or individual pieces of software.

When the requirements for a COSMIC measurement are present, the requirements must be at the so-called functional process level of granularity. It is important therefore to ensure that requirements are at this level of granularity before measurement commences.

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’.

REAL-TIME 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 2.8 (a).

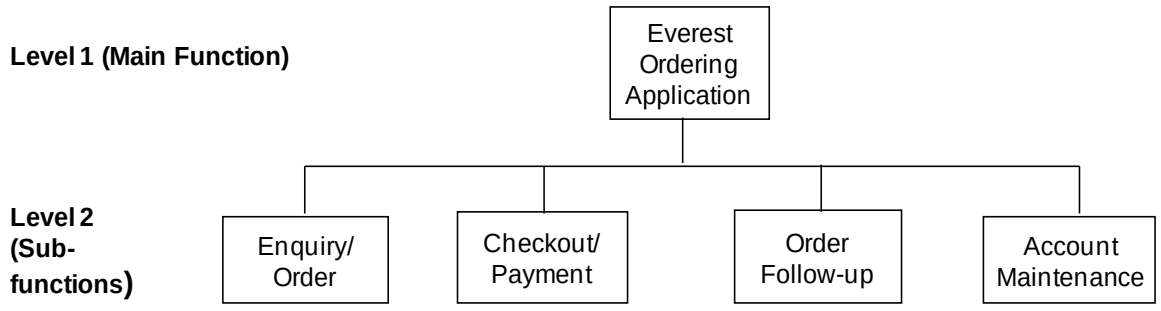


Figure 2.8 (a) - Analysis of the Everest Ordering System: the first two levels of granularity.

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 2.8 (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.

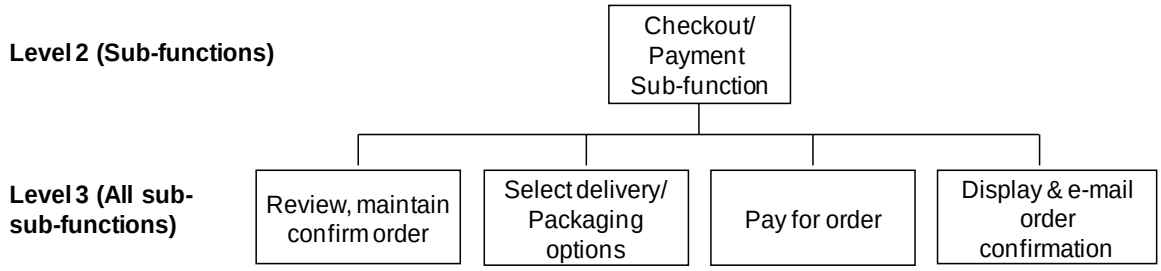


Figure 2.8 (b) - Analysis of the Checkout/Payment Sub-function.

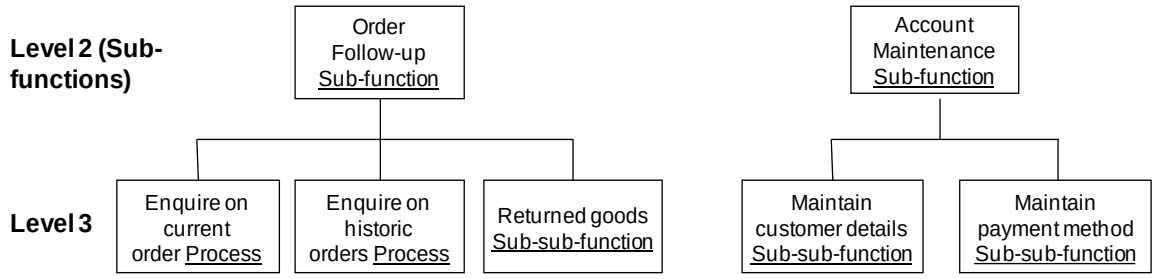


Figure 2.8 (c) - Analysis of the Order Follow-up and Account Maintenance Sub-function.

Figure 2.8 (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 2.8 (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.

In the COSMIC method, the identification of the functional processes is essential if a correct measure of size is to be obtained. The following are examples of how functional processes can be identified.

A human functional user enters data, which starts 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: 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.

A device functional user conveys data, which starts a functional process.

REAL-TIME EXAMPLE 1: A functional process of a real-time software system may be started by its triggering Entry informing the functional process that a clock (functional user)

has ticked. The data group moved conveys data (the tick, perhaps via a single bit) that informs only that an event has occurred.

REAL-TIME EXAMPLE 2: A functional process of an industrial real-time fire detection software system may be started by its triggering Entry initiated by a specific smoke detector (functional user). The data group generated by the detector conveys the information 'smoke detected' (an event has occurred) and includes the detector ID (i.e. data that can be used to determine where the event occurred).

REAL-TIME EXAMPLE 3: A bar code reader (a functional user) at a supermarket checkout starts a scan when a bar code appears in its window (the triggering event). The reader generates a data group, comprising an image of the bar code that is input to the checkout software. The data group image is moved by a triggering Entry into its functional process. The latter adds the product cost to the customer's bill if the code is valid, sounds a 'beep' to inform the customer that the product has been accepted, and logs the sale etc.

Identify separate triggering events - and therefore separate functional processes - when a human functional user makes decisions outside the software on 'what to do next' that are independent in time and that require separate responses from the software.

BUSINESS EXAMPLE 3: 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).

Identify separate triggering events and separate functional processes when the responsibilities for activities are separated.

BUSINESS EXAMPLE 4: 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.

BUSINESS EXAMPLE 5: 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.

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.

BUSINESS EXAMPLE 6: Suppose the orders in the Business Example 1 above 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.

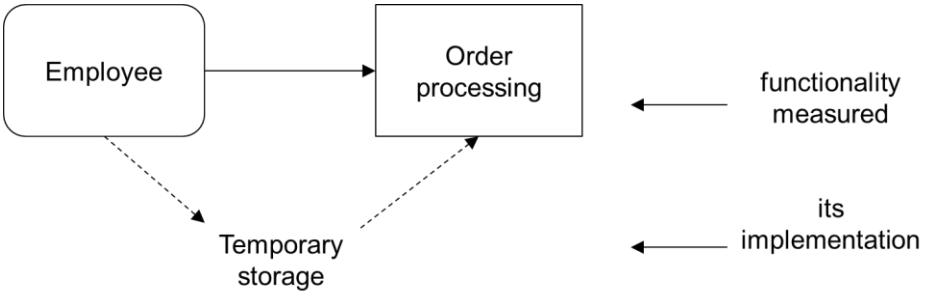


Figure 3.1 – Batch processing of Business Example 4

BUSINESS EXAMPLE 7: 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.

In the real-time applications domain a sensor typically triggers a functional process.

REAL-TIME EXAMPLE 5: When a sensor (functional user) detects that the temperature reaches a certain value (triggering event), the sensor sends a signal to initiate a triggering Entry data movement of a functional process to switch off a heater (another functional user).

REAL-TIME EXAMPLE 6: A military aircraft has a sensor that detects the event ‘missile approaching’. The sensor is a functional user of the software that must respond to the threat. For this software, an event occurs only when the sensor detects something, and it is the sensor (the functional user) that generates a data group to initiate a triggering Entry saying, e.g. ‘sensor 2 has detected a missile’, plus maybe a stream of data about how fast the missile is approaching and its co-ordinates.

Identify functional processes on basis of the organization of data input or examine the menus for some installed software.

BUSINESS EXAMPLE 8: 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.

Identify only the data movements of a functional process; the various processing paths in which they may occur do not lead to separate functional processes.

BUSINESS EXAMPLE 9: 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 10: 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 11: Continuing from Example 10, 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.

REAL-TIME EXAMPLE 12: One triggering Entry (aircraft altitude information sent by the Geographical Positioning System) to a functional process of an avionics system will lead to one of two quite different processing paths within the functional process depending on the value of the Entry, i.e. whether the altitude is above or below a given height. The different paths will display different data groups on the pilot's map and, if the altitude is too low, additional warnings will be issued. There is only one functional process.

3.2 Data groups and objects of interest.

Data groups and objects of interest are embedded in several forms in the text of the requirements. It is the responsibility of the measurer to identify them within the context of the function process being measured.

EXAMPLE 1: 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.

EXAMPLE 2: Read-only memory is persistent storage if retrieval of its data is required by the FUR.

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: 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 3: Assume the same ad hoc enquiry against a personnel database as in Business Example 2, 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 2 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.

BUSINESS EXAMPLE 4: 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.

Identifying data groups and objects of interest in the real-time software domain.

REAL-TIME EXAMPLE 1: A data group entering software from a physical device informs about the current state of the device. In this case the device is the object of interest (and functional user) and the data group conveys its state, such as that a valve is open or closed, leading to the start of a functional process. The physical device is the object of interest. Similarly, a data group output to a device, such as to switch a warning lamp on or off conveys state data about the lamp object of interest.

REAL-TIME EXAMPLE 2: A message-switch software system may receive a message data group as input and route it forward unchanged as output, as per the FUR of the particular piece of software. The attributes of the message data group could be, for example, 'message ID, sender ID, recipient ID, route code and message content'; the object of interest of the message is 'message'.

REAL-TIME EXAMPLE 3: A reference data structure, represents objects of interest whose attribute-values are given in tables found in the FUR, and which are held in permanent memory (ROM memory, for instance) and accessible to most of the functional processes found in the measured software.

REAL-TIME EXAMPLE: 4: Files, commonly designated as 'flat files', represent objects of interest mentioned in the FUR, which are held on a storage device.

A functional user may be an object of interest.

BUSINESS EXAMPLE 5: 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.

REAL-TIME EXAMPLE 5: Suppose a temperature sensor A sends a measure of the current temperature of a material for processing by a functional process. The sensor provides information about its own state and is thus object of interest of the information data group.

3.3 Data attributes.

The COSMIC method does not directly require identification of the data attributes associated with a data group. However, the Guidance on Rules 13 and 14 – Data Movements Uniqueness in section 3.4 of Part 2 requires examining the 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.

REAL-TIME EXAMPLEs: A temperature sensor may, on request, report the attribute 'Temperature'. A sensor of a security system may detect an intruder and send the attribute 'Movement detected'. A message in transmission may consist of the attributes 'From (address), To (address), Contents'.

3.4 Data movements.

Data movement(s) and clock-ticks.

REAL-TIME EXAMPLE 1: For a clock-tick event occurring every 3 seconds, identify an Entry moving a data group of one data attribute. The object of interest (and functional user) is the Clock, the data group conveys the state of the Clock.

Data movement(s) and obtaining the date and/or time from the system's clock.

BUSINESS EXAMPLE 1: 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.

Data movement(s) and any data that are not related to an object of interest to a functional user.

BUSINESS EXAMPLE 2: 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 3: 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).

Data movement(s) for all data describing any one object of interest.

BUSINESS EXAMPLE 4: The most common case is that one Write would be identified that moves a data group containing all the data attributes of an object of interest required to be made persistent in a given functional process.

Data movement(s) when different functional users move different data groups describing the same object of interest.

REAL-TIME EXAMPLE 2: A functional process is required to accept different data groups from two different seismometers (functional users) each responding to the same event e.g. a test explosion. Identify two Entries.

BUSINESS EXAMPLE 5: 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.

Data movement(s) for moving different data groups describing the same object of interest to/from persistent storage.

BUSINESS EXAMPLE 6: 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 7: 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.

Data movement(s) for repeated occurrences of any data movement describing the same object of interest.

BUSINESS EXAMPLE 8: 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 9: 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.

REAL-TIME EXAMPLE 3: Suppose in a real-time functional process, the FUR requires that the same identical data group must be entered from a given functional user, e.g. a hardware device, twice at a fixed time interval in order to measure a rate of change during the process. In COSMIC the two movements of data are considered to be multiple occurrences of the same Entry. Only one Entry may be identified for this data group for this functional process. Note that there is no data manipulation associated with the two occurrences of the Entry, the calculation of the rate of change is associated with the Exit that reports the rate.

REAL-TIME EXAMPLE 4: Suppose a process control system for a machine that produces a flat product such as paper or a plastic film. The machine has an array of 100 identical sensors across the direction of movement of the product to detect breaks or holes in the product. The functional process that must check for breaks or holes receives the same data from each sensor. The position of e.g. a hole in the product can be determined from the position in the string of data values sent from the array of sensors, The processing of data from all the sensors in the array is identical. Identify one functional user for all the sensors and one Entry for the data obtained from all the sensors by the functional process.

Data movement when an Enquiry outputs fixed text.

BUSINESS EXAMPLE 10: 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.

Data movement(s) when a functional process is required to move a data group from persistent storage.

EXAMPLE 1: This example concerns a piece of software A that is required to retrieve a stored data group where the FUR of software A are not concerned with how those data accesses are handled by any other software in the same or different layers.

The functional users of the software A could be, for example, human users if the software A is in the application layer making an enquiry on a stored data group. Figure 3.2 shows the COSMIC data movements for this enquiry. The enquiry is triggered by an Entry, followed by a Read of the data group from persistent storage and then an Exit with the enquiry result. FP A is not concerned with where the data is retrieved from, only that it is persistent data.

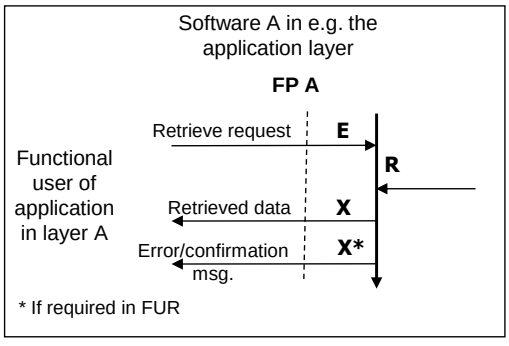


Figure 3.2 – Solution for a Read issued by software A in the application layer.

An exactly analogous model would apply if the functional process FP A were required to make a data group persistent via a Write data movement. According to rule d) for error

conditions in Part 1, the Read and the Write data movements are considered to account for any return code or reporting of an error condition.

Figure 3.2 shows a possible application-specific error message that could be issued by FP A if, for example, the requested record is not found. However, an error condition not specific to the application, e.g. 'disk failure' would not be counted as an Exit for FP A. See also section 4.9 in Part 1 on error/confirmation messages.

Data movement(s) when a functional process is required to obtain some data from another piece of software.

EXAMPLE 2: 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 3.3 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.

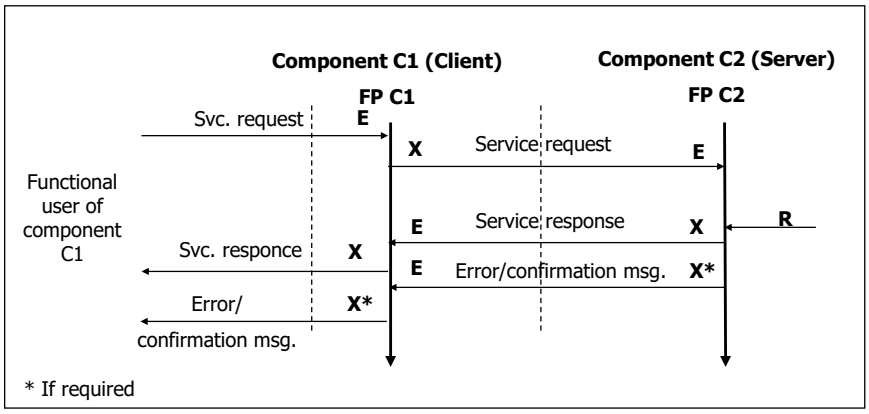


Figure 3.3 – Data exchanges between client and server components.

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 2.2. 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 3.3 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 Example 2 enquiry therefore requires 6 data movements (i.e. 6 CFP) to satisfy the enquiry request for component C1 and 4 CFP for component C2. This compares with the 4 CFP (1 x E, 1 x R and 2 x X) that would have been required for component C1 if it had been able to retrieve the data group from persistent storage within its own boundary via a Read as shown in Figure 3.3.

Component C2 will probably use the services of some storage device driver software in another layer of the software architecture to retrieve the data from the hardware, as in Example 4, Figure 3.5 (b).

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

EXAMPLE 3: See Figure 3.4. Suppose a piece of software A to be measured is allowed to retrieve certain stored data Z 2, 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 3.

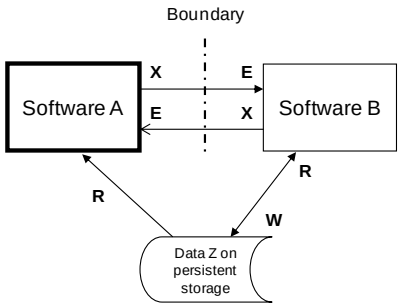


Figure 3.4 – Persistent data Z within the boundary of both software A and B for a Read.

Data movement(s) when persistent data is obtained by the device driver software that interacts with the physical storage device.

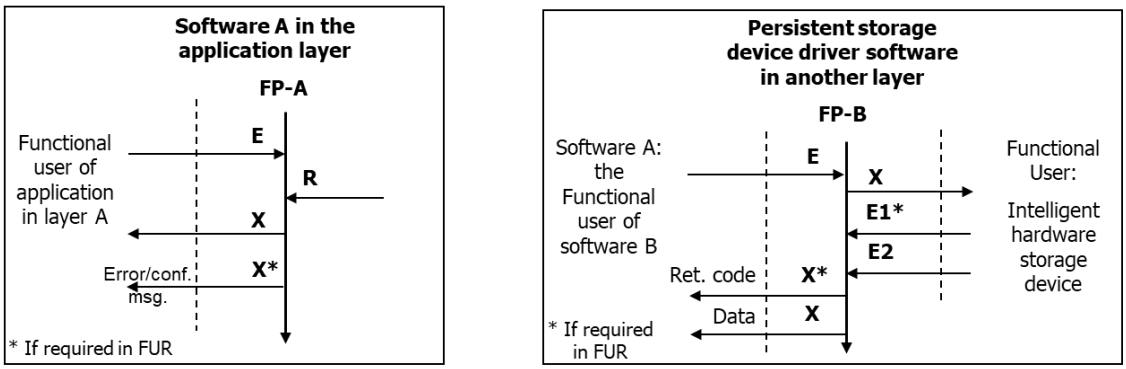
INFRASTRUCTURE EXAMPLE: This example concerns the piece of software A of Example 1 that is required to retrieve a stored data group. Consider a separate piece of software B that is the device driver for the intelligent hardware storage which holds the data group that the software A is required to access. (Ignore the probable presence of an operating system for simplicity; the operating system effectively transmits application requests to the device driver software and returns the results of requests.)

The two pieces of software are in different layers in an architecture such as shown in Figure 2.2. Software A is in e.g. the application layer, and software B is in a device driver layer. Physically, there is probably a hierarchical relationship between the two pieces and (ignoring the operating system) a physical interface between software in the two layers, as shown for example in Figure 2.2. However, the models of the functional processes of software A and B are independent of the nature of the relationship between the layers, which may be hierarchical or bi-directional.

The functional users of the software B in the driver layer are the software A (ignoring the operating system) and the intelligent hardware storage device which holds the required data. ('Intelligent' means that the device must be told what data is needed.)

Suppose that an enquiry functional process FP A of the software A needs to retrieve a stored data group. Figure 3.5 (a) shows the COSMIC model of this enquiry. Figure 3.5 (b) shows the functional process FP B of the software B in the device driver layer that handles the

physical retrieval of the required data from a hardware storage device (such as a disk or USB memory stick).



Figures 3.5 (a) and (b) – Solution for a Read issued by software A in the application layer to software B in the device driver layer.

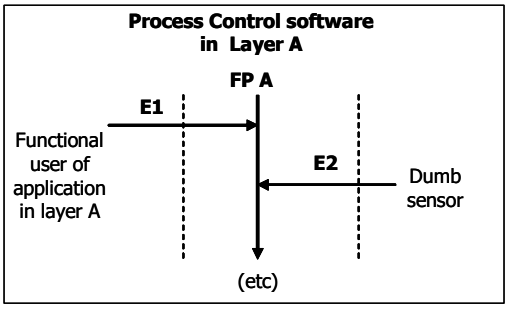
Figure 3.5 (b) shows that the Read request of the software A is received as a triggering Entry to the functional process FP B, which passes on the request as an Exit to the hardware device. The response of the latter depends on the particular hardware device. The device may just return the requested data, shown as Entry E2 in Figure 3.5 b). The device may also issue a separate error message describing the success or the reason for the failure of the request, e.g. 'data not found', or disk error', shown as Entry E1* in Figure 3.5 b). FP B returns the data to the software A as an Exit. FP B also normally issues a 'return code' describing the success or reason for the failure of the request. (Although the return code may be physically attached to the returned data, it is logically a different data group to that of the returned data – it is data about the outcome of the request process). For FP A no Entry for these messages is identified, as the Read data movement accounts for the retuned data and error messages, as Reads and Writes account for any associated reporting of error conditions. For FP A, an Exit is identified for an error/confirmation message, if required.

Note: in practice, there may be more data movements between the device driver software and the intelligent hardware device than are shown in Figure 3.5 b). For example, this Figure does not show the effect of the device driver measuring a timeout for non-response from the hardware.

Data movement(s) when a functional process does not need to tell the functional user what data to send.

REAL-TIME EXAMPLE 5: Suppose a functional process of a real-time process control software system is required to poll an array of identical dumb sensors. At the application level, the request for the data by the functional process and the receipt of the data is accounted for by one Entry. (Since the sensors are identical one Entry is identified and counted.)

Suppose further that the request for the data must in practice be passed to a piece of device driver software in a lower layer of the software architecture, which physically obtains the required data from the sensor array as illustrated in the layered architecture of Figure 2.3. The functional processes of the process control software and of the device driver software for the dumb sensors would be as shown in Figures 3.5 (a) and (b) below.



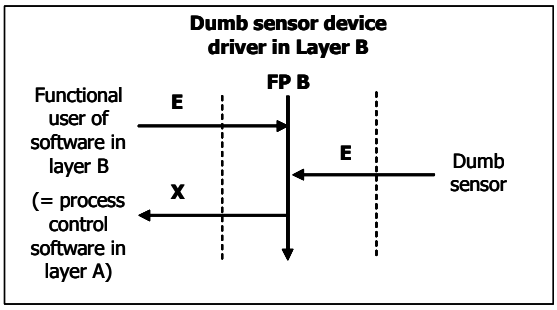


Figure 3.6 (a) and (b) – Solution for a poll of dumb sensors.

Figure 3.6 (a) shows that the software functional process FP A is triggered by an Entry E1 e.g. from a clock tick. This functional process then obtains data via Entry E2 from the dumb sensor array to receive the multiple occurrences of the sensor readings. The dumb sensors are also functional users of the process control software. (The device driver software is hidden at this level.)

Figure 3.6 (b) shows the model for the software that drives the dumb sensor devices. It receives data via an Entry from the process control software (probably in practice via an operating system) as the trigger of a functional process FP B. This functional process obtains the required data via an Entry E from its functional user, the dumb sensor array.

The data group is passed back to the process control software via an Exit. This Exit is received as the Entry E2 by the functional process FP A. FP A then continues with its processing of the sensor data. Again, the fact that there are multiple occurrences of this cycle of gathering data from each of the identical sensors is irrelevant.

The apparent mis-match between the one Entry E2 from a dumb sensor to the process control software and the Entry followed by an Exit data movement of the device driver software is due to the convention that an Entry from a dumb sensor is considered to include any 'request to enter' functionality since the dumb functional user has no capability of dealing with any message from a functional process.

Data movement(s) when the functional user needs to be told what to send.

REAL-TIME EXAMPLE 6: Suppose a functional process sends to one of its functional users, such as an 'intelligent' hardware device or another peer piece of software, some parameters for an enquiry or the parameters for a calculation, or some data to be compressed. The response from the functional user is obtained via the functional process issuing an Exit, followed by the receipt of an Entry data movement, as described in section 3.4, Example 2.

3.5 Data manipulations associated with data movements.

Data manipulation is considered to be accounted for by the associated data movements. For data manipulation in the context of changes to requirements that must be measured see 3.10.

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.

In the COSMIC method control commands are not data movements. It is important to identify control commands in order to prevent them being identified as such.

EXAMPLES 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.
- 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.

Error and confirmation messages, if required, are subject of measurement.

EXAMPLES: 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] 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.

An error/confirmation message with additional data.

BUSINESS EXAMPLE 3: 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:

- 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.

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

BUSINESS EXAMPLE 4: A message passed on from the operating system could be 'printer X is not responding', ignore such messages.

REAL-TIME EXAMPLE 1: In a real-time system, a functional process that periodically checks the correct functioning of all hardware devices might issue a message that reports 'Sensor S has failed', where 'S' is a variable. This message should be identified as one Exit in that functional process to account for moving data about the functional user (and object of interest) Sensor S.

Data values indicating an error condition together with other data.

REAL-TIME EXAMPLE 2. The FUR of Real-Time Example 6 in section 3.4 may also state that the FP's A and B must handle an error condition when the device driver software fails to obtain the data from one or more of the array of dumb sensors. A dumb sensor cannot, by definition, issue an error message. The device driver FP B will, most likely, obtain a string of values from the array of dumb sensors, e.g. state 1, state 2, state 3, no response, state 5, no response, state 7, etc. and will issue this string as an Exit to the FP A of the application where it is received as an Entry. No separate error message should be identified as an Exit from FP B of the device driver software, nor as an Entry to FP A of the process control application.

3.8 Measuring the components of a distributed software system.

See Example 2 in section 3.4.

3.9 Re-use of software.

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.

REAL-TIME EXAMPLE 1: Several functional processes in the same software being measured may need to obtain data from the same sensor (common movement of same data group) or may need to carry out the same scale conversion calculation, e.g. from Fahrenheit

to Centigrade (common data manipulation). Measure this common functionality as in the preceding example.

Functionality that is not in the FUR.

BUSINESS EXAMPLE 2: 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.

REAL-TIME EXAMPLE 2: For a real-time application, the user that starts the application may be the operating system or network management generating a clock signal, or a human operator (e.g. to start a process control system from an operator workstation) must be ignored.

INFRASTRUCTURE EXAMPLE: For a computer operating system, the user that starts the operating system is a bootstrap program that is started when the computer power is switched on must be ignored.

.BUSINESS EXAMPLE 3: 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 4: 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 7 in section 3.1.

REAL-TIME EXAMPLE 3: A modern vehicle has a distributed system of Electronic Control Units (ECUs) to control many functions, e.g. engine management, brakes, air-conditioning, etc. In the AUTOSAR architecture, in a distributed system, the 'Network Management' (NM) module, which is always running, is responsible for activating the ECUs that are connected together via a network ('bus'). This NM module also handles the coordinated switching between the ECU operating states: Normal Operation, Low Power and Sleep. Therefore it is the NM that wakes up or puts to sleep ECUs. When measuring any ECU application software, this NM functionality should be ignored.

3.10 Measurement of the size of changes to software.

The size of change in the COSMIC method include changes to any of the elements contributing to size, including the data manipulations of data movements.

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.

IIf 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.

EXAMPLE 4: 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 the size is 2 CFP. Iff the purpose is to determine the overall application size, the size of the change is 100 CFP.

For estimation purposes it may be advisable to use a different productivity for a part of the functional change, since disconnecting is quite different from 'real' deletes. Alternatively, for estimating purposes it may be preferable to measure the size that will be implemented rather than the size of the requirement.

EXAMPLE 5: 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.

BUSINESS EXAMPLE 1: 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.

Changes to control commands and application-general data.

BUSINESS EXAMPLE 2: When the screen colour for all screens is changed, this change should not be measured.

3.11 Extending the COSMIC measurement method.

Although not a formal part of the COSMIC method, any aspect of the FUR that cannot be measured using the rules of the method may be subject to a separate sizing process. The size however is not to be reported as CFP.

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.