COSMIC Measurement Manual for ISO 19761

Part 3c:
MIS Examples

September 2024

FOREWORD

This document consolidates the examples from the previous Business Applications Guideline and the business examples from the previous Measurement Manual Part 3a. The examples are ordered per COSMIC phase and where needed grouped per topic. These topics are visible in the Table of Contents for ease of search. This document does not contain any explanation of the COSMIC method, if needed consult the COSMIC Measurement Manual Parts 1 and 2. Here and there the term 'convention' (in capitals) was introduced to avoid confusion with the rules of Part 1.

The COSMIC Measurement Manual describes the core measurement method. This version 5.0 consists of the parts:

Part 1: Principles, definitions & rules* (17 pages)

Part 2: Guidelines* (18 pages)

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

Part 3a: Standard Measurement Strategy Examples (13 pages)

Part 3b: Real-time Examples (32 pages)

Part 3c: MIS Examples (58 pages)

Part 3d: Standardized Requirements with Big Data Cleaning Examples (12 pages)

* Parts 1 and 2 describe the entire material necessary for certification.

September 2024 editing:

The title and examples of section 2.9.2 illustrate the new guidelines a) and b) in section '3.7 Measurement of the size of changes to software' of Part 2 - Guidelines.

Further explanations, guidelines, translations, practical examples and other publications are available from www.cosmic-sizing.org.

A public domain version of the COSMIC documentation and other technical reports, including translations into other languages can be obtained from 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:

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 THE MEASUREMENT STRATEGY PHASE.

1.1 Purpose and scope.

1.1.1 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.

1.1.2 Performance and estimating

EXAMPLE 1: If the purpose of the measurement is to determine the software project productivity (or another performance parameter) and/or to support estimating, and the business application is one piece of software running on one technical platform, then the scope will be the whole application.

EXAMPLE 2: If the purpose is to measure the work-output of a project that must develop some application software and also some software that will reside in other layers, then a separate measurement scope must be defined for each piece of software in each layer.

EXAMPLE 3: Suppose the purpose of a measurement is related to development project performance measurement or estimating, the application must be distributed over different technical platforms, and it is known that development productivity varies with the platform. Almost inevitably, the measurer will need to measure separately the size of each component of the application that runs on a different platform by defining a separate measurement scope for each of the components. The performance measurement or estimating can then be carried out separately for each component on each platform.

EXAMPLE 4: If the purpose is to measure the total size of an application portfolio (excluding any infrastructure software) in order to establish its financial value, e.g. at replacement cost, then it may be sufficiently accurate to measure sizes ignoring any functionality arising from any distributed architecture. A separate measurement scope should be defined for each application to be separately measured. (Sizes may also be measured to sufficient accuracy using an approximation variant of the

COSMIC method to speed up this task, see the Early Sizing Practitioners and Experts Guides). An average productivity for replacing the whole portfolio can then be used to determine the replacement cost.

EXAMPLE 5: Suppose over 90% of the FUR for a new business application will be implemented by a standard package, and the remaining 10% by custom software. The purpose is to estimate the effort to implement the whole FUR. For the scope of the FUR that will be satisfied by the package, it would be advisable to consult with the package supplier on how to estimate the effort. (Measuring the functional size of these FUR may or may not be useful; measuring the size of the package is almost certainly irrelevant to the purpose.) The scope of the 10% of the FUR that will be implemented by custom software can be measured in the normal way and effort estimated using a productivity appropriate for the technology of the custom software.

EXAMPLE 6: 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.

1.1.3 Overall scope and measurement scope.

EXAMPLE 1: In any particular software customer/supplier relationship it is always possible for the two parties to limit the scope of the measurement of the software supplied in any way that sensibly satisfies the purpose of the measurement.

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

EXAMPLE 2: Figure 1.1 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.

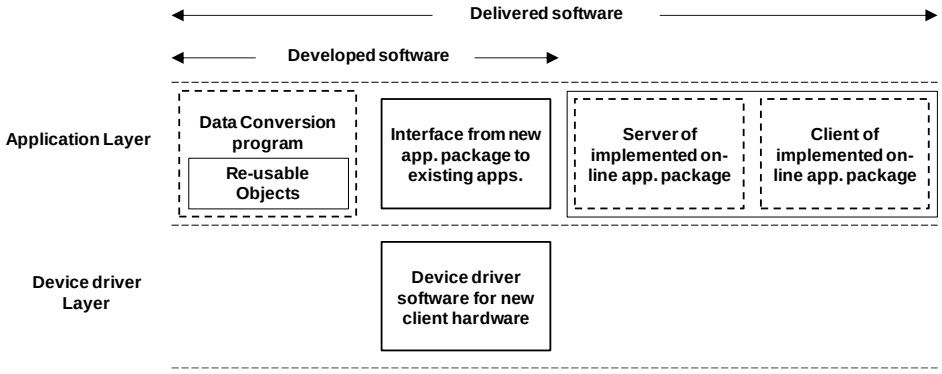


Figure 1.1 - The overall scope and the measurement scopes.

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.

EXAMPLE 3: Whether or not data conversion software should be included in the scope for a particular measurement depends on the purpose of the measurement. If the purpose is to measure the work-output of a project team, then the size of any data conversion software would be included in the scope. If the purpose is to measure the size of the delivered application and maybe the size of the change, then data conversion software would be excluded from the scope.

1.1.4 Measurement patterns.

EXAMPLE: A measurement pattern defines a standard combination of parameters that must be determined in the Measurement Strategy phase of a COSMIC measurement process. The three most common patterns for business application software are:

- On-line business applications considered as a whole ('Whole application');
- Batch processed business applications ('Whole Application');

- On-line business applications considered as components of a multi-tiered implementation.

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

1.2 Functional User Requirements (FUR).

1.2.1 Functional requirements.

EXAMPLE: Functional User Requirements include but are not limited to:

- data transfer (for example Input customer data, Send control signal);
- data transformation (for example Calculate bank interest, Derive average temperature);
- data storage (for example Store customer order, Record ambient temperature over time);
- data retrieval (for example List current employees, Retrieve aircraft position).

For worked examples see the comprehensive examples in section 3 and the COSMIC Case studies (freely available at cosmic-sizing.org).

1.2.2 Non-functional requirements may evolve into software FUR.

EXAMPLE 1: User Requirements that are not Functional User Requirements include but are not limited to:

- quality constraints (for example usability, reliability, efficiency and portability);
- organizational constraints (for example locations for operation, target hardware and compliance to standards);
- environmental constraints (for example interoperability, security, privacy and safety);
- implementation constraints (for example development language, delivery schedule).’

EXAMPLE 2: A statement of FUR may define that a new system must enable a stockbrokers’ clients to enquire on the value of their portfolio of investments over the Internet. Early in the project, a target response time is specified as a NFR. After further study, it is agreed that the response time requirement can only be met by developing the system to run on a certain hardware platform and also by providing continuous, real-time feeds of stock market prices to the portfolio enquiry system.

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

EXAMPLE 3: A statement of FUR defines a new system and includes the NFR that it must be 'easy to use'. As the project progresses, the development team assumes the company standard graphical user interface (or ‘GUI’) must be built. (A specific statement of the FUR for the GUI may never actually be produced, since the development team knows how to interpret such a requirement. But that is immaterial; if the GUI is provided, then the corresponding FUR can be inferred.) GUI features, such as drop-down lists, are functions of the software available to a user and they

are measurable if they involve movements of data about an object of interest. Software with a GUI may have more functionality, and therefore a bigger functional size, than software lacking such functions.

EXAMPLE 4: A requirement that some input data be batch-processed is a non-functional requirement.

1.2.3 Type versus occurrence.

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: 'Call Centre employee' of which there are 100 occurrences.

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.

1.3 Layers.

EXAMPLE 1: The physical structure of a typical layered software architecture supporting business applications software is given in Figure 1.2:

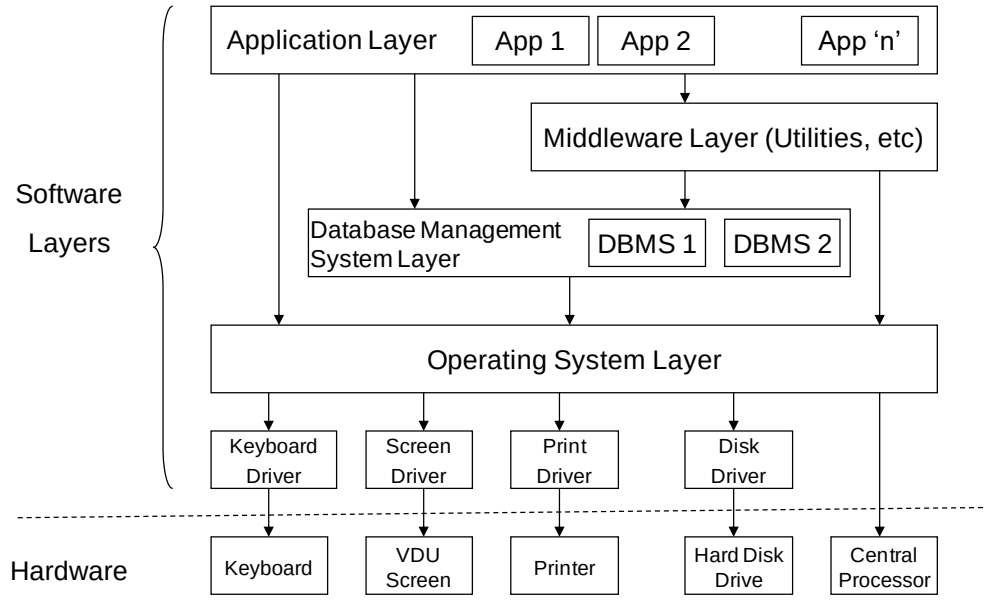


Figure 1.2 - Typical layered software architecture for a Business/MIS system.

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 Figure 1.3). The pieces of software in the application layer of Figure 1.2 are all peers of each other.

EXAMPLE 3: Consider an application A situated in a layered software architecture, as in Figure 1.3, which shows three possible layer structures a), b) and c) according to different architecture ‘views’.

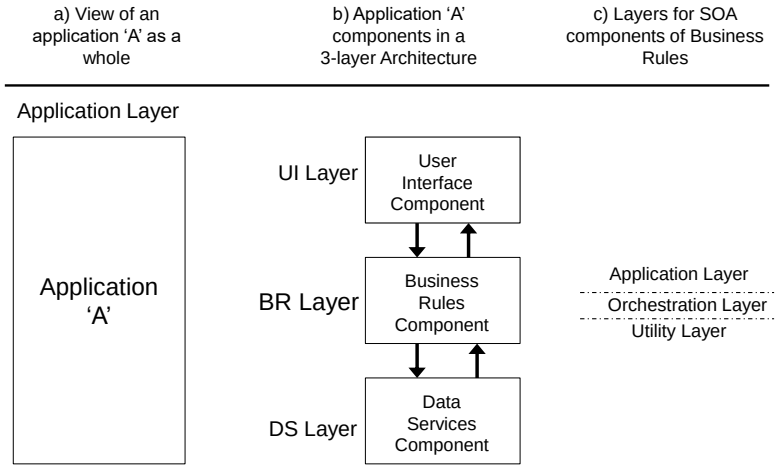


Figure 1.3 - Three views of the layers of an application.

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.)

EXAMPLE 4: A data logging function (for recovery purposes) may be provided as part of the operating system for any application that is set up to use it. An application can use the logging function, but not vice versa. So the correspondence is a hierarchical relationship between them and if the other conditions for layers are met, the logging function is in a different layer to the application.

1.4 Functional users.

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 because the order system FUR are the same for all these employees.

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.

EXAMPLE 3: 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.

1.5 Level of granularity.

EXAMPLE: A group of functional users might be a ‘department’ whose members handle many types of functional processes. 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’.

See for a comprehensive example section 3.3 Different levels of granularity.

1.6 Context diagrams.

EXAMPLE: See Figures 2.5 and 2.8.

2 THE MAPPING PHASE.

2.1 Functional processes.

2.1.1 Identifying functional processes.

EXAMPLE 1: FUR might specify *separate* functional processes to update the data of a person for each of the five possible transitions that he may make between the states of single, married, widowed and divorced because of different needs to add, modify or delete various relationships. Alternatively, the FUR might specify *one* functional process to handle all such changes of status.

EXAMPLE 2: A functional process of an employee 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 employee software who enters the data.

EXAMPLE 3: 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.

EXAMPLE 4 Suppose that on receipt of an order in Example 3, 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.

2.1.2 Different processing paths

EXAMPLE 1: 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.

EXAMPLE 2: Continuing from Example 1, 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.

EXAMPLE 3: 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.

2.1.3 Enquiries.

EXAMPLE 1: Suppose a requirement for an enquiry to display a list of employee names, selected from a file of employee data, by any combination of three input parameters, namely, ‘age’, ‘gender’ and ‘education level’. The three parameters must also be output. If no employees meet the selection criteria, an error message must be issued.

The enquiry can be satisfied by one functional process. The data group moved by the Entry of the functional process may have up to three key data attributes that may occur in seven possible combinations:

- all three parameters together (age, gender, education level);
- any two parameters (age & gender, age & education level, gender & education level);
- a single parameter (age, gender, education level).

It is vital to recognize that these seven possible combinations of the enquiry input parameters all describe one object of interest which could be called ‘the set of employees that satisfies the selection parameters’ (i.e. there is one Entry, not seven Entries.) The main Exit of the enquiry moves a data group that describes the object of interest ‘employee’; it occurs as many times as there are employees that satisfy the selection parameters. The data movements of this functional process are given in the table below, giving a size of 5 CFP.

DM	Key Attributes	Data Group
Entry	Age, Gender code, Education-level	Employee selection parameters

Read	Employee ID	Employee data
Exit	Age, Gender code, Education-level	Employee selection parameters
Exit	Employee ID	Employee name
Exit	Message ID	Error/confirmation message

EXAMPLE 2. When an enquiry functional process that does not seem to need input data is triggered by clicking on a menu button (so this may appear to be a control command), this use of the menu button is an implementation of manually entering selection criteria for the specific enquiry. It provides the triggering Entry for the functional process.

2.1.4 Data entry.

EXAMPLE: A data entry functional process

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

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

Solution for this functional process:

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

The Client ID in the order header is 'the client that places the order'. It is an essential piece of data about the order. We are not entering data about the client. So we do not identify a separate Entry for client. Similarly, the Product ID is an essential piece of data about the order-item, so we do not identify a separate Entry for the Product. Data are entered about only two objects of interest, the Order header and the Order Item. The Reads of the Client and Product data are required to check that the

entered client ID and product ID are valid. The size of this functional process is 7 CFP.

2.1.5 Screen design.

EXAMPLE 1. A single physical transaction screen for entering data can, and often does, provide both the create functional process for the data about an object of interest and the update functions for each stage in the life-cycle of the object of interest, because there is so much common functionality. So depending on the design style, a single physical transaction could account for the implementation of several functional processes. The measurer must examine the FUR to identify a separate functional process for each logically distinct function that is triggered by a separate event.

EXAMPLE 2 Due to the physical limitation of screen sizes, it often occurs that the input or output of a functional process must be spread over more than one screen display. Be sure to ignore the physical screen limitation and inter-screen navigation controls. They do NOT determine where a functional process begins or ends.

2.1.6 Separate functional processes.

EXAMPLE 1: 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).

EXAMPLE 2: 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.

EXAMPLE 3: The task is to update an employee's data record, where the user knows the employee's name, but not the unique employee ID. There may be more than one employee with the same name. The analysis leads to the functional processes FP1, FP2 and FP3 (assuming error/confirmation messages in the FUR). First, in Functional Process 1 ('FP1'), the user is invited to enter the employee's name.

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

Size of FP1 = 4 CFP

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

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

Size of FP2 = 4 CFP

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

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

Size of FP3 = 3 CFP

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

2.1.7 Cardinality of triggering events and functional processes.

EXAMPLE 1: A requirement to display the latest transactions of a bank account ‘on demand’ might be triggered by a) a bank clerk requesting the statement from a terminal at the bank counter on behalf of the account holder and also by b) a clerk in a call center when telephoned by the account holder. Assuming the functional process is identical in both cases and both cases are within the same measurement scope, measure only one functional process.

EXAMPLE 2: When a new employee accepts an offer of employment (a single event) this may lead to one or more functional users initiating multiple functional processes:

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

2.1.8 Drop-down lists.

EXAMPLE 1: The value entered for a data attribute Z of an object of interest A is selected from a drop-down list of values. The values in the drop-down list are either hard-coded, or obtained from a code table and maybe also descriptions. Measure

only the one Entry of the whole data group of which Z is one attribute. The use of a drop-down list does not add to the size.

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

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

For the case where the functional user enters the value of the attribute Z directly (i.e. not via a drop-down list), identify one additional Read (of object of interest B) in the functional process (say 'FP1') for the validation of the value of the attribute Z that has been entered, AND for the alternative case where the user needs to select a value from a list, recognize a *separate functional process* (say 'FP2') for the optional display of the list of values of the attribute. Why a separate FP2? Because the functional user must make a conscious, separate decision to display the valid values of attribute Z.

This separate functional process FP2 should be measured only once for the whole application. There will be as many functional processes of type FP2 in the application as there are attributes for which values may be optionally displayed in this way. Use of this optional alternative is similar to using a help function from any screen. FP2 is

DM	Key Attributes	Data Group
Entry	List ID	Idem, request to display the list
Read	Value of attribute Z	Idem
Exit	Value of attribute Z	Idem, display the value

Total size 3 CFP.

2.1.9 Code tables and their maintenance.

EXAMPLE 1: It is common practice to store attributes of coding systems in tables. Examples are:

- codes and names, e.g. of countries,
- lists of standard names, e.g. accepted credit card suppliers,
- textual clauses, e.g. standard letter clauses,
- the parameters of processing rules, etc.

EXAMPLE 2: Code tables are typically provided with a set of 'CRUD' (Create, Read (i.e. enquiry), Update, Delete) functional processes to maintain the code values, which must be available to the non-business user.

In the simplest possible case, one set of CRUD functional processes might be provided to maintain all codes in one table, where each code-type has just two attributes, namely 'code' and 'description'. For example, to change any one existing code occurrence, the non-business user would have to display the code table contents and page through them to find the particular code occurrence to be

maintained. Effectively there is only one object of interest to the non-business user, namely ‘code’. The Entry could be either a menu selection to display all codes, or the entry of a specific code. In the latter case an error/confirmation message would probably be needed. The enquiry functional process would be:

DM	Key Attributes	Data Group
Entry	Code (or select all codes)	Idem, request to display the code(s)
Read	Code	Code, description
Exit	Code	Idem, display code, description

Total size – 3 CFP (ignoring a possible error/confirmation message)

The C, U and D functional processes would each have size 3 CFP (1 Entry and 1 Write and 1 Exit for an error/confirmation message arising from data validation failure or success). The total size of the set of four processes of the CRUD set is hence 12 CFP, or 13 CFP if an error/confirmation message is needed for an enquiry on a specific code.

EXAMPLE 3: A more realistic situation than described in Example 2 is that special utility software is provided to maintain the codes. Unfortunately, it is impossible to generalize on sizing the functional processes of such software since there are so many potential variations.

EXAMPLE 4: Suppose a case of seven coding systems whose code/description attributes have identical validation needs, so that their values can be maintained by one set of CRUD functional processes. Consider the ‘Create’ functional process. Suppose the non-business user calls up this functional process and must first select from a list the particular coding system that needs new values. He/she can then enter one or more pairs of new code/description attributes. This ‘Create’ functional process has 2 Entries, one for the coding system selection and one for the code/description data entry. It might appear that there should be 7 Writes, but in this case the code/description data group of these seven coding systems have identical validation needs, i.e. are subject to the same FUR so identify one object of interest, there is only 1 Write. The Create functional process is

DM	Key Attributes	Data Group
Entry	Code	Idem, select coding system
Entry	Code	Code, description
Write	Code	Code, description
Exit	Message ID	Error/confirmation message

Total size 4 CFP.

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

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

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

2.1.10 Help functionality

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

2.1.11 Log functionality

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

2.1.12 Batch processing.

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

EXAMPLE. 2 Applications executed in batch processing mode often have a single update functional process containing a Read and then a Write of the same object of interest. The same may apply when a functional process receives input data from another system for immediate update.

EXAMPLE 3. Application A, the software being measured, is required to transmit some persistent data to application B in batch mode (applications A and B are functional users of each other). The functional process of application A that transmits this data has the same data movements regardless of whether the transmission of the data from A to B is in batch mode or whether the data records are sent individually, one at a time. The functional process of application A must have:

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

EXAMPLE 4. Suppose a batch 'job' to produce a standard set of reports, none of which seem to need any external input. The measurer must first decide whether the 'job' consists of one or more functional processes, noting that each functional process in a batch stream must have its own triggering Entry. An example criterion for distinguishing separate functional processes might be that different reports or sets of reports are produced for different types of functional users or are required at different frequency, e.g. weekly versus monthly reports in the same stream. There

must be a good business reason why such a batch stream is divided into more than one functional process.

Each report or set of reports that is considered to be produced by a separate functional process must have one triggering Entry and as many Reads and Exits as are needed to create and output the reports. As in Example 1 the Entry should be analysed as if the selection criteria for the report or set of reports were being entered on-line by the human functional user.

EXAMPLE 5: Often, a single summary report will be produced for the processing of the batch stream. It will normally be found to contain at least one Exit (-type) for each functional process in the stream. If the report shows, for a specific functional process, a count of the number of its occurrences that have been processed, plus a list of the error messages relating to failed occurrences of that functional process, then identify two Exits for that functional process (one for the count and one for the error messages). Then repeat that measurement for each functional process whose summary data may be shown on the report so as to add up the total number of Exits on the report.

EXAMPLE 6: Suppose a business requirement for the application being measured to import some employee data by an interface file from another application in a batch stream. Suppose subsequently, the computer production manager adds a requirement for a general-purpose utility to move any particular version of any file type and to ensure that it is not processed twice. As a result, a standard file header is then added to every interface file in the organization. The file header contains data about the file (file type, file ID, processing date, number of records, hash totals of specific fields, etc.). These data describe an object of interest to the computer production manager called ‘Interface file’.

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

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

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

For processing the header:

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

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

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

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

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

EXAMPLE 7: Suppose as in Figure 2.1 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.

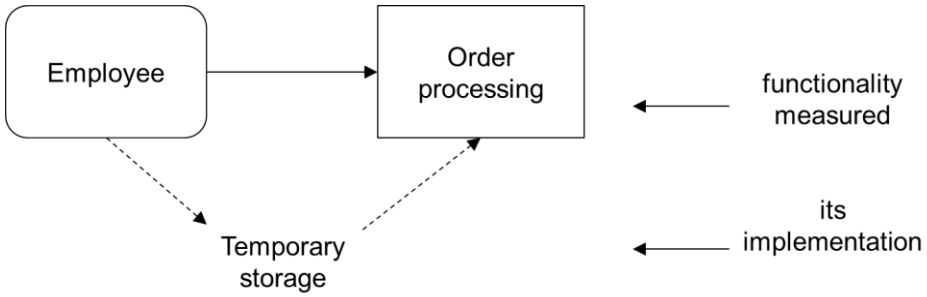


Figure 2.1 – Batch processing with temporarily storage.

EXAMPLE 8: 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 functional 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.

2.2 Data groups and objects of interest.

2.2.1 Common examples.

EXAMPLE 1: 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.

EXAMPLE 2: 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’.

EXAMPLE 3: In the database of a company’s Personnel Information System, ‘employee’ will be an object of interest. The data group which holds data *about* each employee will have attributes such as employee ID, name, address, date of birth, date of employment, etc. Conversely, these data attributes are not objects of interest in this system: we do not want to know any data *about* employee ID, nor about name, address, etc.

2.2.2 Different frequencies of occurrence.

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

EXAMPLE 2: Consider a message to transmit a file of data records to another system, The message consists of a header data group whose attributes describe the file, followed by the data records. The header and the data records have different frequencies of occurrence (one header for multiple occurrences of the data records). Identify the objects of interest ‘header’ and ‘data record’.

EXAMPLE 3: Suppose a requirement for a matrix to record that there is a relationship between certain columns and certain rows, as in a spreadsheet, in order to keep track of which pupils have completed which assignments in a school course. In this example, the ‘pupil/assignment’ relationship is an object of interest. In a paper-based system the relationship would be recorded by entering a ‘tick’ in a chart as each pupil completes each assignment. In a computer system it is sufficient to establish the relationship, i.e. to store the ‘tick’ to create the intersection entity, to indicate that a given pupil has completed a given assignment. Identify the objects of interest ‘pupil’ , ‘assignment’ and ‘completed’.

EXAMPLE 4: Two or more sets of data attributes occurring on the same input or output that are physically separated for aesthetic reasons or for ease of understanding, will belong to the same one data group if they satisfy the frequency rule.

2.2.3 Enquiries and reports.

EXAMPLE 1: A report shows a list of sales in each month for a given product and the total sales for the year of that product. The annual and monthly sales figures have different frequencies of occurrence (12 monthly figures for one annual figure). Identify the objects of interest ‘monthly sales’ and ‘annual sales’.

EXAMPLE 2: Suppose a requirement to enquire on the value of sales to a given customer in a given month, where both the customer ID and the month are entered. The sales amount is calculated from all orders placed by the customer in the month. The functional process is:

DM	Key Attributes	Data Groups
Entry	Customer ID, month	Idem, selection criterium
Read	Customer ID, month	Customer order amount in given month
Exit	Customer ID	Customer ID, customer order amount total in given month
Exit	Message ID	Error/Confirmation Messages

Total size 4 CFP.

EXAMPLE 3: For an enquiry to calculate the sales to a given customer of a given product over a given time period, the input parameters could be labelled ‘Customer ID’, ‘Product ID’, ‘Period start date’ and ‘Period end date’. These are the key attributes of a data group about an object of interest (the ‘goods sold to a given customer of a given product over a given time period’). This is not an enquiry about the persistent data of objects of interest ‘Customer’, or ‘Product’, but about data that is the result of an intersection of these two and the given time-period.

EXAMPLE 4. Suppose a database in which each customer is allocated a ‘customer-category’, (where the latter is coded P = Personal, R = Retailer or W = Wholesaler). There are FUR to develop an enquiry to calculate and display:

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

The start and end dates of the time-period must be entered and must also be output to enable understanding. There is no requirement for an error or confirmation message. The data needed for this enquiry will be taken from a database of completed single-item orders, where each order could have attributes, e.g. order ID, product ID, customer ID, customer-category code, order value, and date payment received. This date is the criterion for deciding if the product has been sold. Figure 2.2 shows an example output from this enquiry:

Sales by Customer-category

Period: Jan 01 2017 to Mar 31 2017

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

Figure 2.2 – Sales by customer-category for the given time period

Discussing first the output of this enquiry, two levels of aggregation of sales data can be distinguished. The sales per customer-category and the total sales have different frequencies of occurrence. This indicates that we have data describing two objects of interest and hence two Exits. The two objects of interest are:

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

As a cross-check on this conclusion, 'the goods sold to a customer of any one category in a time-period' is a different 'thing' from 'the goods sold to all customers in the time-period'. In this example, both 'things' are really different (physical) 'things' in the world of the functional user … about which the software is required to move a data group either to or from a functional user, or to or from persistent storage, as per the definition of an object of interest. (The 'goods sold' are real 'things', i.e. physical products that left a warehouse).

Discussing next the input to this enquiry – the time-period – this is a data group moved by the triggering Entry of the enquiry. This data group describes an object of interest that we could call 'the time-period of the enquiry'. Data defining the time-period must also be output so that the user can understand the sales values. The time-period must also be counted as a separate Exit when it is output because of differing frequency of occurrence of this data group.

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

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

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

Total size – 4 CFP.

Comment on the analysis:

- Applying Rule 11 (the 'frequency rule') tells us that we must have two Exits as the frequency of occurrence (and the key attributes) of 'Sales to all Customers' differ relative to 'Sales per Customer-category'.
- The key attributes of the 'Time-period definition' data group on the output (Start date' and 'End Date') and the total 'Sales to all Customers in the Time-period' ($2,921K), both occur once on the report and have the same key identifying attributes (Start date and End date). The frequency rule tells us that they are both attributes of the same one data group, hence there is one Exit to move these attributes.

- The output heading ‘Sales by Customer-category’ is fixed text; it is not counted as an Exit – see section 2.8.

2.2.4 Checking the size of complex output.

EXAMPLE: Applying the last paragraph of section 3.3.2 on identifying data groups in Part 2, suppose the two sales figures of Example 4 ‘Sales per Customer-category in the Time-period’ and ‘Sales to all Customers in the Time-period’ were required to be output by two separate functional processes on separate reports, instead of on the same one report as in Figure 2.2.

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

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

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

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

We observe that the two different data groups shown as output on the report of Figure 2.2 from a single functional process become three different data groups when reported separately by two functional processes. (The three data groups have different attributes and the two functional processes output a total of three Exits). However, there are only two unique sets of key attribute combinations, regardless of whether the data are reported separately by two functional processes or together by one functional process on the one report.

2.2.5 Data attributes.

EXAMPLE 1: 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.

EXAMPLE 2: ‘Country’ may be a secondary object of interest, or not an object of interest at all to a manufacturing company, but would be a primary object of interest to a an international transport company. ‘Currency’ and ‘Currency exchange rates’ will be primary for a bank and either secondary or of no interest at all to the applications of a city administration. ‘Department’ may be primary for an application that maintains an organizational structure but only an attribute for the asset register.

EXAMPLE 3: In the input to a simple ‘create employee’ functional process, an attribute may be labelled ‘grade’ when what is really meant is ‘employee grade’. This is an attribute of the inbound data group about an employee; it would be incorrect to assume that this indicates a separate data group ‘grade’. There is only one Entry ‘employee data’ in this simple case.

(However, this case may not be so simple. All the considerations of whether ‘grade’ might be another object of interest for these FUR, with its own attributes, e.g. ‘grade salary’, apply as in the examples in section 2.1.8 on drop-down lists and in section 2.3.4 on ‘data attribute or object of interest’).

2.2.6 Object of interest sub-types.

EXAMPLE: Sometimes, separate objects of interest should be recognized as sub-types of a particular object of interest. A sub-type of an object of interest inherits all the attributes of the object of interest but also has its own unique attributes.

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

FUR 1: Suppose the object of interest ‘Customer’ has an attribute that indicates its category as ‘Personal’, ‘Retailer’, or ‘Wholesaler’. If the rules governing the processing of the data of all customers are the same, and all customers have the same data attributes, then these three categories are NOT regarded as sub-types of Customer. ‘Customer-category’ is simply one attribute of Customer amongst many.

FUR 2: Alternatively suppose the following FUR.

- ‘For the data stored about Customers, all customers shall have the attributes: Customer ID, Customer Name, Address, Telephone no., Customer-category.
- ‘Customer-category’ has three values P = Personal, R = Retailer, W = Wholesaler.
- Personal customers have no additional attributes.
- Retailer customers have additional attributes: Retailer Sales Region, Credit limit, Last annual turnover, Retailer price discount scale.
- Wholesaler customers have additional attributes: Account manager, Credit limit, Last annual turnover, Wholesaler price discount scale, Wholesaler payment terms’.

Given that Customers have some common attributes, but also have different attributes according to their Customer-category, it follows that Personal, Retailer and Wholesaler are sub-types of Customer. Logically, therefore, the object of interest ‘Customer’ has three sub-type objects of interest (‘Personal Customer’, ‘Retailer Customer’ and ‘Wholesale Customer’). The model of the OOI and its sub-types is shown in Figure 2.3.

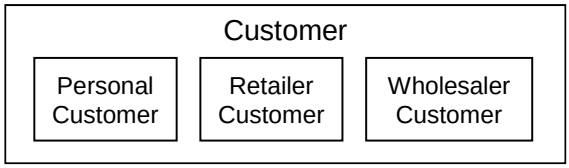


Figure 2.3 – Object of interest with its sub-types

We now introduce various additional FUR for different types of functional processes that will illustrate the effects of needing to reference these object of interest types and sub-types. We consider only how the Entries, Reads and Writes vary due to the effect of the three sub-types on the FUR.

FUR 3. ‘A function is required to produce name-and-address labels for all Customers’. A functional process would Read only the object of interest ‘Customer’.

FUR 4. ‘A function is required that enables a user to enter orders for a Personal Customer.’ A functional process would need only one Read of the object of interest ‘Personal Customer’ in order to validate the existence of the particular Personal Customer placing the order.

FUR 5. ‘A single function is required to enable a user to enter orders for any Retailer or Wholesaler Customer.’ A functional process would need to refer to (i.e. Read) the three separate objects of interest when validating the entered data, namely:

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

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

FUR 6. ‘A single function is required to enter data about a new customer of any type.’ A functional process would need to have separate Entries and Writes for each of the three sub-types Personal, Retailer, Wholesaler as well as a Read of Customer to check that the Customer does not already exist, i.e. a total of seven CFP for these sub-processes.

2.3 Data movements.

2.3.1 Common examples.

EXAMPLE 1: 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.

EXAMPLE 2: 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.

EXAMPLE 3: 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.

EXAMPLE 4: A process is required for a human user to enter and store data about a new employee. The dialogue proceeds as follows:

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

- The software validates the entered data for format errors, etc., generates a unique employee ID, stores the new employee record and confirms that the process has been completed by displaying a screen for entry of the next new employee's data.

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

Human user or software actions	**Data crossing the boundary**	**Comment**	**DM Measured**
User selects 'Enter data for new Employee'' from the menu	Message to software: 'Display screen for new employee data entry'	Menu selection is not measured	-
Software displays formatted data entry screen	Text field headings for entry of data for a new employee	Ignore 'blank' data entry screens. Only the entered data are measured	-
User enters the new employee title and Social Security Number (SSN).	New employee name New employee SSN	These are just the first two new employee attributes of the triggering Entry; processing of this Entry is not complete yet.	Entry (Employee)
Software issues an error message if the SSN is invalid	Error message, e.g. 'Invalid SSN'	Ignore the user's repeated attempts to enter a valid SSN.	Exit (Error/Conf. message)
Software searches the file of existing employees to find if the new employee name and SSN already exist	-	-	Read (Existing Employee)
Software issues a warning message if an existing employee is found with the same name and SSN	Warning message (e.g. 'Employee already exists in system')	This is another occurrence of an error/confirmation message. It has already been measured. Ignore.	-
If an existing employee is found with the same name and SSN, the software displays their details,	Existing employee ID, title, name, address, SSN, date of birth, sex, marital status	This output enables the user to check that this employee's data really has already been entered.	Exit (Existing Employee)
User closes screen showing data for existing Employee. Software re-displays menu	'Close screen' command (in). Menu displayed (out)	Closing the screen is a 'control command'. Ignore.	-
User enters the other new employee attributes.	New employee title, address, date of	The system validates the entered data for field	-

	birth, sex, marital status,	formats, etc. (Data manipulation)	
Software may issue other validation error messages	Example: 'Error: address postcode is invalid'	More occurrences of an error/confirmation message Ignore.	-
Software generates a unique Employee ID	-	Data manipulation	-
User presses 'Enter' when all employee attributes have been validated	'Enter' command	A control command. (Think of this as completion of the Triggering Entry)	-
Software makes the employee record persistent	-	-	Write (Employee)
Software signals successful completion by displaying a screen for entering the next new employee data	Text field headings for entry of data for a new employee	Another occurrence of an error/confirmation message. Ignore.	-

The functional process is:

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

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

2.3.2 Control commands, data unrelated to an object of interest.

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.

EXAMPLE 2: Do not identify data movements for moving control commands, e.g. functions to :

- display/not display a header, (sub-) totals that have been calculated, or display values in ascending or descending order,
- navigate up and down and between physical screens, e.g. via 'page up' and 'page down', or by scrolling, by navigation buttons, display a blank screen for data entry, or 'close screen',
- hit a Tab or Enter key, or pressing a button to continue.

- click ‘OK’ to acknowledge an error message or to confirm some entered data, etc. use menu commands and links to web pages that enable the user to navigate to one or more specific functional processes but which do not themselves initiate any one functional process.

EXAMPLE 3: 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.

2.3.3 Data movement uniqueness.

EXAMPLE 1: If the same data group is moved to two physical devices or to two destinations, only one Exit is identified. But if the data manipulation of the Exits to the two devices or destinations differ significantly (beyond the completely trivial differences, i.e. that cause some extra analysis or design), two Exits are identified.

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

EXAMPLE 3: As per Example 2. The printed output contains the same attributes as the screen display, but the printed layout is different: two Exits are identified. An example would be where a data group is displayed on a screen in graphical form but is printed in numerical form. The data manipulation and formatting differ for the two presentations of this data.

EXAMPLE 4: A file of one or more outbound data groups must be distributed by a business application to more than one destination (= functional user) and the FUR of the application require differences in processing for these outbound data groups (i.e. resulting in different data manipulations, and different attributes in the Exits) to the different functional users. One Exit per object of interest is identified for each functional user for which different processing is required. If identical data groups are sent to different functional users, identify one Exit.

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.

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’.

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 and a file with newly-defined attributes describing the same object of interest . Identify two Reads, one for each file, for the functional process.

2.3.4 Data attribute or object of interest.

EXAMPLE 1: Suppose an order-processing application for a manufacturer that supplies lighting goods in bulk to individuals and companies. The application enables order-desk staff to enter and store a lot of data about customers, e.g. customer-ID, customer-name, customer-address, customer-contact telephone, customer-credit-limit, customer-category-code, date-of-last-order, etc. If there is either an error in the entered data or the data have been processed abnormally, the order-desk staff must be notified via an error/confirmation message.

In this system, 'customer' is clearly an object of interest to many functional users, e.g. to the order-desk staff. So the simplest functional process to enter data about a customer would be measured at 4 CFP, as below.

DM	Key Attributes	Data Group
Entry	Customer ID	Customer data
Read	Customer ID	Customer data
Write	Customer ID	Customer data
Exit	Message ID	Error/confirmation messages.

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

EXAMPLE 2: Notice the attribute 'customer-category-code' in the above list. It can have several code values, e.g. P, R, W, etc., standing for Personal, Retailer, Wholesaler, etc. respectively. Suppose when entering the value of this attribute, the order-desk user is presented with a drop-down list showing the permitted descriptions for 'customer-category'. The user then selects the appropriate description and the corresponding customer-category-code is entered and stored as an attribute of customer. The descriptions are provided only for human interpretation. There is only a requirement to store customer-category-code as an attribute of 'customer'. There is no requirement to store data about customer-category in the customer record, so customer-category is not an object of interest to the order-desk user. There is still only one Entry for this functional process; the presence of the list of customer-category descriptions does not affect the size of the functional process, which is still 4 CFP, as in Example 1.

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

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

However, supposing customer codes and descriptions are stored in maintainable tables, the functionality needed for such code table maintenance would have

functional processes to create new customer-category codes and descriptions, and to update, delete and read them. In other words, they process data about customer-category. Customer-category is thus an object of interest for the non-business user in these functional processes. For these functional processes, therefore, any movement of data (E, X, R, W) about customer-category in these functional processes must be identified. (Examples of the functional processes that might be needed to maintain parameter tables are given in section 2.1.9 Code tables and their maintenance)..

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

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

In this Example 3, therefore, when entering data about a new customer, the entered customer-category-code must be validated against the already-stored corresponding attribute of the object of interest 'customer-category'. So the simplest functional process to enter data about a new customer, as described in Example 1, must now in this Example 3, include a Read and an Exit to display the customer-category codes and maybe other attributes of the customer-category object of interest, so that the user can select the correct code.

A GUI drop-down list that enables an order-desk user to select a customer-category description might be identical for all Examples 1, 2 and 3. But in Examples 1 and 2, this list is related to the 'customer' object of interest, whilst in Example 3, the list is related to the 'customer-category' object of interest.

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

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

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

2.3.5 Date and time.

EXAMPLE 1. If FUR require the attributes date or time to be output, normally business applications obtain these parameters from the operating system, no data movement should be measured for this functionality.

EXAMPLE 2: 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.

EXAMPLE 3: For an account of how timer functionality may work and may be measured, see Part 3b: Real-time examples.

2.3.6 Validating input data.

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

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

EXAMPLE 3. If as in Example 2 the customer names must be displayed for the user to select the customer that is placing the order, then measure one Exit for the display of the customer names.

EXAMPLE 4. When entering a post-code (or Zip-code) as part of an address, suppose the entered code must be checked against a list of valid codes available via a service that is outside the scope of the software being measured. Measure one Exit/Entry pair to account for the data movements needed to validate the entered post-code.

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

EXAMPLE: See Figure 2.4. 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.

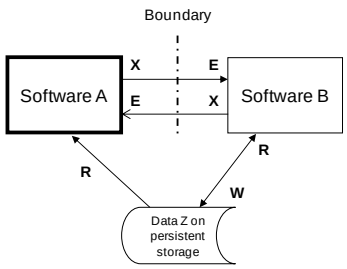


Figure 2.4 – Persistent data Z within the boundary of both software A and B for a Read.

2.4 Measuring the components of a distributed software system.

2.4.1 A client-server application.

EXAMPLE: Suppose a simple two-component, client-server application. Component A executes on a PC front-end that communicates with component B on a main-frame computer holding some data describing one object of interest. The (human) functional user triggers a functional process on the PC that requires this data to be read from the main-frame and displayed on the PC. (The following analysis ignores possible error/confirmation messages.

Case a) Measurement scope is the whole application.

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

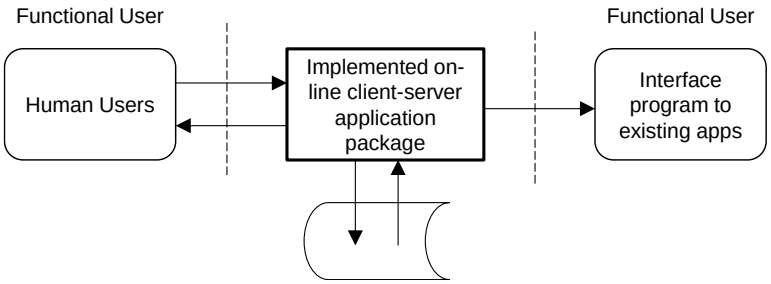


Figure 2.5 - Context diagram for the client-server application.

The data movements of the functional process to obtain the required data are then as shown in the Message Sequence Diagram of Figure 2.6 (total 3 CFP):

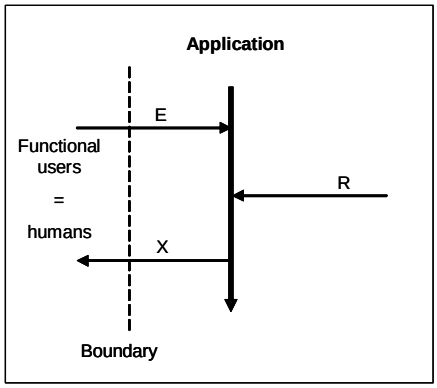


Figure 2.6 – Measurement scope is the whole application

Case b) Measurement scope is component A

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

Figure 2.7 shows the data movements that component A needs in order to obtain the required data from component B. Component A must issue a ‘request to obtain’ (or ‘get’ command) with the data selection criteria as an Exit to component B and receive back the required data from component B as an Entry. (Component A has no knowledge of how component B obtains the requested data; it could be via a Read, or by local calculation, or from some other software.)

Component B has become another functional user of component A, in addition to component A’s (human) functional users. The data movements of case b) are therefore as shown in Figure 2.7 (4 CFP).

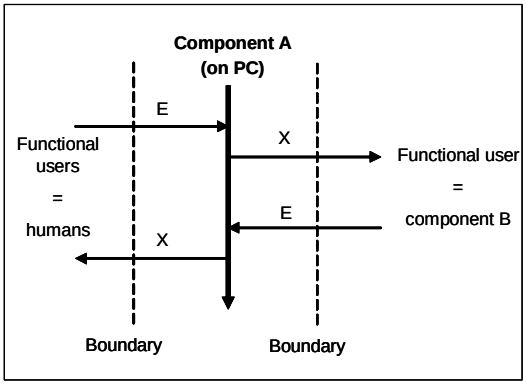


Figure 2.7 – Measurement scope is component A

Case c) Measurement scope is component B.

Defining the scope as ‘component B’ reveals that component B’s functional user is now the PC application software component A, where the triggering event of a request-to-obtain-data occurs. Note that the components A and B are functional users of each other; their exchange of data takes place across a mutual boundary. The data movements of component B are shown in Figure 2.8.

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

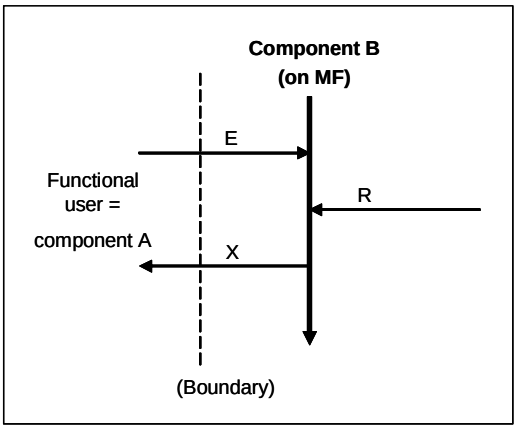


Figure 2.8 - Measurement scope is component B

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

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

As another cross-check on this conclusion, and following the ‘aggregation rules’ in Part 2 Guidance on Rule 23 the size of the application ignoring the physical split into components should be:

- the total size obtained by adding up the size of each component measured separately, i.e. 7 CFP)
- less (the size of the inter-component data movements, i.e. 4 CFP)

resulting in 3 CFP, as in case (a).

2.4.2 Both components are client and server.

EXAMPLE: Now suppose an application with two components distributed over separate technical platforms as in the Example of section 2.4.1, where each component may trigger functional processes in the other component. An example might be an application that has component A on a laptop that enables field-sales staff to enter data over the internet to another component B on a server. In addition, the server can trigger functional processes to send data back to the laptop, independently of the functional processes on component A.

Case a) Measurement scope is the whole application.

This case now differs from the Example a) in section 2.4.1. The application has two functional users, the human (e.g. field sales staff) functional users of the laptop and ‘something’ (e.g. a clock) that triggers the transmission of data from the server to the laptops.

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

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

Case b) Measurement scope is component A

Figure 2.9 shows the context diagram for Cases b) and c)

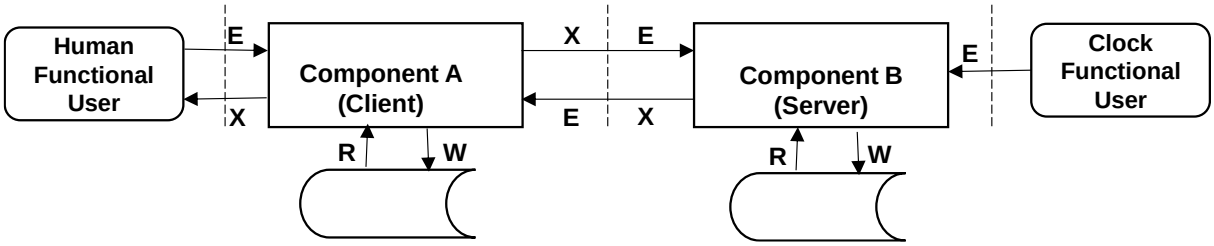


Figure 2.9 – Context diagram for cases b) and c).

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

Case c) Measurement scope is component B

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

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

2.4.3 Asynchronous communications*.*

EXAMPLE: Suppose in Figure 2.10 (b) the component A is in a hotel reservation system and component B is in the system of a credit card supplier. When a hotel customer wants to reserve a hotel room, the credit card system may not be available or may be heavily loaded. In order not to keep the hotel customer waiting, the communication between the two systems takes place asynchronously.

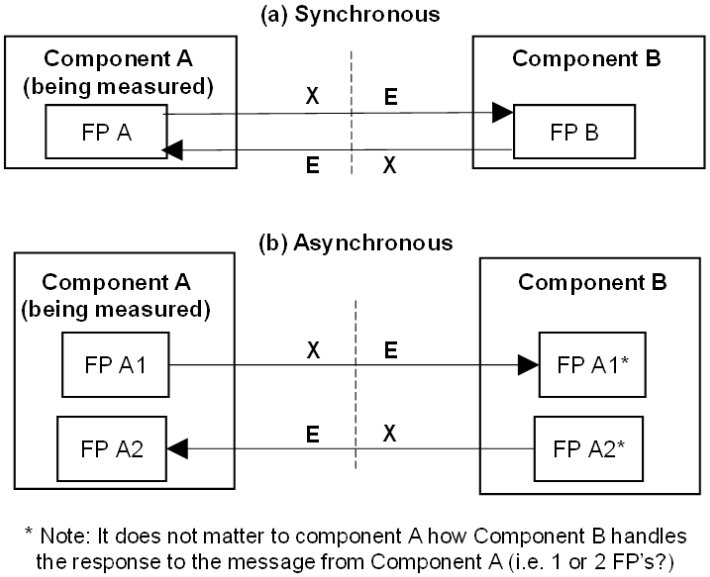


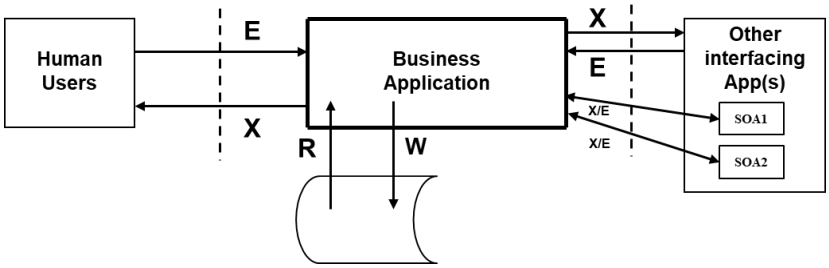
Figure 2.10 - COSMIC functional model of synchronous and asynchronous communications

2.5 Re-use and the FUR.

EXAMPLE 1: If a business application uses (reusable) components, for example SOA services, a measurement of the business application may exclude or include the components:

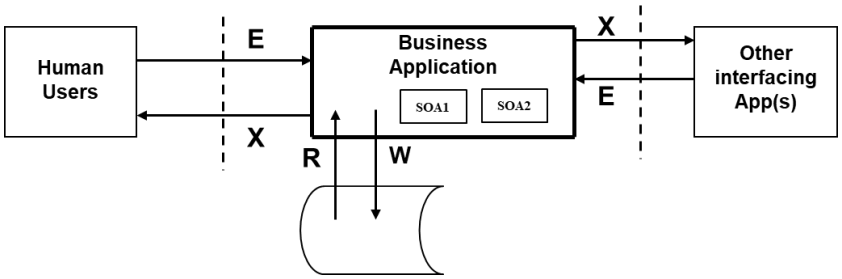
A. When the reusable components are *outside the scope* of the software being measured:

- these reusable components are then functional users of the business application and in Figure 2.11a they belong to the ‘Other interfacing Apps’:
  - the functionality that these reusable components provide will not be sized,
  - i.e. only the data movements to-from each of the re-usable components will be included in the ‘business application’ scope.



2.11a - Business application, components outside the scope.

B. When the reusable components are *within the scope* of the software being measured:
   - the functionality of these reusable components is then (part of the) functionality of the business application, as in Figure 2.11b:
     - the movements of data within the business application to/from these reusable components should be ignored: in the context diagram of Figure 2.11b, the SOA1 and SOA2 reusable components are neither functional users or persistent storage.



2.11b - Business application, components within the scope.

Note. The reuse of components should be recorded and taken into account for estimating and benchmark comparisons.

EXAMPLE 2: 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.

EXAMPLE 3: An application has FUR for two functional processes FP1 and FP2. Functional process FP1 enables the functional user to maintain a ‘credit-worthiness indicator’ for any customer, that has three values (levels) ‘excellent’, ‘normal’ and poor’. Functional process FP2 enables a customer account to be debited or credited. The FUR state that if the value of a debit using functional process FP2 causes the account balance to become negative, then the existing credit-worthiness indicator must be automatically reduced by one level. The developer sees an opportunity to

avoid duplicate coding of the same functionality. He implements functional process FP1 according to its FUR and then implements functional process FP2 using the part of the functionality of functional process FP1 that deals with setting the credit-worthiness indicator. Two functional processes are identified:

C. functional process FP1 to maintain the credit-worthiness indicator;

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

2.6 Error/confirmation messages

2.6.1 Messages without an object of interest

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 do not move data about an object of interest and should be considered as occurrences of one Exit in each functional process where such messages occur (‘error/confirmation messages’).

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.

2.6.2 Messages with an object of interest

EXAMPLE 1: 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.

EXAMPLE 2: In an order-entry functional process, for the automatic production of a letter when an order is not accepted due to a credit-check failure, identify an Exit for each object of interest about which data is found in the letter.

2.6.3 Messages which are not specified in the FUR.

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

2.7 Data manipulation

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.

EXAMPLE 2: The FUR state that the amount field on an input screen may only accept not-negative values and must display amounts from $1000,- in bold. The validation and presentation is provided by the data manipulation of the Entry data movement concerned.

EXAMPLE 3: The FUR state that amounts on an output screen must be displayed in descending order. This presentation is provided by the data manipulation of the Exit data movement(s) concerned.

2.8 Fixed text and other fixed information.

CONVENTION : 'Fixed text' is text that:

E. is available to the software being measured, either hard-coded in the software, or available from persistent storage or from other software;

F. may be selected but not changed by the normal functional user. (It may be maintainable by a system administrator).

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

EXAMPLE: 'Other fixed information' includes diagrams, logo's, photos, audio or video output.

2.8.1 Fixed information that is output on request.

CONVENTION. Each function that may be triggered independently to output fixed text should be treated as a separate functional process. If this category of fixed text is maintainable via functional processes (where the fixed text is maintainable, it should be considered as 'codes' of the software being measured) – see section 2.1.9.

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

DM	Key Attributes	Data Group
Entry	Fixed text ID	Idem
Read	Fixed text ID	Fixed text ID, fixed text
Exit	Fixed text ID	Fixed text

Total size 3 CFP

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

EXAMPLE 3: 'Help' text is fixed text. Another form of Help information could be an explanatory video. See section 2.1.10 for how to analyze and measure Help functionality.

EXAMPLE 4: Menus are fixed text.

EXAMPLE 5: Error and confirmation messages are fixed text. See section 2.6 for how to analyze and measure error and confirmation messages.

2.8.2 Fixed information that is output 'automatically'.

CONVENTION : All these types of fixed text should be ignored unless they must be changed – see section 2.9.3.

Note: enquiry output and report headings may contain data describing an identifiable object of interest. For an example, see Figure 3.3 i), where the heading contains the year of the 'time-period definition' data group. Such variable data should obviously not be ignored.

EXAMPLE: Fixed text and fixed information that is output automatically include

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

2.9 Measurement of changes.

2.9.1 Changes to data.

EXAMPLE 1: The object of interest 'Employee' contains 'number of dependents' as an attribute. It is decided to store more data about dependents. Consequently, an object of interest 'Dependent' is added and linked to Employee. Data about individual dependents must now be included in the 'create employee' input, and the attribute 'number of dependents' is removed from the Employee object of interest.

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

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

The 'create employee' functional process is

DM	Key Attributes	Data Group
Entry	Employee ID	Employee data
Read	Employee ID	Employee data (To check if the Employee already exists)

Write	Employee ID	Employee data
Exit	Errors	Error/confirmation messages

Total size 4 CFP

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

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

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

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

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

The size of the functional change to the 'create employee' functional process is 2 Entries + 2 Writes + 1 Exit = 5 CFP. Probably many more functional processes that must deal with the modified or added data groups must also be functionally changed. The changes to these functional processes must also be measured. Also, there may be new or modified error/confirmation message occurrences. If so, the data manipulation associated with the data movement will be changed.

EXAMPLE 2: 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 3: A bank statement shows the interest payable each month on positive balances. The algorithm to calculate the interest must be modified in some detail although no change is needed for the input data for the interest calculation. The bank statement is unchanged in content and layout but the data manipulation associated with one attribute is modified. The functional change is measured as 1 CFP for the modified Exit that shows the monthly interest.

EXAMPLE 4: 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 5: 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.

2.9.2 Sizing a deletion or addition of a part of an application.

Note the difference between *size of the FUR* and *size of the change to the FUR*.

Example 1:
If a piece of software of size 100 CFP has been removed from an application, the size of the FUR has been decreased by 100 CFP and:

- if this piece of software consists of 2 functional processes of 50 CFP then the size of the change to the FUR is 2 CFP.
- if the piece of software consists of 20 functional processes of 5 CFP the size of the FUR still decreases by 100 CFP but the size of the change to the FUR is 20 CFP.

Example 2:

If the same piece of software of size 100 CFP has been added, the size of the FUR increases by 100 CFP and the sizes of the changes to the FUR are still:

- 2 CFP when the FUR has 2 functional processes,
- 20 CFP when the FUR has 20 functional processes.

2.9.3 Changes to fixed text.

CONVENTION: If a change is required to a data group containing ‘Fixed text or other fixed information that is output on request’ (see section 2.8), then the Entries, Exits, Read and Writes that move the data group should be measured as changed according to the normal rules. The fixed text of field headings is measured differently from the headings of input or output screens or reports and application-general data.

EXAMPLE 1: Normally, fixed text field headings on input or output only need to change if the data associated with the headings must be changed. As any required change to the data accounts for any changes to the associated headings, the latter should be ignored. It is the change to the data that must be measured.

EXAMPLE 2: Any required change to the fixed text heading of an input or output screen, or to a report heading, or to application-general data should be ignored, i.e. it should not be measured.

EXAMPLE 3: Presentation of an attribute concerns its font, background color, field length, field heading, number of decimal places, etc. If there is a requirement to change the presentation of an attribute, including its field heading appearing on input or output, when there has been no change to the associated data (this might be needed, for example, to improve ease of understanding), then one Entry or one Exit may be counted for any input or output data group respectively in which one or more field headings must be changed.

EXAMPLE 4. A company decides to add one first name (between parentheses) to the initials of the employee registration. The initials field is enlarged to accommodate the additional first name, and the field heading is changed from ‘Initials’ to ‘Initials (first name)’. As the change to the initials field in the Entry, Exit, Read and Write data movements accounts for any changes to the associated headings, the latter should be ignored.

EXAMPLE 5. A company’s employee registration has an existing initials field in which, following the initials, between parentheses one first name may be added. As it appeared that this possibility often was overlooked it has been decided to change the field heading from ‘Initials’ to ‘Initials (first name)’. As the presentation of the attribute has changed, without a change to the associated data, an Entry or Exit may be counted for any input or output data group respectively in which the field headings must be changed.

EXAMPLE 6. After a merger of companies A and B the new company’s name is C, which will replace the names on all screens of the former companies A and B. This is a change to application-general data and should not be measured.

EXAMPLE 7: 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.

2.9.4 Other changes.

EXAMPLE 1: When the screen color for all screens is changed, this change should not be measured.

3 COMPREHENSIVE EXAMPLES.

3.1 Data movements in enquiries.

3.1.1 Common enquiries.

EXAMPLE 1: Simple enquiry

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

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

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

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

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

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

EXAMPLE 2: More complex enquiry

The FUR of Example 2 is identical to that of Example 1 but with the addition ‘also, output the Client name and address with the Client ID for each Client that placed an order in the period.’

The solution for this functional process is now:

DM	Key Attributes	Data Group / (Data Attributes)
Entry	Start date, End date	Time-period definition
Read	Order ID	Order data (Order ID, client ID, etc.)
Read	Client ID	Client data (Client ID, client name, address, etc.)
Exit	Start date, end date	Time-period definition
Exit	Start date, End date, Client ID	Client order_address data (Client ID, client name, address, number of orders)

In this example it is now necessary to read the Client object of interest in order to obtain the client name and address, so there are now two Reads. However, as in Example 1, we still have only two Exits. Adding the Client name and address attributes to the ‘Client order data’ data group of Example 1 does not alter the fact that there is still only one movement of data describing a Client in Example 2, hence one Exit.

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

EXAMPLE 3: More complex enquiry

The FUR of Example 3 is identical to that of Example 2, but the functional process must allow up to 3 different time-periods to be entered and the output must show the number of orders placed by the client in each period.

The report could be laid out in many ways. First, it could be laid out as in Example 2, with three blocks, one for each of the three time periods:

Period 1 start and end dates

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

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

Client C, etc.

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

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

Client ID, client name, client address

Period 1 start and end dates, # orders.

Period 2 start and end dates, # orders.

Period 3 start and end dates, # orders.

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

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

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

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

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

For all three possible layouts, the data group ‘Time-period definition’ now has up to three occurrences and the ‘Client’ data group has one occurrence for every client that placed an order in at least one of the time-periods. But applying the frequency rule tells us that the functional process outputs two data groups (distinguished by differing frequencies of occurrence and different key attributes), regardless of the report layouts. Hence the functional process of Example 3 is still the same size of 5 CFP as that of Example 2, ignoring possible error/confirmation messages.

It is important to note that all three report layouts convey the same information, even though the physical groupings of data differ. In consequence, the physical distribution of data attributes on a report cannot be relied upon to determine the data groups to which they belong. Use the frequency rule to decide on the data groups.

EXAMPLE 4: Simple enquiry with entered condition

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

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

A set, and a member of that set, are not the same ‘thing’.

EXAMPLE 5: If there were also a requirement (FUR2) to output the age limit in addition to the requirement of FUR1, then there would be an extra Exit because 'age limit' is an attribute of the object of interest: 'the set of all employees that satisfy the search parameter' of the enquiry. The two Exits have different frequencies of occurrence. The analysis would be as below.

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

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

EXAMPLE 6: If there were now a further requirement (FUR3) to output the total number of employees that satisfy the age-limit criterion in addition to the age limit (of FUR2), this would be a second attribute of the same object of interest ('the set of all employees that satisfy the search parameter') that already has 'age limit' as an attribute. The size of the functional process satisfying FUR1 + FUR2 + FUR3 would be unchanged at 4 CFP.

3.1.2 Multi-stage enquiry

EXAMPLE 1. A set of FUR state: 'Client data must be displayed after entering a client name. If there is only one client with the entered name, all the client details are shown immediately. If there are more clients with the same name, the software must show a list of the clients with this name, plus sufficient client data (e.g. their address) to distinguish them – referred to as 'client summary data'. The user may then select the right client and the software will show its details.'

Solution: two functional processes are identified. The first functional process produces the client details for the entered name or alternatively the list of clients with that name, plus their address, i.e. the client summary data (see the Guidance on Rule 11 in Part 2 of the Measurement Manual).

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

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

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

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

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

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

3.2 Data movements in reports.

3.2.1 Report with multi-level aggregations.

EXAMPLE: Consider the following set of FUR:

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

A report is required that lists all company employees by name, sorted by division and by department-within-division. The report should also show sub-totals of the number of employees for each department and for each division, and the total number of employees for the whole company. The report can be initiated at any time from a menu of functions available to Personnel staff. Today's date must also be output; we assume date and time can be obtained from the operating system, i.e. it does not have to be input or Read (see section 2.3.5 for how to measure this.)

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

DM	Key Attributes	Data Group (Data attributes)
Entry	Report request ID	Report-type selection (This is implied when selecting the menu item)
Read	Employee ID	Employee data
Read	Department ID	Department data (includes Division ID for the Department)
Exit	Employee ID	Employee name (grouped by department within division)
Exit	Today's date, Department ID	Department details (Department ID, Department employee subtotal at today's date)
Exit	Today's date, Division	Division details (Division name, Division

	ID	employee subtotal at today's date)
Exit	Today's date	Company details (Today's date, Company employee total)

In this example, one functional process is needed to produce the report, which must be triggered by an Entry and which must have two Read data movements, of the 'Employee' and of the 'Department' objects of interest, to obtain the data it needs from persistent storage. ('Department' must be an object of interest to the functional users of this application since it effectively defines the company structure.)

The four data groups that are output all have different frequencies of occurrence (and different key attributes), so must be distinguished according to the Guidance on Rule 11 in Part 2 of the Measurement Manual. There are four Exits.

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

In total, therefore, this functional process has 1 Entry, 2 Reads and 4 Exits, i.e. its size is 7 CFP (ignoring the possible need for an error/confirmation message, which is not stated in the FUR).

3.2.2 Report with two-dimensional multi-level aggregations

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

Figure 3.1 shows the relationships between the various objects of interest that make up the company's product structure and how multi-item Orders relate to a Product-type. (The 'crows-foot' symbol in Figure 3.1 illustrates the degree of the relationship between the various objects of interest.)

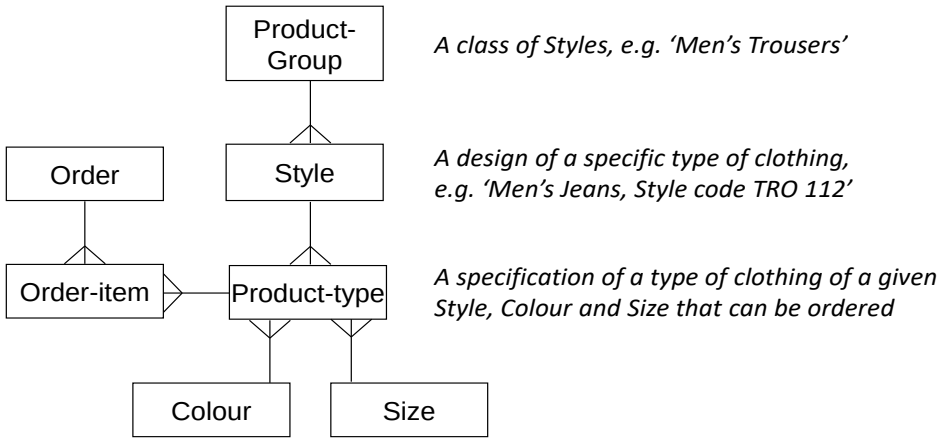


Figure 3.1 –The company's product structure and its product-types

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

J. A Product-Group is identified by a 3-alpha Product-Group code, e.g. TRO (for trousers), SHI (for shirts), etc. Product-group has no other attributes.

K. Each Style is uniquely identified by its 3-alpha Product-Group code and by a 3-digit Style Number which is unique within that Product-Group. Together, these form a 6-character 'Style code' key. Examples of Style codes are:

- SHI049 (where '049' indicates this is the 49th shirt design)
- TRO112 (where '112' indicates this is the 112th trouser design).

L. A Style has other attributes such as Style name, Style description, Fabric code, Designer, Date of introduction, Manufacturing cost, etc.

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

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

M. Each Order has as attributes – Order ID, Customer ID, Date of the sale, ID of salesman responsible, etc.

N. An order can be placed for multiple Product-types, each in a separate 'Order-item'. Each Order-item has as attributes: Order-item number, Style Code, Colour name, Size code, quantity ordered, sale price.

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

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

Figure 3.2 shows an example '2-dimensional' sales report, which is abbreviated for convenience, namely:

O. Sales of the whole Product Range appear in the bottom row, by month and for the whole Time-period;

P. the Product Range comprises several Product Groups, but Figure 3.2 shows rows for only two occurrences: Shirts and Trousers;

Q. Each Product Group has several Styles, but Figure 3.2 shows rows for only three occurrences of Styles for each Product Group;

R. The total Time-period of the report spans nine months, but Figure 3.2 shows columns for only four occurrences of months, as well as the final column for total sales in the whole Time-period. Note that the year associated with any month is found in the report heading. (For example the column headed 'July' is really 'July 2016', a month/year combination.)

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

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

Figure 3.2 – Clothing Company's sales report

We start to analyze the report by examining the sales amount data attributes at six levels of aggregation (three levels of aggregation on the product dimension by two levels of aggregation on the time dimension). These six levels are color-coded in Figure 3.2 to help distinguish them. Each sales amount is a single-attribute data group (-type). The six groups are as follows, with their key attributes:

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

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

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

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

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

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

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

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

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

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

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

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

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

It is now interesting to demonstrate the applicability of the last paragraph of section 3.3.2 on identifying data groups in Part 2. Figure 3.3 shows the reports that would be needed if, for example, the sales amounts i) and v) in the list above were required to be output by separate functional processes.

i) Shirt Style Monthly Sales (2015/16)

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

v) Product-Group Sales

Period: July 2016 - March 2017

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

Figure 3.3 – Separate reports for two of the six Sales objects of interest

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

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

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

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

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

If we were to apply this same analysis to find the unique (or unique combinations of) identifying key attributes for all six sales amounts, i to vi, if they were output by six separate functional processes, the count of unique (or unique combinations of) identifying key attributes for all the Exits output by the six processes would be the same, **nine** in total, including the error message as when the same data are output on the one report of Figure 3.3.

3.2.3 Report data in graphical form.

EXAMPLE : Measurement of output data does not change if the data are presented graphically rather than in table form. Figure 3.4 shows the expenses of a company's departments in 2014 and 2015.

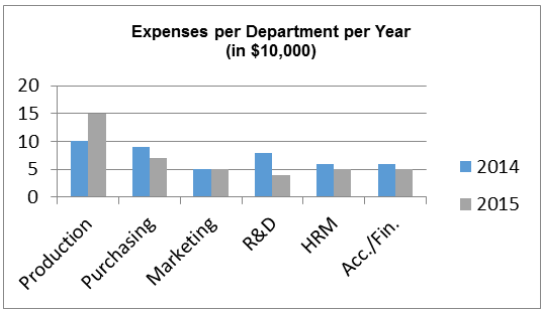


Figure 3.4 – Data in graphical form

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

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

The graph title explains the meaning of the amounts; it is fixed text that should not be measured (see section 2.8).

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

3.3 Different levels of granularity

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 3.5 (a).

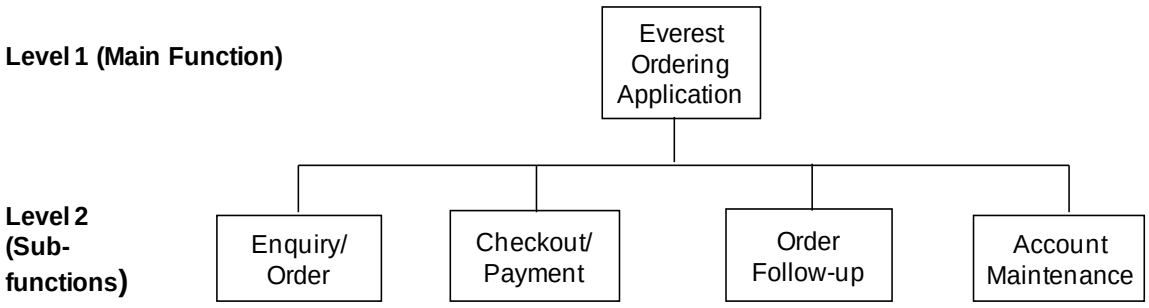


Figure 3.5 (a) - Analysis of the Everest Ordering System: the first two levels of granularity.

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 3.5 (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.

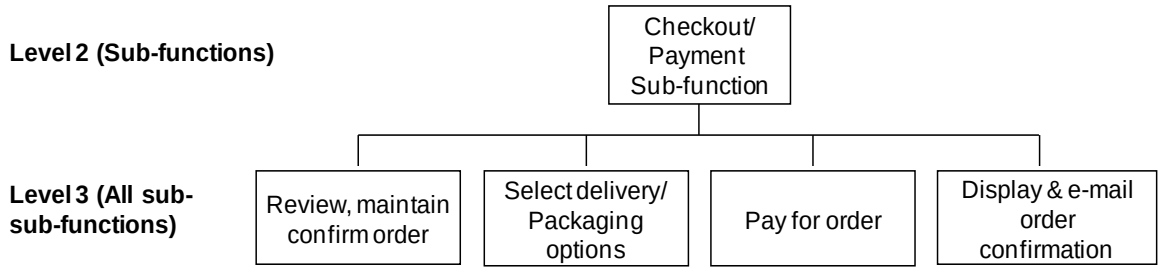


Figure 3.5 (b) - Analysis of the Checkout/Payment Sub-function.

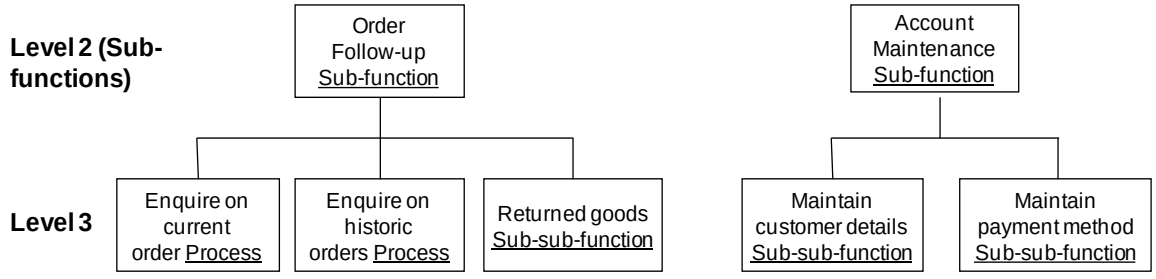


Figure 3.5 (c) - Analysis of the Order Follow-up and Account Maintenance Sub-function.

Figure 3.5 (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[1] at level 3 (for two enquiries of the Order follow-up sub-function, each one corresponding with a triggering event). 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 3.5 (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.

3.4 Measuring an expert system

See the revised version, the Expert System Measurement Case Study in the Knowledge Base.