COSMIC Measurement Manual for ISO 19761

Combination of Parts Version 5.0 August 2021

Part 1: Principles, Definitions & Rules

Part 2: Guidelines

Part 3a: Standard Measurement Strategy Examples

Part 3b: Real-time Examples

Part 3c: MIS Examples

Part 3d: Standard. Reqs with Big Data Cleaning Examples

(Issued Nov. 2022 with unchanged copies of the existing Parts)

COSMIC Measurement Manual for ISO 19761

Part 1: Principles, Definitions & Rules

Version 5.0

August 2021

Foreword

Much of modern society depends on software and the importance of regulated software measurement has never been greater. Software functional sizing is a reliable and consistent means of estimating, planning and benchmarking software work.

The COSMIC measurement method is the modern evolved standardized way of measuring functional size of software. It evolved in the late 1990's, as a rework of the earlier function point sizing methods and it was endorsed as an ISO engineering standard in 2003.

COSMIC sizing is applicable to sizing all types of software. The methodology is equally suited to modern agile and scaled agile approaches to software development as traditional approaches.

The Measurement Manual

The COSMIC Measurement Manual describes the core measurement methodology. This version 5.0 shortens version 4.0.2 while not modifying its substance in terms of measurement definitions, rules and guidance. This manual consists of three 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)

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

Further explanations, guidelines, translations, practical examples and other publications are available from 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).

August 2021 minor editing:

In the Foreword the names of the Parts have been updated. No other changes.

Table of Contents

1. INTRODUCTION.

1.1 Purpose.

The purpose of this document is to state the Principles, Definitions and Rules of the COSMIC Functional Size Measurement method (the 'COSMIC Method'), as well as the COSMIC measurement process.

This Part 1 of the COSMIC Measurement Manual document contains only reference material, i.e. ***what*** to do, as described in ISO 19761 [3]. For guidance developed by the COSMIC Group as to ***how*** to apply COSMIC to different situations refer to Part 2, and for **examples**, refer to Part 3. The COSMIC Group has also published additional documents to illustrate its use in various contexts contexts (Agile, Business Applications, Real-time, etc.) and technologies (SOA, Mobile, etc.)

1.2 Overview.

The COSMIC method involves applying a set of models, principles, rules and processes to measure the Functional User Requirements (or 'FUR') of a given piece of software.

The result is a numerical 'value of a quantity' (as defined by ISO) representing the functional size of the piece of software according to the COSMIC method. This numerical value is on a ratio scale: therefore valid mathematical operations can be performed using the values.

The COSMIC method adopts the ISO definition of Functional User Requirements (FUR).

1.3 The models and principles of the COSMIC method.

The COSMIC method is based on software engineering principles categorized in two models:

The COSMIC Software Context Model: it contains the principles that pertain to identifying the nature and structure of the Software to be measured as required by the COSMIC method, leading to the identification of its FUR.

The Generic Software Model: it contains the principles to be applied to the FUR in order to extract and measure the elements that contribute to the functional size using the COSMIC method.

PRINCIPLES - The COSMIC Software Context Model.

1. A software application is typically structured into layers.
2. A layer may contain one or more separate pieces of software.
3. A piece of software is described by its Functional User Requirements (FUR).
4. The FUR are expressed at a level of granularity that exposes its functional processes.
5. A piece of software delivers functionality to its functional users as identified in the FUR.
6. A piece of software to be measured is defined by the scope of the functional size measurement (FSM), which is confined wholly within a single layer.
7. The scope of the FSM defines the functional processes to be measured, and depends on the purpose of the measurement.

PRINCIPLES - The COSMIC Generic Software Model.

1. A piece of software interacts with its functional users across a boundary, and with persistent storage within this boundary.
2. A functional process consists of sub-processes called data movements.
3. There are four data movement sub-types: Entry, Exit, Write and Read. A data movement sub-type includes any associated data manipulation.
4. A data movement moves a single data group.
5. A data group consists of a unique set of data attributes that describe a single object of interest.
6. Each functional process is initiated by a triggering event, detected by a Functional User and which in turn initiates a data movement called the triggering Entry.
7. The functional size is based on the types of the elements used for measurement, not on the number of occurrences.
8. The size of a functional process is equal to the number of its data movements where one data movement has a size of 1 COSMIC Function Point.
9. The size of a piece of software is the sum of the sizes of the functional processes within the scope of the FSM.

NOTE: The principles are written using the terminology of the COSMIC method.

1.4 Definitions.

In this sub-section:

- the definitions from ISO documents are reproduced 'as is' but without the ISO NOTES that have been transferred to the main body of the text;
- texts underlined refer to terms defined in this sub-section.

application.
software system for collecting, saving, processing, and presenting data by means of a computer. [Adapted from [4]]

Base Functional Component.
(BFC).
elementary unit of Functional User Requirements defined by and used by an FSM method for measurement purposes. [1]

boundary.
conceptual interface between the software being measured and its functional users.

component.
any part of a software system that is separate for reasons of the software architecture, and/or that was specified, designed or developed separately.

control command.
command that enables human functional users to control their use of the software but which does not involve any movement of data about an object of interest of the FUR of the software being measured.

COSMIC unit of measurement.
1 CFP (COSMIC Function Point), which is defined as the size of one data movement.

data attribute.
smallest parcel of information, within an identified data group, carrying a meaning from the perspective of the software's Functional User Requirements. [3]

data group.
data group type.
distinct, non-empty, non-ordered and non redundant set of data attributes where each included data attribute describes a complementary aspect of the same one object of interest. [3]

data manipulation.
any processing of the data other than a movement of the data into or out of a functional process, or between a functional process and persistent storage. [3]

data movement.
data movement type.
Base Functional Component which moves a single data group. [3]

E - Abbreviation for Entry type.
Entry.
Entry type.
data movement that moves a data group from a functional user across the boundary into the functional process where it is required. [3]

error/confirmation message.
Exit issued by a functional process to a human user that either confirms only that entered data has been accepted, or only that there is an error in the entered data.

Exit.
Exit type.
data movement that moves a data group from a functional process across the boundary to the functional user that requires it. [3]

event.
something that happens.

functional process.
functional process type.

elementary component of a set of Functional User Requirements, comprising a unique, cohesive and independently executable set of data movements. [3]

functional process level of granularity.
level of granularity of the description of a piece of software at which

- the functional user (-types) are individual humans or engineered devices or pieces of software (and not any groups of these) AND
- single event (-types) occur that the piece of software must respond to (and not any level of granularity at which groups of events are defined).

functional size.
size of the software derived by quantifying the Functional User Requirements. [1]

Functional Size Measurement
FSM.
process of measuring Functional Size. [1]

Functional Size Measurement method.
FSM method
specific implementation of FSM defined by a set of rules, which conforms to the mandatory features of ISO/IEC 14143-1:2007. [1]

functional user.
user that is a sender and/or an intended recipient of data in the Functional User Requirements of a piece of software. [3]

Functional User Requirements.
FUR.
sub-set of the user requirements describing what the software shall do, in terms of tasks and services. [1]

input.
data moved by all the Entries of a given functional process.

layer.
partition resulting from the functional division of a software system. [3]

level of decomposition.
any level resulting from dividing a piece of software into components (named 'Level 1', for example), then from dividing components into sub-components ('Level 2'), then from dividing sub-components into sub-sub components (Level 3'), etc.

level of granularity.
any level of expansion of the description of any part of a single piece of software (e.g. a statement of its requirements, or a description of the structure of the piece of software) such that at each increased level of expansion, the description of the functionality of the piece of software is at an increased and uniform level of detail.

measurement method
logical sequence of operations, describe generically, used in the performance of measurements. [5]

measurement process.
process of establishing, planning, performing and evaluating software measurement within an overall project or organizational measurement structure. [3]

measurement (strategy) patterns.
standard template that may be applied when measuring a piece of software from a given software functional domain, that defines the types of functional user that may interact with the software, the level of decomposition of the software and the types of data movements that the software may handle.

model.
description or analogy used to help visualize a concept that cannot be directly observed.

OOI - Abbreviation for object of interest
object of interest.
object of interest type.
any 'thing' that is identified from the point of view of the Functional User Requirements about which the software is required to process and/or store data. [3]

output.
data moved by all the Exits of a given functional process.

peer software.
piece of software that reside in the same layer as, and exchanges data with, another piece of software. [3]

persistent storage.
storage which enables a functional process to store data beyond the life of the functional process and/or which enables a functional process to retrieve data stored by another functional process, or stored by an earlier occurrence of the same functional process, or stored by some other process. [3]

purpose of measurement.
statement that defines why a measurement is being made, and what the result will be used for.

R – Abbreviation for Read type.
Read.
Read type.
data movement that moves a data group from persistent storage within reach of the functional process which requires it. [3]

scope.
scope of the FSM.
set of Functional User Requirements to be included in a specific functional size measurement instance. [3]

software
set of computer instructions, data, procedures and maybe documentation operating as a whole, to fulfill a specific set of purposes, all of which can be described from a functional perspective through a finite set of Functional User Requirements, technical and quality requirements.

software system.
system that consists only of software.

sub-process type.
part of a functional process that either moves data (into the software from a functional user or out of the software to a functional user, or to or from persistent storage) or that manipulates data.

system.
combination of hardware, software and manual procedures organized to achieve stated purposes. [adapted from [2]]

triggering Entry.
triggering Entry type.
the Entry data movement of a functional process that moves a data group generated by a functional user that the functional process needs to start processing.

triggering Event.
triggering event type.
event that causes a functional user of the piece of software to initiate ('trigger') one or more functional processes. [3]

unit of measurement.

particular quantity, defined and adopted by convention, with which other quantities of the same kind are compared in order to express their magnitudes relative to that quantity. [5]

user
any person or thing that communicates or interacts with the software at any time. [3]

value (of a quantity)
magnitude of a particular quantity, generally expressed as a unit of measurement multiplied by a number.

W - Abbreviation for Write type.
Write.
Write type.
data movement that moves a data group from a functional process to persistent storage. [3]

X – Abbreviation for Exit type.
See Exit.

2. COSMIC MEASUREMENT PROCESS.

The COSMIC measurement process consists of three phases – see Figure 2.1:

- the Measurement Strategy phase, in which the purpose and scope of the FSM are defined. The Software Context Model is then applied so that the software to be measured and the required measurement are unambiguously defined – see Section 3.
- the Mapping Phase in which the Generic Software Model is applied to the FUR of the software to be measured to produce the COSMIC model of the software that can be measured – see Section 4.
- the Measurement Phase, in which actual sizes are assigned - see Section 5.

Rules for how measurements should be recorded are given in Section 6.

The relationship of the three phases of the COSMIC method is shown in Figure 2.1.

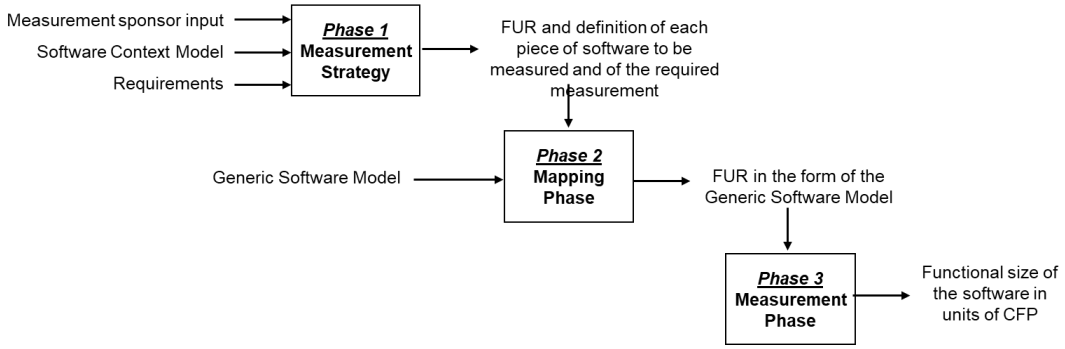


Figure 2.1 – The COSMIC method measurement process.

3. MEASUREMENT STRATEGY PHASE.

3.1 Deriving the measurement strategy from the software context model.

This section describes the key parameters that must be considered in the Measurement Strategy phase before actually starting to measure. The sub-sections give the rules to help the Measurer through the process of determining a measurement strategy, as shown in Figure 3.1.

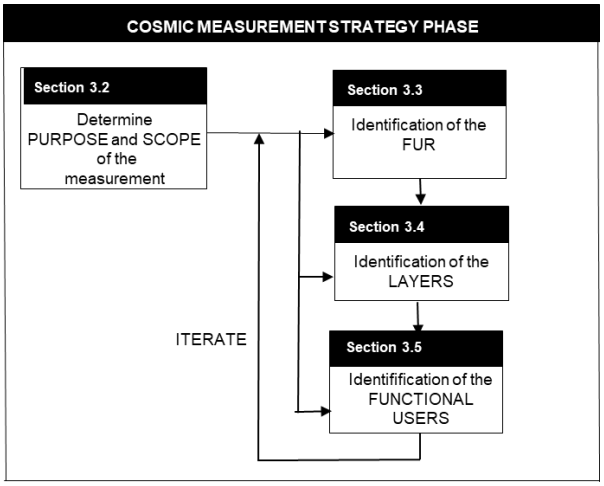


Figure 3.1 - The process of determining a Measurement Strategy.

RULE 1: Measurement activities.

The determination of the COSMIC functional size **shall** involve all of the activities and rules described in Sections 3.2 to 3.6.

3.2 Determination of the purpose and scope of the FSM.

RULE 2: Purpose and scope.

The purpose and the scope of the FSM **shall** be determined before commencing a particular measurement exercise.

NOTE: Once the purpose of the FSM has been determined, the process of determining the scope(s) of the FSM, the functional users, the layers and the boundaries may require some iterations.

3.3 Identification of the FUR.

RULE 3: Identification of the FUR.

The FUR identified to be within the scope of the FSM **shall** be used as the exclusive source from which the functional size of the software is to be measured.

NOTE: The term 'FUR' means those functional user requirements that are completely defined so that a COSMIC functional size measurement is possible.

3.4 Identification of the layers.

3.4.1 The scope of the FSM and layers

Software may have components of its functionality that exist in different layers of the software's operating environment.

RULE 4: If required for the purpose of the measurement exercise, each such layer **shall** be identified.

RULE 5: A single piece of software to be measured **shall not** have its scope defined to extend over more than one layer.

NOTE 1: FUR may state explicitly, may imply, or the measurer may infer, that the FUR apply to software in different layers or to different peer items whose size must be measured separately. Alternatively, the measurement analyst may be faced with sizing existing software which appears to be in different layers or to consist of separate peer items. In both cases, guidance is needed to help decide if the FUR of the software comprise one or more layers or peer items.

NOTE 2: Layer identification is an iterative activity. The exact identification of layers can be refined as the measurement activity progresses.

3.4.2 Characteristics of layers

RULE 6: Characteristics of layers.

Layers identified within the scope of the FSM **shall** have the following characteristics:

a) Software in each layer shall deliver functionality to its functional users.

b) Software in a subordinate layer shall provide functional services to software in a layer using its services.

c) Software that shares data with other software shall not be considered to be in different layers if they identically interpret the data attributes that they share.

3.5 Identification of the functional users.

RULE 7: Functional Users.

All functional users that trigger, provide information to, or receive information from functional processes in the FUR of the software within the scope of the FSM **shall** be identified.

NOTE 1: This rule above corrects an omission in ISO 19761. An amendment to the standard is in preparation.

NOTE 2: As persistent storage is on the software side of the boundary, it is not considered to be a functional user of the software being measured.

3.6 Identification of software boundaries

RULE – Identification of boundaries.

RULE 8: The boundary of each piece of software within each layer and in the scope of the FSM **shall** be identified.

RULE 9: Once the boundaries have been identified, each FUR within the scope of the FSM shall be allocated to a piece of software.

4. MAPPING PHASE.

4.1 General – Mapping the FUR to the Generic Software Model.

Figure 4.1 shows the steps of the process for mapping the FUR as in the available software artefacts to the form required by the COSMIC Generic Software Model.

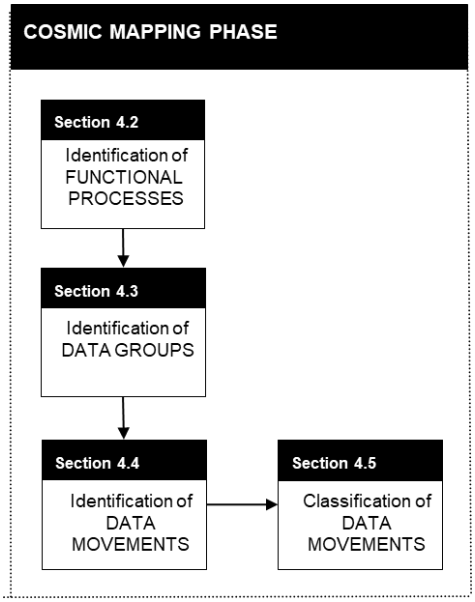


Figure 4.1 –The process of the COSMIC Mapping Phase.

4.2 Identification of functional processes.

RULE 10: Identification of functional processes.

Each functional process identified in the scope of the FSM **shall**:

a) be derived from at least one identifiable FUR,

b) be initiated by an Entry data movement from a functional user informing the functional process that it has detected a triggering event,

c) comprise at least two data movements, namely always one Entry plus either an Exit or a Write.

d) belongs to one, and only one layer,

e) be complete when a point of asynchronous timing is required to be reached according to its FUR.

NOTE 1: the COSMIC Group has subsequently clarified sub-clause e) above as the equivalent of the following statement: 'the set of all data movements that is needed to meet its FUR for all the possible responses to its triggering Entry'.

NOTE 2: The Generic Software Model is a logical model. A functional process occurrence may start processing before data has been entered e.g. when a human user clicks on a menu to display a blank screed for data entry.

NOTE 3: In a set of FUR, each event which causes a functional user to trigger a functional process

- cannot be sub-divided for that set of FUR,
- has either happened or it has not happened.

4.3 Identification of objects of interest and data groups.

RULE 11: Identification of objects of interest and data groups.

Each data group identified in the scope of the FSM **shall**:

a) be unique and distinguishable through its unique collection of data attributes,

b) be directly related to one object of interest described in the software's FUR.

NOTE 1: An object of interest can be any physical thing, as well as any conceptual thing or part of a conceptual thing in the world of a functional user.

NOTE 2: Examples of 'thing' include, but are not limited to, software applications, humans, sensors, or other hardware.

NOTE 3: The term object of interest is used in order to avoid terms related to specific software engineering methods. The term does not imply objects in the sense used in Object Oriented methods. Similarly, the word Entity is avoided because of its use in Data Modelling.

NOTE 4: Constants or variables which are internal to the functional process, or intermediate results in a calculation, or data stored by a functional process resulting only form the implementation, rather than the FUR, are not data groups.

4.4 Identification of data movements.

This step consists in identifying the data movements (Entry, Exit, Read and Write) of each functional process. Figure 4.2 illustrates the overall relationship between the four types of data movement, the functional process to which they belong and the boundary of the measured software.

RULE 12: Identification of data movements.

Each functional process identified in 4.2 **shall** be partitioned into its component data movements.

NOTE: The COSMIC method defines a data movement type as a BFC.

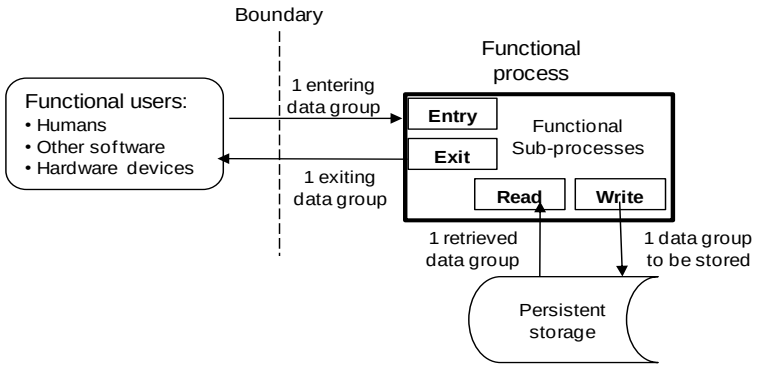


Figure 4.2 – The four types of data movement & their relationship with a functional process.

RULE 13: Functional Process – Single Entry.

For any one functional process, a single Entry data movement **shall** be identified and counted for the entry of all data describing a single object of interest that the FUR require to be entered, unless the FUR explicitly require data describing the same single object of interest to be entered more than once in the same functional process.

RULE 14: Functional Process – Single Exit, Read or Write.

Similarly, a single Exit, Read or Write data movement **shall** be identified and counted for the movement of all data describing a single object of interest that the FUR requires of that type (e.g. Exit, Read or Write, respectively), unless the FUR explicitly require data describing the same singe object of interest to be moved more than once in the same functional process by a data movement of the same type (e.g. Exit, Read or Write, respectively).

RULE 15: Functional Process – Occurrences.

If a data movement of a particular type (Entry, Exit, Read or Write) occurs multiple times with different data values when a functional process is executed, only one data movement of that type **shall** be identified and counted in that functional process.

4.5 Classification of data movements.

RULE 16: Entry.

An Entry **shall:**

a) receive a single data group which originates from the functional user side of the boundary,

b) account for all required formatting and presentation manipulations along with all associated validations of the entered data attributes, to the extent that these data manipulations do not involve another type of data movement.

NOTE: An Entry accounts for all manipulations that might be required to validate some entered codes or to obtain some associated descriptions. However, if one of more Reads are required as part of the validation process, these are identified and counted as separate Read data movements.

c) include any ‘request to receive the Entry data’ functionality, where it is unnecessary to specify what data should be entered.

RULE 17: Exit.

An Exit **shall:**

a) send data attributes from a single data group to the functional user side of the boundary,

b) account for all required data formatting and presentation manipulations, including processing required to send the data attributes to the functional user, to the extent that these manipulations do not involve another type of data movement.

RULE 18: Read.

A Read **shall:**

a) retrieve a single data group from persistent storage.

b) account for all logical processing and/or mathematical computation needed to read the data, to the extent that these manipulations do not involve another type of data movement,

c) include any ‘request to read’ functionality.

RULE 19: Write.

A Write **shall:**

a) move data attributes from a single data group to persistent storage.

b) account for all logical processing and/or mathematical computation to create the data attributes to be written, to the extent that these manipulations do not involve another type of data movement.

RULE 20: Write – Delete.

A requirement to delete a data group from persistent storage shall be a single Write data movement.

5. MEASUREMENT PHASE.

5.1 The measurement phase process.

The general method for measuring a piece of software when its FUR have been expressed in terms of the COSMIC Generic Software Model is summarized in Figure 5.1.

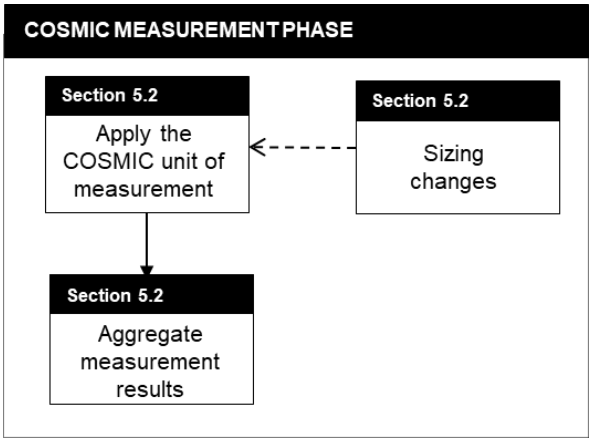


Figure 5.1 – The process of the COSMIC Measurement Phase.

5.2 Calculation of the functional size.

RULE 21: Size of a data movement.

A unit of measurement, 1 CFP[1], **shall** be assigned to each data movement (Entry, Exit, Read or Write) identified in each functional process.

RULE 22: Size of a functional process.

The results of 5.2, as applied to all identified data movements within the identified functional process, **shall** be aggregated into a single functional size value for that functional process by:

a) multiplying the number of data movements of each type by its unit size,
b) totaling the sizes from step a) for each of the data movement types in the functional process.

Size (functional process) = Σ size(Entries) + Σ size(Exits) + Σ size(Reads) + Σ size(Writes).

RULE 23: Functional size of the identified FUR of each piece of software to be measured.

The size of each piece of software to be measured within a layer **shall** be obtained by aggregating the size of the functional processes within the identified FUR for each piece of software.

NOTE: Within each identified layer, the aggregation function is fully scalable. Therefore a sub-total can be generated for individual functional processes, individual software pieces or for the whole layer, depending on the purpose and scope of the FSM.

RULE 24: Functional size of changes to the FUR.

Within each identified layer, the functional size of changes to the FUR within each piece of software within the scope of the FSM **shall** be calculated by aggregating the sizes of the corresponding impact data movement according to the following formula:

[1]The unit of measurement was known as a 'Cfsu' (COSMIC functional size unit) prior to v3.0 of COSMIC method.

Size (Changes to a piece of software) = Σ size (added data movements) +

Σ size (changed data movements) +

Σ size (deleted data movements)

summed over all functional processes for the piece of software.

NOTE: A data movement is considered to be changed if any of the attributes of the data group are changed, or if any changes are needed to the data manipulation associated with the data movement.

6. MEASUREMENT REPORTING.

The result must be reported and data about the measurement recorded so as to ensure that the result is always unambiguously interpretable.

RULE 25: Labelling.

A COSMIC measurement result on the FUR for a piece of software that conforms to the mandatory rules of ISO 19761 **shall** be labelled according to the following convention:

CFP (ISO/IEC 19761:2011).

RULE 26: Documentation of the measurement results.

The documentation of the COSMIC measurement results **shall** include the following information:

a) Identification of each piece of software in the scope of the FSM (name, version identification or configuration identification).

b) A description of the measurement purpose and scope.

c) A description of the relationship of each piece of software within the scope of the FSM with its functional users, both peer-to-peer and between layers.

d) The functional size of each piece of software within the scope of the FSM, calculated according to 5.2 and reported according to 6.

NOTE: Documentation is required for each piece of software measured within each layer.

REFERENCES.

[1] ISO/IEC 14143-1: 2007 Information technology – Software measurement – Functional size measurement – Part 1: Definition of concepts, International Organization for Standardization – ISO, Geneva, 2017.

[2] ISO/IEC 15288: 2008 Systems and Software Engineering – System Life Cycle Processes, International Organization for Standardization – ISO, Geneva, 2008.

[3] ISO/IEC 19761: 2011 Software engineering – COSMIC: a functional size measurement method, International Organization for Standardization – ISO, Geneva, 2011.

[4] ISO/IEC 24570:2005 Software engineering -- NESMA functional size measurement method version 2.1, International Organization for Standardization – ISO, Geneva, 2010.

[5] ISO Guide 99: 1993, International vocabulary of basic and general terms in metrology (VIM), International Organization for Standardization – ISO, Geneva, 2019.

COSMIC Measurement Manual for ISO 19761

Part 2: Guidelines

Version 5.0

August 2021

Foreword.

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

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

This Part 2 of the COSMIC Measurement Manual presents the set of Guidelines developed by the COSMIC Group to facilitate accuracy, repeatability and reproducibility of COSMIC measurement results.

August 2021 minor editing:

In the Foreword the names of the Parts have been updated. No other changes.

This COSMIC measurement manual is based on ISO/IEC standard 19761:2011, reconfirmed in 2019. This version 5.0 of the COSMIC Measurement Manual replaces version 4.0.2: it shortens version 4.0.2 while not modifying its substance in terms of measurement definitions, rules and guidance.

A public domain version of the COSMIC Measurement Manual and other technical reports, including translations into other languages can be found in 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. INTRODUCTION.

1.1 Purpose of this document.

The purpose of this document is to provide guidance to the practitioners for the application of the COSMIC ISO 19761 Function Points standard presented in Part 1.

This Part 2 is complementary to Part 1 and is supplemented by Part 3 which presents a large number of examples.

1.2. The COSMIC measurement process.

The COSMIC measurement process presented in Part 1 is reproduced in Figure 1.1:

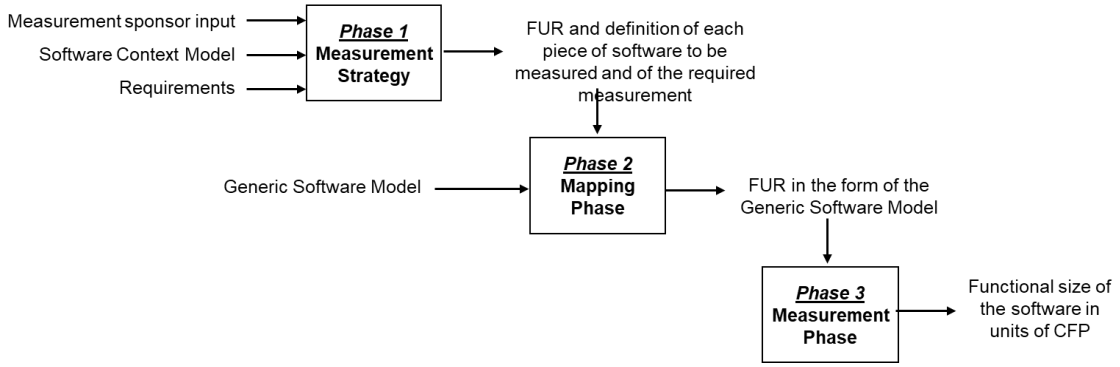


Figure 1.1 – The COSMIC method measurement process.

1.3. Types and occurrences.

In general, the 'type' of a thing is an abstract class of all the things that share some common characteristic, so that the things are subject to the same FUR. (Synonyms of 'type': 'category', 'kind'). An 'occurrence' of a thing is when it becomes real by assigning values to its attributes. Known in the Object-Oriented world as 'instantiation' – the creation of an instance.'

2. THE MEASUREMENT STRATEGY PHASE.

2.1 Overview of the measurement strategy phase.

Figure 2.1 presents graphically the Measurement Strategy Phase.

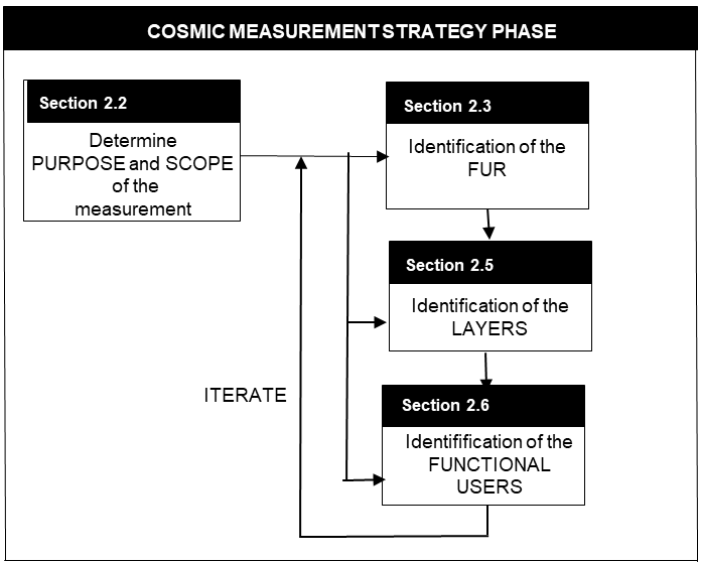


Figure 2.1 - The process of determining a Measurement Strategy.

2.2 Determine purpose and scope of the FSM.

The term ‘purpose’ is used in its normal English meaning. The purpose helps the measurer to determine:

- The scope of the measurement and hence the artefacts which will be needed for the measurement.
- The functional users.
- The functional changes.
- The point in time in the project life-cycle when the measurement will take place.
- The required accuracy of the measurement, and hence whether the standard COSMIC method should be used, or whether an approximation technique should be used (e.g. early in a project’s life-cycle, before the FUR are fully elaborated).

As an aid to determining a measurement strategy, the Guideline for 'Measurement Strategy Patterns' describes, for each of several different types of software, a standard set of parameters for measuring software sizes, called a ‘measurement strategy pattern’ (abbreviated to ‘measurement pattern’. Consistent use of the same measurement patterns should help measurers to ensure that measurements made for the same purpose are made in a consistent way, may be safely compared with other measurements made using the same pattern and will be correctly interpreted for all future uses. A side benefit of using a standard pattern is that the effort to determine the Measurement Strategy parameters is much reduced.

The COSMIC Group recommends that measurers study and master the COSMIC method, especially the Measurement Strategy parameters, before using the standard patterns.

2.3 Identification of the FUR from software artefacts.

As illustrated in Figure 2.2, FUR can be derived from software engineering artefacts that are produced before the software exists. Thus, the functional size of software can be measured prior to its implementation in a computer system.

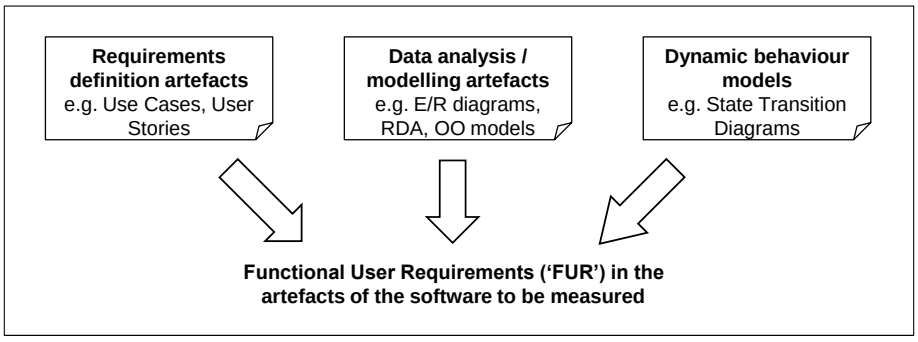


Figure 2.2 – Pre-implementation sources of Functional User Requirements.

NOTE: Some existing software may need to be measured without there being any, or with only a few, architecture or design artefacts available, and the functional requirements might not be documented (e.g. for legacy software). In such circumstances, it is still possible to derive the FUR from the artefacts of the computer system even after it has been implemented, as illustrated in Figure 2.3.

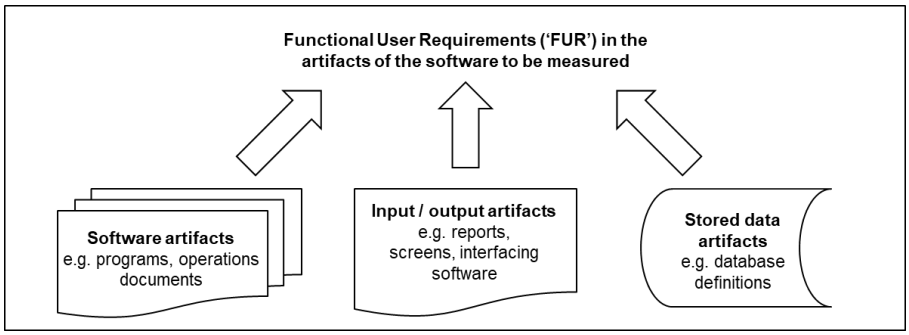


Figure 2.3 – Post-implementation sources of Functional User Requirements (FUR).

The process to be used and hence the effort required to extract the FUR from different types of software engineering artefacts or to derive them from installed software will obviously vary; these processes cannot be dealt with in the Measurement Manual. The COSMIC method assumes that the FUR of the software to be measured either exist or that they can be extracted or derived from its artefacts, in light of the purpose of the measurement.

If the measurer understands these two models, it will always be possible to derive the FUR of a piece of software to be measured from its available artefacts, though the measurer may have to make some assumptions due to missing or unclear information.

2.4 Non-Functional Requirements.

System NFR can be very significant for a software project. In extreme cases, a statement of requirements for a software-intensive system can require as much documentation for the NFR as for the functional requirements. The COSMIC method can be used to measure some requirements that may be first expressed as non-functional. Several studies have shown that some requirements that initially appear as *system* NFR evolve as a project progresses into a mixture of requirements that can be implemented in software functions, and other requirements or constraints that are truly 'non-functional'. See Figure 2.4.

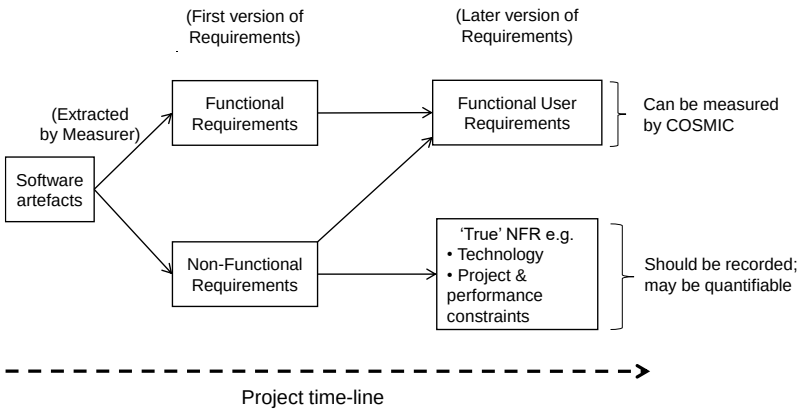


Figure 2.4 - Many requirements initially appearing as NFR evolve into FUR as a project progresses.

This is true for many system quality constraints, such as response time, ease of use, maintainability, etc. Once identified, these software functions that have been 'hidden' in NFR at the beginning of a project can be sized using the COSMIC method just as any other software functions. Not recognizing this 'hidden' functional size is one reason why software sizes can appear to grow as a project progresses.

More in-depth discussions on system and software Non-Functional Requirements (NFR) are presented in: 'Guideline on Non-Functional & Project Requirements'.

2.5 Identification of the layers.

Software architectures can be very complex, however COSMIC employs a very simplified view of the software architecture that is sufficient for the purpose of measurement. It is the task of the measurer to perform that simplification. This section provides guidance to the measurer.

Since the scope of a piece of software to be measured must be confined to a single software layer, the process of defining the scope(s) of FSM may require that the measurer first has to decide what are the layers of the software's architecture. This sub-section discusses 'layers' of software as these terms are used in the COSMIC method because:

- the measurer may be faced with measuring some software in a 'legacy' environment of software that evolved over many years without ever having been designed according to an underlying architecture The measurer may therefore need guidance on how to distinguish layers according to the COSMIC terminology;
- the expressions 'layer' and 'layered architecture' are not used consistently in the software industry. When the measurer must measure some software that is described as being in a 'layered architecture', it is advisable to check that 'layers' in this architecture are defined in a way that is compatible with the COSMIC method. To do this, the Measurer should establish the equivalence between specific architectural objects in the 'layered architecture' paradigm and the concept of layers as defined in this manual.

In a defined software architecture, each layer may have the following characteristics:

a) Software in one layer provides a set of services that is cohesive according to some defined criterion, and that software in other layers can utilize without knowing how those services are implemented.

b) The relationship between software in any two layers is defined by a 'correspondence rule' which may be either:

- 'hierarchical', i.e. software in layer A is allowed to use the services provided by

software in layer B but not vice versa (commonly referred to as Client-Server);

- ‘bi-directional’, i.e. software in layer A is allowed to use software in layer B, and vice versa (commonly referred to as peer-to-peer (P2P).

c) Software in one layer exchanges data groups with software in another layer via their respective functional processes.

d) Software in one layer does not necessarily use all the functional services supplied by software in another layer.

e) Software in one layer of a defined software architecture may be partitioned into other layers according to a different defined software architecture.

Guidance on Rule 4: Identification of Layers

If the overall scope of the FSM extends over multiple layers, the measurer should proceed as follows:

- If the software to be measured exists within an established architecture of layers that can be mapped to the COSMIC layering characteristics as defined above, then that architecture should be used to identify the layers for measurement purposes.
- If however, the purpose requires that some software is measured that is not structured according to the COSMIC layering characteristics, the measurer should try to partition the software into layers by applying the principles defined above.
- Conventionally, infrastructure software packages such as database management systems, operating systems or device drivers, that provide services that can be used by other software in other layers, are each located in separate layers.

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.

Once identified, each layer can be registered in the Measurement report, with the corresponding label.

2.6 Identification of the functional users.

GUIDANCE on Rule 7: Identification of the functional users.

The identification of functional user (or users) is determined by the Functional ‘User’ Requirements that must be measured and by the purpose of the measurement.

NOTE: There is nothing absolute about a functional user, i.e. identify the functional users per functional process. A sender/receiver of data may be a functional user of one or more functional processes, but not a functional user of other functional processes, even in the same software being measured.

2.7 Levels of decomposition.

Size measurements of the components of a piece of software are only directly comparable for components at the same level of decomposition. This is important because sizes of pieces of software at different levels of decomposition cannot be simply added up without taking into account the aggregation rules of chapter 5. Further, as a consequence, the performance (e.g. the productivity = size/effort) of projects to develop different pieces of software can only be compared if all the pieces of software are at the same level of decomposition.

Different levels of decomposition of a piece of software may correspond to different 'views' of the software's layers, e.g., as in Figure 1.3 in Part 3c. However, software may be decomposed into 'levels' regardless of whether or not it is designed using a layered-architecture model.

2.8 Context diagrams.

It can be helpful when defining a scope of FSM and the functional users to draw a 'context diagram' for the software being measured. Context diagrams show the flows of data between the piece of software and its functional users (humans, hardware devices or other software) and also show the flows of data between the piece of software and persistent storage.

A context diagram is an instance of a measurement pattern applied to the software being measured. Symbols used in context diagrams are presented in Figure 2.5.

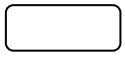
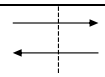
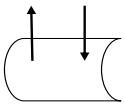
Symbol	Interpretation
	The piece of software to be measured (box with thick outline) i.e. the definition of a measurement scope.
	Any functional user of the software being measured.
	The arrows represent <u>all</u> the movements of data crossing a boundary (the dotted line) between a functional user and the software being measured.
	The arrows represent <u>all</u> the movements of data between the software being measured and 'persistent storage'. (The flowchart symbol for 'data storage' emphasizes that persistent storage is an abstract concept. This symbol indicates that the software does not interact directly with physical hardware storage.)

Figure 2.5 - Symbols of context diagrams.

2.9 Identification of the level of granularity.

COSMIC requires the FUR to be expressed at a level of detail sufficient to create the COSMIC measurement models: this is called the level of granularity.

To derive a functional size using the COSMIC FSM using the RULES of ISO 19761, the necessary level of granularity is that at which individual functional processes and their data movements can be identified and defined. When functional details are missing at other levels of granularity of the requirements, measurements should be made using one of the size approximation techniques described in the related 'Early Software Sizing with COSMIC: Practitioners Guide'.

NOTE 1 In the initial stages of a software development project, actual requirements are specified 'at a high level', that is, in outline, or in little detail. As the project progresses, the actual requirements are refined, (e.g., through versions 1, 2, 3 etc.), revealing more and more detail 'at lower levels'. These different degrees of detail of the actual requirements are known as different 'levels of granularity'.

NOTE 2: Measurers should be aware that when requirements are evolving early in the life of a software project, at any moment different parts of the required software functionality will typically have been documented at different levels of granularity.

For an example of measuring at varying levels of granularity and of decomposition, see the telecoms system example in the 'Guideline for early or rapid COSMIC functional size measurement using approximation approaches'

3. THE MAPPING PHASE.

Functional processes are composed of sub-processes that move data (*'data movements'*) and optionally, may manipulate data (*'data manipulation').*

3.1 Mapping the FUR to the Generic Software Model.

Figure 3.1 shows the steps for mapping the FUR in the available software artefacts to the form required by the COSMIC Generic Software Model.

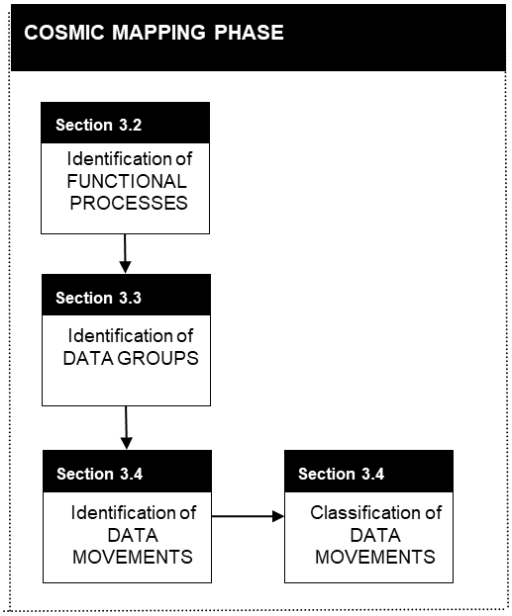


Figure 3.1 –The process of the COSMIC Mapping Phase.

Several guidelines are available that describe how to map from various data-analysis and requirements-determination methods, used in different domains to the concepts of the COSMIC method:

- Guideline for Sizing Business Application Software,
- Guideline for Sizing Data Warehouse Application Software,
- Guideline for Sizing Service-Oriented Architecture Software, and
- Guideline for Sizing Real-time Software.

For the business and real-time domains there are also Quick Reference Guides available that give an overview of the process in a few pages.

3.2 Identifying functional processes.

The first step of the Measurement Phase is to identify the set of functional processes of the piece of software to be measured, from its FUR.

The relationships between a triggering event, the functional user and the Entry data movement that triggers a functional process being measured are presented in Figure 3.2 where: *a triggering event causes a functional user to generate a data group that is moved by the triggering Entry of a functional process to start the functional process.*

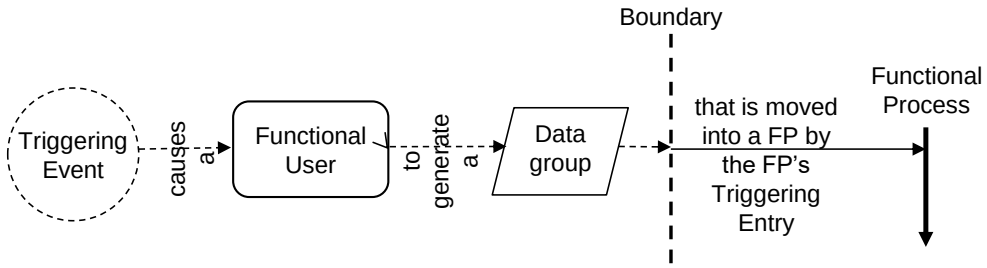


Figure 3.2 – Relationships between a triggering event, a functional user & a functional process.

NOTE: For ease of reading, the reference to the data group is omitted when stating that a functional user *initiates* a triggering Entry that starts a functional process, or even more simply that a functional user *initiates* a functional process.

Guidance on Rule 10: Identification of Functional Processes

The process of identifying functional processes, after the functional users have been identified, given the FUR for the software being measured follows the chain of Figure 3.2:

1. Identify the separate events in the world of the functional users that the software being measured must respond to – the 'triggering events'

   NOTE: Triggering events can be identified in state diagrams and in entity life-cycle diagrams, since some state transitions and entity life-cycle transitions correspond to triggering events to which the software must react.

2. Identify which functional user(s) of the software may respond to each triggering event;
3. Identify the data group(s) (i.e. the triggering Entry or Entries) that each functional user may initiate in response to the event;
4. Identify the functional process started by each triggering Entry.

Use the following checks to ensure that candidate functional processes (FP) have been properly identified:

1. Do all the identified FPs of the piece of software measured reside in the same layer?

2. Are all the identified FPs comprised of an Entry and at least 1 Write or Exit data movement?

3.3 Identification of data groups.

3.3.1 Identification of data groups.

Having identified the functional processes, the next step is to identify their data movements. The following guidance assists in the identification of data groups and hence objects of interest particularly in the output of functional processes.

GUIDANCE on Rule 11: Identifying different data groups moved in the same one functional process.

For all the data attributes appearing in an Entry-Exit-Read-Write of a functional process:

a) sets of data attributes that have different frequencies of occurrence describe different objects of interest;

b) sets of data attributes that have the same frequency of occurrence but different identifying key attribute(s) describe different objects of interest;

c) all the data attributes in a set resulting from applying parts a) and b) of this guidance belong to the same one data group, unless the FUR specify that there may be more than

one data group describing the same object of interest (see the Guidance on Rules **13 and 14 – Data Movements Uniqueness**, cases b) and c)).

NOTE 1: A functional user of the software being measured may be the object of interest of a data group sent or received by the functional user.

NOTE 2: In theory, a data group might contain only one data attribute if this is all that is required, from the perspective of the FURs, to describe the object of interest. In practice, such cases occur commonly in real-time application software (e.g. the data group entered to convey the tick of a real-time clock or the entry of the state of a sensor); they are less common in business application software.

NOTE 3: There is nothing absolute about an object of interest, i.e. identify the objects of interest per functional process. A 'thing' may be an object 'of interest' to a functional user via one or more functional processes, but not be an object 'of interest' to another functional user via other functional processes, even in the same software being measured.

The origin of a data group can be of many forms, e.g.:

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

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

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

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

3.3.2 About the identification of objects of interest and data groups.

The definition and principles of objects of interest and of data groups are intentionally broad in order to be applicable to the widest possible range of software: this sometimes results in it being difficult to apply the definition and principles when measuring a specific piece of software. See Part 3 for examples to assist in the application of the principles to specific cases

When faced with a need to analyze a group of data attributes that is moved in or out of a functional process or is moved by a functional process to or from persistent storage, *it is critically important* to decide if the attributes all convey data about a single 'object of interest', since it is the latter that determine the number of separate 'data groups' as defined by the COSMIC method that will be moved by data movements.

For instance, if the data attributes to be input to a functional process are attributes of three separate objects of interest, then three separate 'Entry' data movements must be identified. Deciding on the number of data groups can be difficult when analyzing the output of a functional process of a business application which may include:

- multiple data groups, each describing a different object of interest, e.g. a report showing totals at various levels of aggregation;
- the results of enquiries where the output will vary depending on the input;
- data groups that may even be unrelated to each other, e.g. an invoice which includes an advertisement for an unrelated service.

When analyzing complex output, e.g. reports with data describing several objects of interest, consider each separate candidate data group as if it were output by one separate functional process. Each of the data group types identified this way must also be distinguished and counted when measuring the complex report.

3.3.3 Data or groups of data that are not candidates for data groups.

Any data appearing on input or output screens or reports that are not related to an object of interest to a functional user should not be identified as indicating a data group.

The COSMIC Generic Software Model assumes that all manipulation of data within a functional process is associated with the four data movement types. Hence no data groups may be identified arising from data manipulation within a functional process in addition to the data groups moved by the Entries, Exits, Reads and Writes of the functional process.

3.3.4 Identification of data attributes (optional).

In the COSMIC method, it is not mandatory to identify the data attributes. However, understanding the concept of a 'data attribute' is necessary to understand how to measure changes'. A requirement to change a data attribute can result in the data movement to which the attribute belongs being indicated as 'changed'.

Also, it may be helpful to analyze and identify data attributes in the process of distinguishing data groups and objects of interest.

3.4 Identification of data movements.

This step consists in identifying the data movements (Entry, Exit, Read and Write) of each functional process.

GUIDANCE on Rule 12: Data Movements

The following guidance helps to confirm the status of a candidate Entry data movement:

GUIDANCE on Rule 16: Entry (E).

a) The data group of a triggering Entry may consist of only one data attribute which simply informs the software that 'an event Y has occurred'.

b) For clock-ticks that are triggering events identify an Entry from a functional user, in this case the Clock.

c) Unless a specific functional process is necessary, obtaining the date and/or time from the system's clock is not considered an Entry or any other COSMIC data movement.

NOTE: Very often, especially in business application software, the data group of the triggering Entry has several data attributes which inform the software that 'an event Y has occurred and here is the data about that particular event'.

GUIDANCE on Rule 17: Exit (X).

a) For an enquiry which outputs fixed text, (where 'fixed' means the message contains no variable data values), identify an Exit for the fixed text output.

b) When identifying Exits, ignore all fields and other headings that enable human users to understand the output data.

GUIDANCE on Rule 18: Read (R).

Do not identify a Read when the FUR of the software being measured specify any software or hardware functional user as the source of a data group, or as the means of moving the data group to persistent storage. Interaction with other functional users is by definition across a boundary (Generic Software Model, principle 1), which is handled by an Entry data movement. The actual Read takes place within the boundary of the software being measured.

GUIDANCE on Rule 19: Write (W).

Do not identify a Write when the FUR of the software being measured specify any software or hardware functional user as the destination of the data group or as the means of retrieving a persistently-stored data group. Interaction with other functional users is by definition across a boundary (Generic Software Model, principle 1), which is handled by an eXit data movement. The actual Write takes place within the boundary of the software being measured.

NOTE: When the FUR require data to be stored or to be retrieved from storage, the measurer must investigate whether the data can be stored or retrieved within its own boundary, i.e. to/from 'persistent storage', or whether data is required to be stored/retrieved with help of a functional user of the software being measured (i.e. via some other piece of software, or directly to or from a hardware device).

GUIDANCE on Rules 16 to 19 – Data manipulation associated with data movements.

The data manipulation associated with any of these data movements does not include any data manipulation that is needed after the data movement has been successfully completed, nor does it include any data manipulation associated with any other data movement.

The following guidance cover the most common situation (guidance a)) and other possible valid cases (guidance b) and c)):

- In a) the occurrences of the data group are subject to the same FUR: so one data group and one data movement is identified.
- In b) and c) the same applies to *each different data group separately:* so one data group and one data movement per different data group is identified.

GUIDANCE on Rules 13 and 14 – Data Movements Uniqueness.

a) Unless the FUR are as given in guidance b) or c), all data describing any one object of interest that is required to be entered into one functional process is identified as one data group moved by one Entry.

NOTE 1: A functional process may, of course, have multiple Entries, each moving data describing a different object of interest.
NOTE 2: The same equivalent guidance applies to any Read, Write or Exit data movement in any one functional process.

b) If FUR specify that different data groups must be entered into one functional process, each from a different functional user, where each data group describes the same object of interest, then one Entry is identified for each of these different data groups.

NOTE 1: The same equivalent guidance applies for Exits of data to different functional users from any one functional process.
NOTE 2: Any one functional process has only one triggering Entry.

c) If FUR specify that different data groups must be moved from persistent storage into one functional process, each describing the same object of interest, then one Read is identified for each of these different data groups.

NOTE 1: The same equivalent guidance applies for Writes in any given functional process.
NOTE 2: This guidance is analogous to rule b). In the case of the FUR to read different data groups describing the same object of interest, they will likely have originated from different functional users. In the case of the FUR to write different data groups, they will likely be made available to be read by different functional users.

GUIDANCE on Rules 16-17: Functional process requiring data from a functional user.

a) When the functional process does not need to tell the functional user what data to send, a single Entry is sufficient (per object of interest).

b) When the functional process needs to tell the functional user what data to send, an Exit followed by an Entry are necessary.

GUIDANCE on Rules 16-19: Control commands in applications with a human interface.

In an application with a human interface 'control commands' are ignored as they do not involve any movement of data about an object of interest.

Error and confirmation messages are specific forms of an Exit and the Rules governing identification apply.

GUIDANCE on Rules 16-19: Error/confirmation messages & other indications of error conditions.

a) One Exit is identified to account for all types of error/confirmation messages issued by any one functional process of the software being measured from all possible causes according to its FUR.

b) If a message to a human functional user provides data in addition to confirming that entered data has been accepted, or that entered data is in error, then this additional data is identified as a separate data group moved by an Exit in the normal way.

c) All other data, issued or received by the software being measured, to/from its hardware or software functional users should be analyzed according to the FUR as Exits or Entries respectively, according to the normal COSMIC rules, regardless of whether or not the data values indicate an error condition.

d) Reads and Writes are considered to account for any associated reporting of error conditions. Therefore, no Entry to the functional process being measured is identified for any error indication received as a result of a Read or Write of persistent data.

e) No Entry or Exit is identified for any message indicating an error condition that might be issued whilst using the software being measured but which is not required to be processed in any way by the FUR of that software, e.g. an error message issued by the operating system.

3.5 Measuring the components of a distributed software system.

When the purpose of a measurement is to measure separately the size of each component of a distributed software system, a separate scope of FSM must be defined for each component. In such a case the sizing of the functional processes of each component follows all the rules as already described.

From the process for each measurement (… define the scope, then the functional users and boundary, etc. …) it follows that if a piece of software consists of two or more components, there cannot be any overlap between the scope of FSM of each component. The scope of FSM for each component must define a set of complete functional processes. For example, there cannot be a functional process with part in one scope and part in another. Likewise, the functional processes within the measurement scope for one component do not have any information about the functional processes within the scope of another component, even though the two components exchange messages.

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

3.6 Re-use of software.

Any two or more functional processes in the same software being measured may have some functionality that is identical or very similar in each process and is described separately elsewhere in the requirements. This phenomenon is referred to as 'functional commonality', or functional 'similarity'.

However, each functional process is defined, modelled and measured independently of, i.e. without reference to any other FUR in the same software being measured.

Therefore, if the FUR for a given functional process makes a reference to FUR elsewhere in the requirements then the size of that referenced functionality is to be included in the size of the functional process being measured.

3.7 Measurement of the size of changes to software.

A 'functional change' to existing software is interpreted in the COSMIC method as 'any combination of additions of new data movements or of modifications or deletions of existing data movements, including to the associated data manipulation'. The terms 'enhancement' and 'maintenance'[1] are often used for what we here call a 'functional change'.

The need for a change to software may arise from either:

- a new FUR (i.e. only additions to the existing functionality), or
- from a change to the FUR (perhaps involving additions, modifications and deletions) or
- from a 'maintenance' need to correct a defect.

The rules for sizing any of these changes are the same but the measurer is alerted to distinguish the various circumstances when making performance measurements and estimates.

A data movement is considered to be functionally modified as follows.

GUIDANCE on Rule 24: – Modifying a data movement.

a) A data movement is considered to be functionally modified if at least one of the following applies:

- the data group moved is modified.
- the associated data manipulation is modified .

b) A data group is modified if at least one of the following applies:

- one or more new attributes are added to the data group.
- one or more existing attributes are removed from the data group.
- one or more existing attributes are modified, e.g. in meaning or format (but not in their values).

c) A data manipulation is modified if it is functionally changed in any way.

d) If a data movement must be modified due to a change of the data manipulation associated with the data movement and/or due to a change in the number or type of the attributes in the data group moved, one changed CFP is measured, regardless of the actual number of modifications in the one data movement.

e) If a data group must be modified, data movements moving the modified data group whose functionality is not affected by the modification to the data group is not identified as changed data movements.

NOTE: A modification to any data appearing on input or output screens that are not related to an object of interest to a functional user is not identified as a changed CFP.

f) A normal measurement convention is that the functional size of a piece of software does not change if the software must be changed to correct a defect so as to bring the software in line with its FUR. The functional size of the software does change if the change is to correct a defect in the FUR.

Modified data movements have no influence on the size of the piece of software as they exist both before and after the modifications have been made.

When a piece of software is completely replaced, for instance by re-writing it, with or without extending and/or omitting functionality, the functional size of this change is the size of the replacement software, measured according to the normal rules for sizing new software.

NOTE Usually, the size of a functional change (discussed here) and the change in the functional size of the software differ.

4. OTHER TOPICS.

4.1 Extending the COSMIC measurement method – Local extension.

The COSMIC method of measuring a functional size does not presume to measure all possible aspects of software 'size'. Thus the method is currently not designed to measure separately and explicitly the size of the FUR of data manipulation sub-processes. The influence on size of data manipulation sub-processes is taken into account via a simplifying assumption that is valid for a wide range of software domains.

Nevertheless, the COSMIC size measure is considered to be a good approximation for the method's stated purpose and domains of applicability. Yet, it may be that within the local environment of an organization using the COSMIC measurement method, it is desired to account for such functionality in a way which is meaningful as a local standard. When such local extensions are used, the measurement results must be reported according to the special convention presented in section 6.

4.2 COSMIC in Agile.

The COSMIC method is being used successfully to measure the size of User Stories in Agile software developments, which may have very few data movements. This practice is well-established, with many reports of the CFP sizes of Agile iterations (or 'sprints'), aggregated from the sizes of individual User Stories, that correlate very well with the effort to develop the iteration (and much better than the correlation with effort of sizes measured using Story Points). See (https://cosmic-sizing.org/publications/guideline-for-sizing-agile-projects-with-cosmic/)

5. MEASUREMENT PHASE.

The general process for measuring a piece of software when its Functional User Requirements have been expressed in terms of the COSMIC Generic Software Model is summarized in Figure 5.1 below.

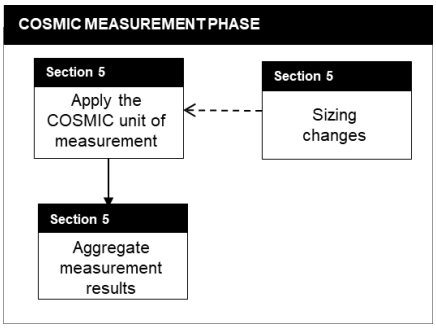


Figure 5.1 – The process for the COSMIC Measurement Phase.

GUIDANCE on Rule 23: Aggregation of functional sizes.

a) Sizes of pieces of software or of changes to pieces of software may be added together only if measured at the same functional process level of granularity of their FUR.

b) Sizes of pieces of software and/or changes in the sizes of pieces of software within any one layer or from different layers are added together only if it makes sense to do so, for the purpose of the measurement.

c) The size of a piece of software is obtained by adding up the sizes of its components (regardless of how the piece is decomposed) and eliminating the size contributions of inter-component data movements.

Within each identified layer, the aggregation function is fully scalable. A sub-total can be generated for individual functional processes or for all the functional processes of the software, depending on the purpose and scope of the measurement exercise.

6. MEASUREMENT REPORTING.

The result must be reported and data about the measurement recorded so as to ensure that the result is always unambiguously interpretable. COSMIC measurement results are to be reported and archived according to the following conventions.

GUIDANCE on Rule 25: COSMIC measurement labeling.

A COSMIC measurement result is noted as '***x*** CFP (v) ', where:

- '**x**' represents the numerical value of the functional size,
- 'v' represents the identification of the version of the standard COSMIC method used to obtain the numerical functional size value 'x'.

NOTE: If a local approximation method was used to obtain the measurement, but otherwise the measurement was made using the conventions of a standard COSMIC version, the above labeling convention is used, but use of the approximation method should be noted elsewhere.

GUIDANCE on Rule 25: COSMIC local extensions labeling.

A COSMIC measurement result using local extensions is noted as:

' ***x*** CFP (v.) + z Local FP', where:

- '**x**' represents the numerical value obtained by aggregating all individual measurement results according to the standard COSMIC method, version v,
- 'v' represents the identification of the version of the standard COSMIC method used to obtain the numerical functional size value 'x'.
- 'z' represents the numerical value obtained by aggregating all individual measurement results obtained from local extensions to the COSMIC method.

COSMIC Measurement Manual for ISO 19761

Part 3a: Standard Measurement Strategy Examples

Version 5.0

August 2021

Foreword.

The COSMIC Measurement Manual for ISO/IEC 19761:2011 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)

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

The purpose of this Part 3a of the COSMIC Measurement Manual is to document a set of measurement (strategy) pattern examples ('standard strategies') for commonly-occurring situations, that ensure 'like-for-like' size comparisons. For each standard strategy it will be discussed whether the resulting size measurements can be safely used to establish valid, comparable project performance measurements and performance benchmarks, and hence can be used for estimating new projects. The examples are presented per functional domain.

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

August 2021 minor editing:

In the Foreword the names of the Parts have been updated. No other changes.

Editors:

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

Other members of COSMIC Measurement Practices Committee:

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

Table of Contents.

1 INTRODUCTION TO THE STANDARD STRATEGY EXAMPLES.

1.1 Purpose of standard strategies.

A house has many different sizes, even when measured using the same measurement unit (e.g. square meters). The different sizes depend on *what* you want to measure (e.g. usable floor area, external 'footprint' area, etc.), *who* needs the measurement (e.g. architect, builder, occupier, estate agent/realtor, etc.), and even *when* you measure, because plans may change during construction.

Similarly, a piece of software can have different functional sizes, even when measured using one method such as the COSMIC Functional Size Measurement method with the same unit of measure, the COSMIC Function Point (CFP). These different sizes of the piece of software may result from:

- different purposes of the measurement, which result in different measurement *scopes* (i.e. the extent of functionality to be included in the measurement), and
- the *view* of the software by its functional users (e.g. a human end-user of the software may not 'see' all the functionality that the developer must provide).

Regarding the last point, functionality a component uses to exchange data with other components is invisible to the external human user so will not be included in the size of the 'whole'. i.e. the size of the whole is less than the sum of the sizes of its components. Measurers must therefore clearly identify and distinguish measurements of 'whole' pieces of software from measurements of software components.

To ensure 'like-for-like' size comparisons a set of measurement (strategy) pattern examples ('standard strategies') for commonly-occurring situations have been developed. A standard strategy defines a standard combination of parameters, namely the scope and the functional users (the 'what' and the 'view' above), that must be determined in the Measurement Strategy phase of a COSMIC measurement process. As it also defines the possible kinds of data movements, a standard strategy provides a template for drawing the context diagram for the software to be measured.

Given the purpose of the measurement the measurer selects the appropriate standard strategy and completes the measurement strategy, including statement of the standard strategy used. If a standard strategy is missing, the measurer should define a non-standard ('local') strategy and record it for later reuse (see chapter 4).

Systematically using the standard strategies is important because:

- it entails a standardization of the measurement process, speeding it up and increasing the measurement routine by repetition of this approach,
- measurements with the same standard strategy are based on comparable functionality, therefore can be used for productivity reasons and for estimating effort for future development of similar software.

Note that many organizations will specialize in developing software for one specific functional domain, so will not need all these standard strategies.

1.2 Terminology.

Terms used in this Part are either standard COSMIC terms or are taken from the ‘COSMIC/ISBSG Concise Data Collection Questionnaire (see chapter 5).

benchmarking
comparing a development productivity against another development productivity, within an organization or between organizations.

dumb device
device that only provides or accepts data

intelligent device
device that can receive and process data, or that must be told what data to send.

levels of decomposition
for the purposes of this document, the levels of decomposition are

- Whole: no need to measure components separately (synonym: no decomposition),
- Major Component: component of the software at the first ’Level 1’ of decomposition,
- Minor Component: component at any level of decomposition below the Major Component level.

Note: in the domains of real-time and infrastructure software it may be hard to clearly distinguish different levels of decomposition (or scale of the software). Great care is therefore needed in these domains to consider software scale to ensure like-for-like performance comparisons.

standard strategy

synonym of ‘measurement (strategy) pattern’.

suited for external benchmarking
sizes measured using the same standard strategy are comparable across multiple organizations and resulting project performance data are suitable for submission to benchmarking organizations such as the ISBSG.

suited for internal benchmarking
sizes measured using the same standard strategy are comparable within one organization.

2 STANDARD STRATEGIES.

Note: sizes using a standard strategy may be considered suitable for both internal and external benchmarking, unless a Suitability reservation has been made.

2.1 Business applications.

2.1.1 Standard strategy for whole on-line business applications.

The standard strategy that corresponds with the context diagram shown in Figure 2.1 should be used when the need is to size an on-line business application A, to be measured as a whole. Functional users are humans and interfacing applications.

The Functional User Requirements (FUR) of any on-line business application A define the functionality needed by the business, including the human interface requirements for the customer of the software. The FUR do not describe functions provided by the operating environment or by 'control commands'.

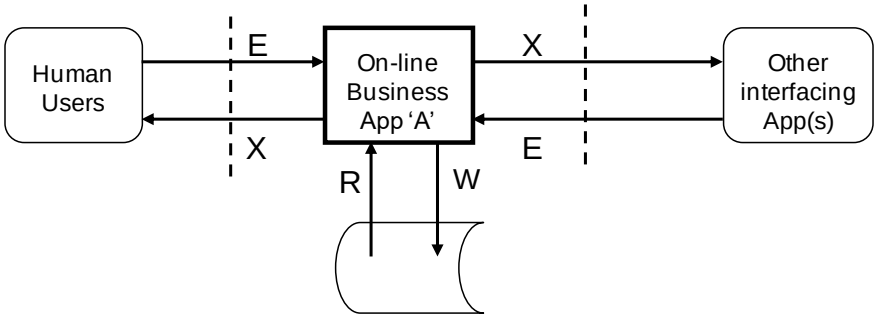


Figure 2.1 – Whole online business application.

2.1.2 Standard strategy for major components of on-line business applications.

The standard strategy that corresponds with a context diagram similar to Figure 2.2 should be used when the need is to size a major component of an on-line business application separately. As an example, the context diagram for an on-line business application that is developed using a multi-layer (or multi-tier) architecture with the common three major components User Interface (UI), Business Rules (BR) and Data Services (DS). The two boundaries shown between these major three components coincide with the interfaces between the three layers (or tiers) in which these components reside.

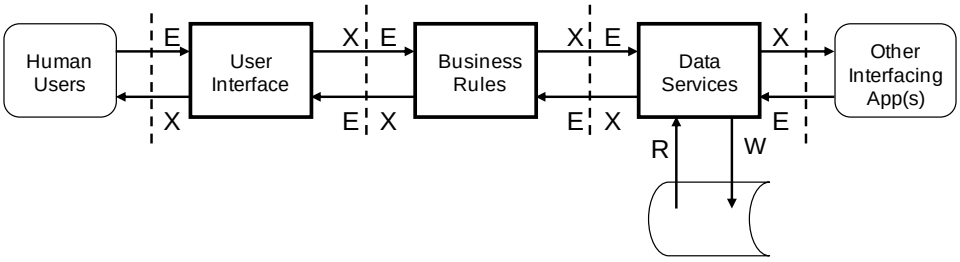


Figure 2.2 – Major components of an online business application.

Each major component can be measured in the normal way. The level of decomposition is 'Major Component' (or 'Level 1' as per the COSMIC definition of Level of Decomposition).

Suitability for external benchmarking: the approach to external benchmarking will depend on two main factors.

First, is the purpose to compare the performance against external benchmarks of (a) the whole project, or (b) of each separate sub-project to develop each major component?

Second, are the major components all developed using the same programming language and technology platform, or using different language/technology combinations?

For purpose (a), if the major components are developed using the same programming language and technology platform then the performance of the 'whole'

application may be sensibly compared against the performance of the whole on-line business application software in section 2.1.1, because the size of the 'whole' application is the sum of the sizes of the three major components less the size contribution from all Entries and Exits crossing the two boundaries between the three major components.

For purpose (b), the size of each of the three major components may be combined with the respective development effort for each component to derive three separate productivity figures, one for each component. However, a problem is that project effort on several activities at the beginning of the project (e.g. for requirements elicitation) and towards the end of the project (e.g. for integration, system and user acceptance testing) is common to all three components. Even a project developed following an 'Agile' model may have some 'common effort' across all major components. This common effort can either be ignored so that the productivity figures represent only the design, programming and unit testing effort for each major component, or the common effort must be allocated in some way to each major component. There is no standard way of doing this, so external benchmarking at the major component level has this inherent problem.

Suitability for internal benchmarking: in practice this should be easier than external benchmarking since it is possible to define local standards to enable fair comparisons that take account of (or that ignore) 'common' project activities, and that have the same functionality and language/technology distribution.

2.1.3 Standard strategy for batch processed business applications.

The standard strategy that corresponds with the context diagram shown in Figure 2.3 should be used when the need is to size an application B as a whole that processes data in batch mode.

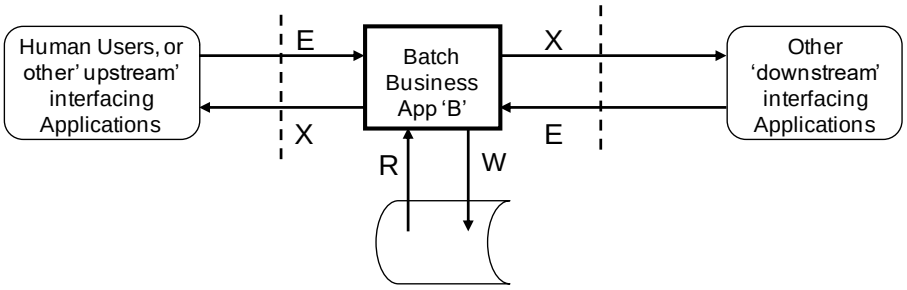


Figure 2.3 - Business application processed in batch mode.

Any human users of a batch process do not interact directly with the software. However, human users provide the input data for the functional processes of the application and receive the output data from the batch processing. Functional users are the applications that send data to (or that make a batch of input data available for) the application B for batch processing and applications that receive data from application B.

For the different mechanisms by which data may be input to a batch-processed application B (e.g. via an interface file, or by transmission) or by which a batch-processed application may be triggered when there seems to be no input data (e.g. to produce a series of reports), see Part 3c MIS Examples.

Suitability for external benchmarking: The size of a batch-processed application may be sensibly compared against the size of the on-line business application software in section 2.1.1. However, when analyzing project performance data for benchmarking purposes, projects that deliver batch applications should be distinguished from those delivering on-line applications as the different technologies used normally lead to different project performance levels.

Suitability for internal benchmarking: Suitable, subject to distinguishing batch from on-line applications.

For an example of how to measure a software system comprising an on-line and a batch component, see the example of section 3.1.

2.2 Real-time applications.

2.2.1 Standard strategy for whole real-time applications.

This standard strategy should be used when the need is to size of a typically large-scale, real-time application, seen as a whole, with hardware devices and other interfacing software as its functional users. Large-scale real-time systems may have to interface with thousands of 'intelligent' input/output devices containing embedded software and/or many other software systems. Examples would be the application software of:

- telecoms network systems,
- large-scale process control systems, e.g. for paper or steel-making,
- military command/control systems,
- major sub-systems of a distributed avionics system (e.g. for fuel management, automatic landing etc.),
- the 'hub' of a wide-scale environmental monitoring system,
- large-scale Internet of Things systems.

The typical context diagram for a real-time application is shown in Figure 2.4. It shows the real-time application interacting with many possible types of functional users: 'dumb' and 'intelligent' hardware devices and other interfacing software.

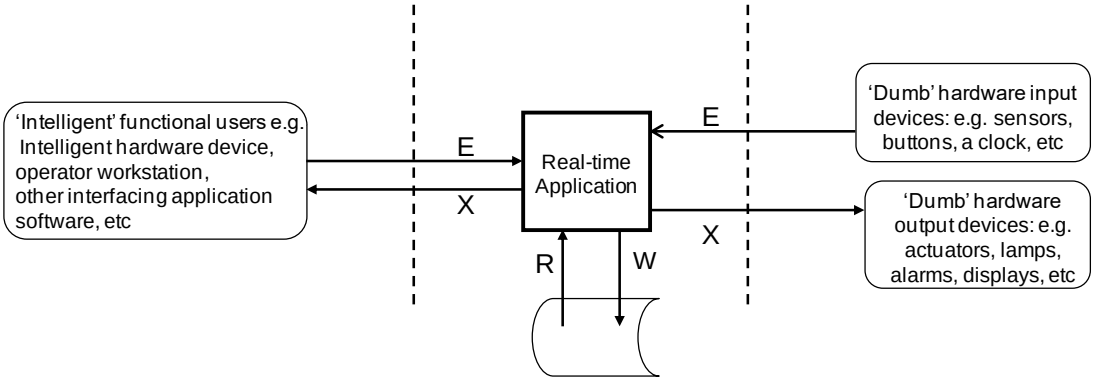


Figure 2.4 - A real-time application to be measured as a whole.

The real-time application may interact with its hardware and software functional users either directly, or indirectly via an operating system or (in the case of hardware devices via device driver software). Interactions with this operating environment

software (if any) should be ignored when measuring the real-time application, because these are not functional users in this standard strategy.

Suitability for external benchmarking: suitable, with other real-time applications of comparable scale.

2.2.2 Standard strategy for major components of real-time applications.

This standard strategy should be used when the need is to measure separately a major component of a (distributed) real-time application. The standard strategy of section 2.1.2 should be used for each major component, except that the standard strategy of section 2.2.4 should be used if the major component of the application is a human operator workstation.

2.2.3 Standard strategy for simple real-time embedded applications.

This strategy should be used to measure software that may typically have to support a limited number of 'dumb' input/output devices. It includes software embedded in microprocessors to monitor or control devices such as:

- domestic appliances (e.g., washing machines, burglar alarms, etc.),
- simple office appliances such as a stand-alone copier,
- electronic control units of a vehicle (e.g., the air conditioning, lighting, tire pressures, etc.),
- a simple, single elevator,
- small-scale Internet of Things systems.

A microprocessor with embedded software may be part of a large-scale distributed real-time system, e.g., a process control system to control a power plant.

Generally, simple real-time embedded applications allow interactions with humans only via buttons, a numeric keypad for data entry (e.g., to enter a security code), specialized displays capable of showing only short messages with limited character sets, audible alarms, and such-like. Such microprocessors may have communications capability and thus become components of distributed real-time control systems. The embedded application to be measured may or may not use a simple operating system and/or dedicated device driver software. For the latter, see section 2.3.2.

The general context diagram for simple real-time embedded application software that interacts directly with 'dumb' hardware input or output devices is the same as Figure 2.4 in section 2.2.1. A simple embedded application may also communicate with other application software functional users as part of a distributed real-time application. Level of Decomposition: no decomposition.

Suitability for external benchmarking: suitable in principle, but in practice comparators must be very carefully chosen since the range of what may be considered as 'simple' is so wide. The key factor to consider is the scale (size) of the comparators. In general, the productivity to develop a small real-time embedded application (to operate stand-alone or as a component of a distributed system) will be very much higher than the productivity to develop a large-scale real-time application.

2.2.4 Standard strategy for operator workstations of real-time applications.

The context diagram for the application software of a human operator workstation used to monitor or control a real-time application such as shown in section 2.2.1 is shown below.

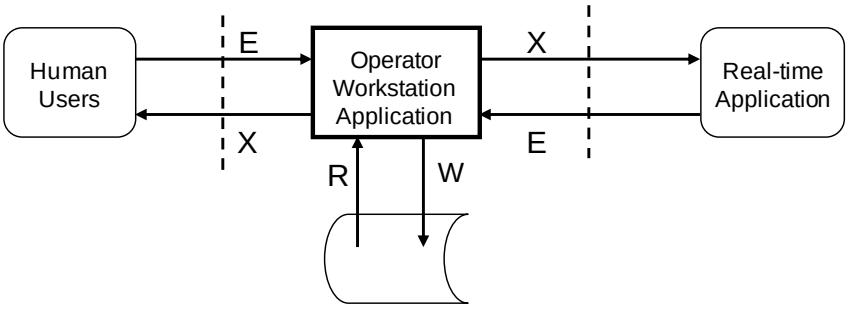


Figure 2.5 - The human operator workstation of a real-time software system.

Such workstation applications are typically needed to enable an operator to start and stop, control the configuration, set parameters, collect statistics, etc of real-time systems such as telecoms networks, major process control systems and such-like.

It is probably advisable to measure the operator workstation functionality separately from the real-time application itself as the two components have quite different design considerations. The operator workstation application functionality should certainly be measured separately if it is built using different technology from that of the real-time application.

This context diagram is essentially the same as that for an on-line business application as described in section 2.1.1. All devices that the human operator uses (e.g., keyboard, screen, alarm) and supporting software that is used to communicate with the workstation should be ignored. Level of decomposition: no decomposition.

Suitability for external benchmarking: suitable. Performance measurements from the domain of business application and real-time application software should still be analyzed separately as software from the two domains is generally subject to quite different non-functional constraints that result in different development productivity levels.

2.3 Infrastructure software.

2.3.1 Standard strategy for reusable software components.

The standard strategy that corresponds with the context diagram shown in Figure 2.6 should be used when the need is to size reusable software such as object-classes and components of a Service-Oriented Architecture. These are ‘minor components’, usually at the lowest level of decomposition. Level of decomposition: these are ‘minor components’ that usually cannot be decomposed further.

Suitability for external benchmarking: suitable, against similar components at the same level of decomposition.

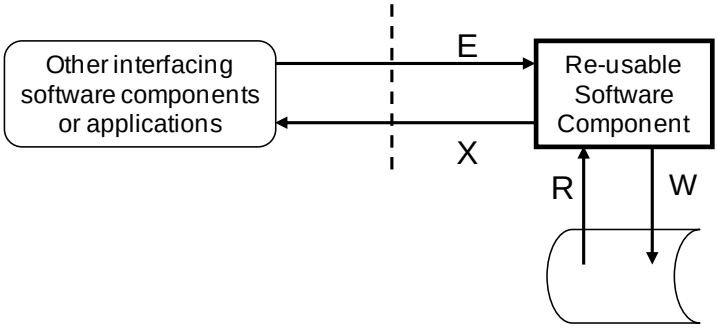


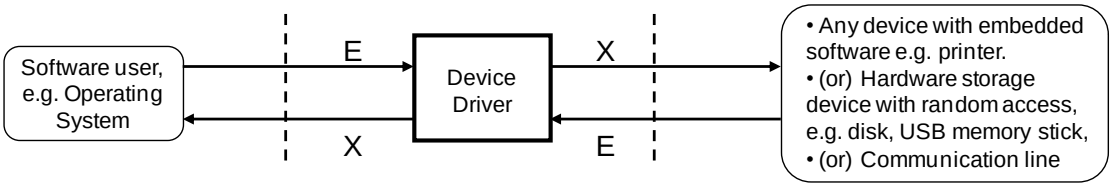
Figure 2.6 - Reusable software component.

2.3.2 Standard strategy for device driver software.

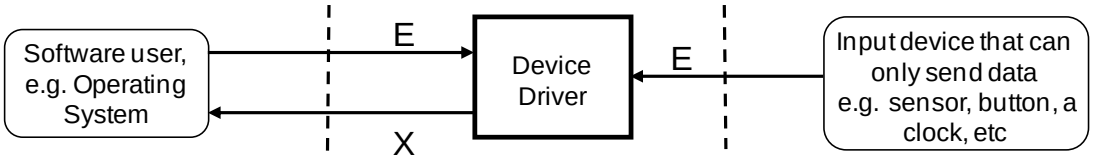
Three categories of device-driver software can be defined. Their respective context diagrams are shown below.

Suitability for external benchmarking: suitable, against similar device drivers.

a) 'Intelligent' Input / Output Device



b) 'Dumb' Input Device



c) A 'Dumb' Output Device

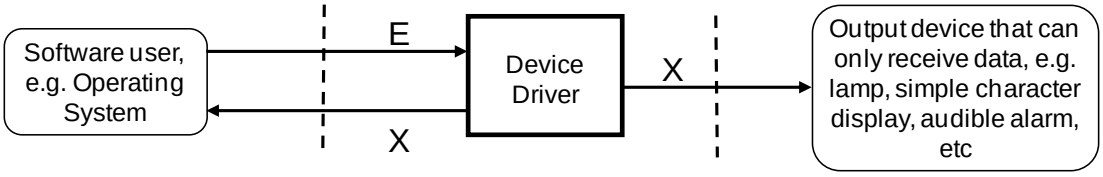


Figure 2.7 - Device driver software.

3 USING STANDARD STRATEGIES IN PRACTICE.

The purpose of this chapter is to give an example of how measurement standard strategies can be used to help decide which size of a piece of software to measure in various circumstances, and to describe how measurement results using the strategies should be reported.

3.1 Example of a mixed on-line/batch business application system

A project is required to deliver a business application that has two parts. A 'front-end' provides on-line services to human users and a 'back-end' executes in batch mode.

Data is entered during the day by human users to the on-line front-end. Physically, transactions are accumulated in an interface file as a series of records that become the input for overnight batch processing by the back end.

An example might be in an organization that provides services to customers to trade in a financial market. During the day, customers enter ‘buy’ and ‘sell’ orders of various types (e.g., for immediate, future or conditional execution) using the on-line front-end. Overnight, data about each order is processed in batch mode to update customers’ accounts, to produce a statement of each trade for each customer and to produce management reports.

The diagram below shows the two parts of the application. The measurement context diagrams from sections 2.1.1 and 2.1.3 are used to model the on-line front-end and the batch back-end respectively.

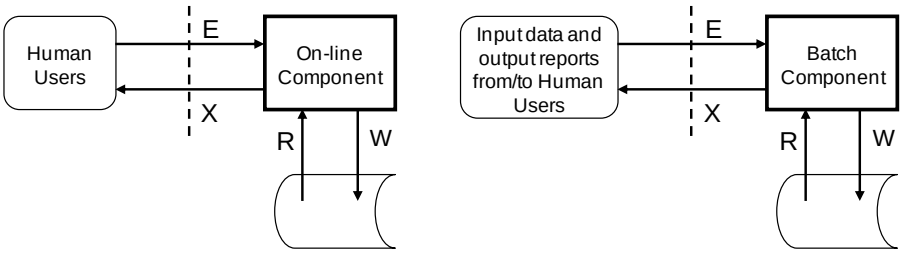


Figure 3.1 - Example of a mixed on-line/batch business application system.

Note that in the model each order processed by the on-line front-end results in a record of data about the order being moved to persistent storage by a Write command. Overnight, the batch back-end physically accesses the interface file of these orders for its input. In the model, the batch back-end has functional processes that:

- process each order and update the customer’s account,
- produce the customer statements and the management reports.

For measurement of batch processing see Part 3c MIS Examples.

There are two ways of dealing with project productivity measurements in order to ensure future comparability and their use to develop benchmarks and for estimating:

- Derive separate productivity measurements for each part of the application using the sizes and the corresponding effort to develop each part. Project effort on activities that are common for both parts, such as requirements determination and integration testing, could be allocated to the effort for each part, or alternatively ignored. In the future, analyze the performance of projects that deliver mixed on-line and batch-processed functionality separately, always ignoring the effort for these common activities
- Derive a single productivity measurement for the whole application by adding the sizes and dividing by the total effort on both parts. In the future, analyze the performance of mixed on-line/batch application developments as a separate category, considering the proportions of the functionality split between the on-line and batch parts.

Note that an on-line application and a batch-processed application should always be measured separately, with their own standard strategies.

3.2 Reporting strategy data.

Record the standard strategy or the local strategy used for the measurement. If applicable, add some measure or account of the reusable software that was used.

4 NON-STANDARD STRATEGIES, EXTENDING THE COSMIC METHOD.

4.1 Non-standard strategies.

Users of the COSMIC method should develop their own local strategies if the standard strategies described here do not meet their size comparability needs. Note that a strategy only should be developed if the strategy will be used more often. Given the purpose of strategies, single use makes no sense.

4.2 Examples of extending the COSMIC method.

EXAMPLE 1: A (standard) strategy could be extended to measure separately and explicitly the size of the FUR of data manipulation sub-processes.

EXAMPLE 2: A (standard) strategy could be extended to measure separately the influence of the number of data attributes per data movement on software size.

5 THE ISBSG AND ISO ORGANIZATIONS.

5.1 ISBSG.

ISBSG is the International Software Benchmarking Standards Group. ISBSG has a data repository of many software projects, submitted by leading IT and metrics companies from around the world. This data can be used as a benchmark for IT projects to improve planning and estimation. Data suited for external benchmarking can be submitted using the 'Concise Data Collection Questionnaire' for new development, enhancement or re-development projects measured using the COSMIC sizing method.

5.2 ISO.

ISO is the International Organization for Standardization. It develops and publish International Standards. The series of ISO standards on software project benchmarking below has been published. Suppliers of estimating methods and of benchmarking services such as the ISBSG all provide definitions of the various factors they take into account to enable comparability.

ISO/IEC 29155. Systems and software engineering - Information technology project performance benchmarking framework.

Part 1: Concepts and definitions.

Part 2: Requirements for benchmarking.

Part 3: Guidance for reporting.

Part 4: Guidance for data collection and maintenance.

COSMIC Measurement Manual for ISO 19761

Part 3b: Real-time Examples

August 2021

Foreword

This document consolidates the examples from the previous Real-time Guideline and from Part 3 of the Measurement Manual. The examples are ordered per COSMIC phase and should speak for themselves, therefore this document does not contain any explanation of the COSMIC method, if needed consult the COSMIC Measurement Manual. Also, the real-time related text has been omitted as it seems superfluous for the intended audience.

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

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

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

August 2021 minor editing:

In the Foreword the names of the Parts have been updated. No other changes.

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.

For examples of purposes and scopes see chapter 3 Comprehensive Examples and the COSMIC Case studies (freely available at cosmic-sizing.org).

1.2 Functional User Requirements (FUR)

EXAMPLE: In principle the COSMIC method can be applied to functional requirements for information processing before they are allocated to software or to hardware, regardless of the eventual allocation decision. For example, it is straightforward to size the functionality of a pocket calculator using COSMIC without any knowledge of what hardware or software (if any) is involved.

1.3 Non-functional requirements

EXAMPLE: Dependability or fault tolerance requirements for aerospace systems are achieved mostly through a combination of redundancy and backup of the physical systems. A function, such as engine monitoring, is implemented on two or more separate embedded computers. This function has a strict timing constraint stated as an NFR: '*each separate computer must respond within a specific time. If any one of the computers repeatedly responds later than the required time, or its results disagree with the others, it must be out-voted*' (by a mechanism specified as a functional requirement). A requirement for fault tolerance that when initially stated may appear as non-functional therefore evolves into FUR that can be measured. The timing mechanism can also be partly implemented in software and this functionality can also be measured (see also [5]).

1.4 Types versus occurrences.

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

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

1.5 Layers.

1.5.1 A layered architecture of embedded real-time software.

EXAMPLE 1: The physical structure of a typical layered software architecture supporting a piece of embedded real-time software is given in Figure 1.1. (Note: simple uni-tasked real-time embedded software may not need a real-time operating system.)

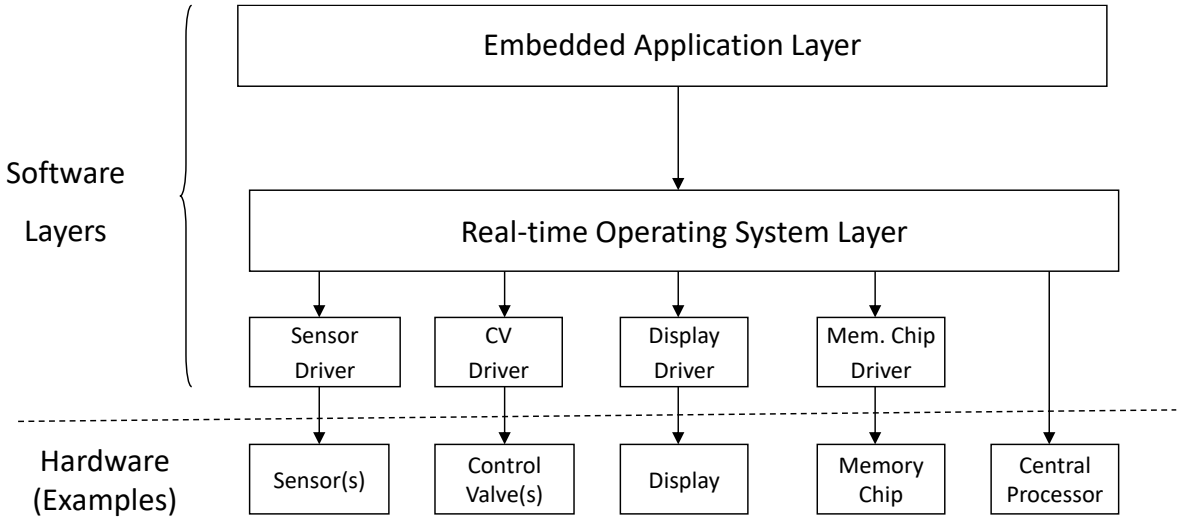


Figure 1.1 - Typical layered architecture for a real-time embedded-software system.

1.5.2 Layered architectures in industry

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

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

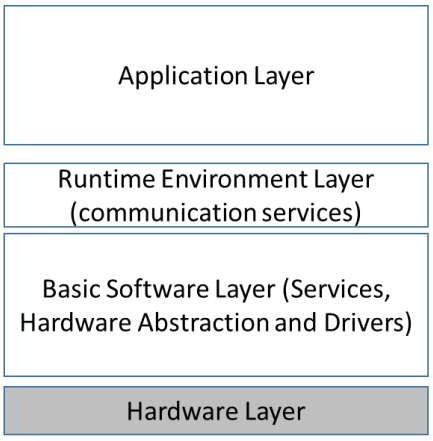


Figure 1.2 – The structure of the AUTOSAR architecture

1.6 Functional users.

1.6.1 Typical functional users.

EXAMPLE 1. When measuring real-time software, the functional users that interact with the software being measured will be typically any of the following:

- a clock or timer,
- sensors (e.g. of temperature, pressure, voltage) that provide input, either when polled, or via interrupts, or by sending their data and/or status at intervals;

- hardware devices that receive output (e.g. a valve or motor actuator, switch, lamp, heater);
- hardware chips, having the ability to trigger functional processes (e.g. watchdog chips);
- 'dumb' hardware memory such as a ROM which can only respond to a request for data;
- communications devices (e.g. telephone lines, computer ports, aerials, loudspeakers, microphones);
- hardware devices with which humans interact (e.g. push buttons, keyboards or displays);
- other pieces of software that supply data to or require data from the software being measured.

EXAMPLE 2. Section 3.1 describes an industry process which is controlled by a programmable logic controller (PLC). The purpose is to measure the size of all the embedded software functionality needed to make the system work, not just the limited view of the functionality as seen by a human operator. The process is started by a human operator pushing a start button. But in this example, given the measurement purpose, the start button is considered to be a functional user, not the operator who pushes it to start the process.

EXAMPLE 3. The embedded software of a mobile device (tablet, smart phone, cellphone) has to interact with several types of buttons, a screen (which may serve as an input device as well as output display), its battery, loudspeaker, aerial, etc. A human user of such a device sees only a small part of the functionality that the software needs to provide its services. So it is possible to measure two functional sizes, depending on the choice of functional users.

1.6.2 Types of functional users.

EXAMPLE 1. One or more buttons (-types)?

Consider a factory that has a moving production line that can be stopped by pushing a button; there are buttons at several different locations along the line. Should the Measurer identify one or several functional users (types)? The answer depends on the functional 'user' requirements that must be measured. The issue from this example is whether pressing the buttons leads to different triggering events and separate functional processes, e.g.

a) Requirements: Any operator may press a button to stop the line in an emergency. When a button is pressed, the system logs the time at which the line was stopped and the button that was pressed. There are many buttons along the line that all have the same effect. As the buttons are subject to the same FUR ('pressing any button must stop the line'), identify only one functional user type and one functional process type to meet these FUR;
b) Requirements as case a) but there is also a requirement for a button in a supervisor's office which is used by the supervisor to stop the line at the end of the work-day. If it has only the same effect as case a), then still identify only one functional user type – again they are subject to the same FUR - and one functional process type.
c) Requirements as case b) but in addition to its use for stopping the line at any time, there is a requirement that when the button in the supervisor's office is pressed AND held down for three seconds, the system stops the line and then produces a

log of the day's stop/start events. (The timing of the three seconds is controlled by the button itself.) We now have two functional users (any stop button on the line, and the supervisor's stop button) and two functional processes. The two functional processes share some functionality (stopping the line), but are invoked by different triggering events (emergency stop, and end-of day stop) and have different effects.

d) Requirements as case c) but there is an additional requirement that the supervisor has a second button that when pressed will start or re-start the line after it had been stopped and log the start time. Now we have three functional user types (any stop button on the line, and the supervisor's stop and start buttons) and 3 functional processes (stop the line, stop the line and produce a report from the supervisor's first button, and start or re-start the line from the supervisor's second button).

EXAMPLE 2: Each wheel of a car has a sensor that obtains the pressure of its tire. At regular intervals, a functional process must obtain the pressure of all four tires. If the pressure is too low or too high - the range of safe pressures is in the software - the software shows which tire has a pressure problem indicated on a diagram of the four wheels on a display screen at the dashboard. The functional users are the four sensors and the four indications on the display screen. However, the four sensors are subject to the same requirement (and idem for the indications on the screen), so identify one functional user type 'sensor' and one functional user type for the indications on the display screen.

1.6.3 Incompatibility of functional users

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

1.7 Levels of decomposition and granularity.

EXAMPLE 1: A group of functional users might be a 'control panel' that has many types of instruments, or 'central systems'. A group of events might be indicated in a statement of functional requirements at a high level of granularity by an input stream to an avionics software system labelled 'pilot commands'.

EXAMPLE: For an example of sizing at varying levels of decomposition and granularity, see the telecoms system example in the Early Software Sizing with COSMIC: Experts Guidelines [6].

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

1.8 Context diagrams.

EXAMPLE: Figure 2.3 shows the context diagram for a simple intruder alarm embedded software system.

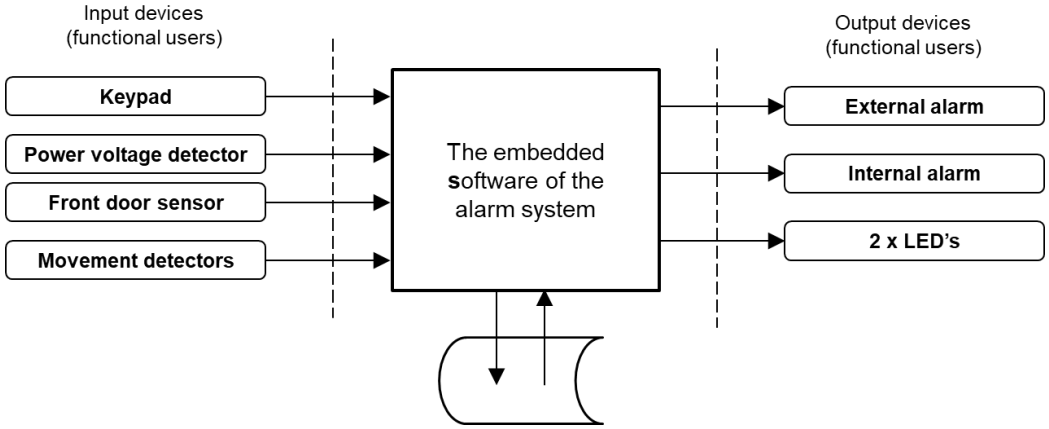


Figure 1.3 - Context diagram for the embedded software of an intruder alarm system.

2. THE MAPPING PHASE.

2.1 Triggering events and functional processes

2.1.1 Identifying a functional process.

EXAMPLE 1: A sensor detects a stimulus to which the software must respond.

- the triggering event is the stimulus that the sensor is designed to detect;
- the functional user is the sensor;
- the sensor generates and sends a message (a data group), which is moved into a functional process by its triggering Entry data movement, informing that the event has occurred; this message may also carry other data about the triggering event.

EXAMPLE 2: A piece of software A must pass a request to a piece of software B for a service.

- software A effectively generates the triggering event when it needs the service from software B by generating the request for service (a data group that provides the input data needed for the service);
- software A is the functional user of software B;
- the request for service message is moved into a functional process in software B by its triggering Entry; the functional process can then provide the service.

EXAMPLE 3: A functional process of a real-time software system may be started by its triggering Entry informing the functional process that a clock (functional user) has ticked. The data group moved conveys data (the tick, perhaps via a single bit) that informs only that an event has occurred.

EXAMPLE 4: A functional process of an industrial real-time fire detection software system may be started by its triggering Entry initiated by a specific smoke detector (functional user). The data group generated by the detector conveys the information 'smoke detected' (an event has occurred) and includes the detector ID (i.e. data that can be used to determine where the event occurred).

EXAMPLE 5: A bar code reader (a functional user) at a supermarket checkout starts a scan when a bar code appears in its window (the triggering event). The reader generates a data group, comprising an image of the bar code that is input to the checkout software. The data group image is moved by a triggering Entry into its functional process. The latter adds the product cost to the customer's bill if the code is valid, sounds a 'beep' to inform the customer that the product has been accepted, and logs the sale etc.

EXAMPLE 6: When a sensor (functional user) detects that the temperature reaches a certain value (triggering event), the sensor sends a signal to initiate a triggering Entry data movement of a functional process to switch off a heater (another functional user).

EXAMPLE 7: A military aircraft has a sensor that detects the event 'missile approaching'. The sensor is a functional user of the software that must respond to the threat. For this software, an event occurs only when the sensor detects something, and it is the sensor (the functional user) that generates a data group to initiate a triggering Entry saying, e.g. 'sensor 2 has detected a missile', plus maybe a stream of data about how fast the missile is approaching and its co-ordinates.

2.1.2 A clock triggering a functional process

EXAMPLE 1: A piece of software must execute a control process each time a clock 'ticks'.

- the clock effectively generates the triggering event by generating a 'tick' (a data group);
- the clock is a functional user of the software;
- the 'tick data group' is moved into a functional process by its triggering Entry to start its task.

EXAMPLE 2: The speedometer software of a car is connected to a rotation measurement sensor located on the drive shaft that measures its revolutions per minute (rpm), and to a key-in sensor, a clock, and a display unit for the driver. The software's persistent storage contains the parameters needed to send messages to a pre-defined variety of display units. The speedometer software is required to capture at key-in time the display parameters and initialize the installed display unit. A clock triggers the software at five millisecond intervals to capture rpm information from the drive shaft, calculate the speed, and send the speed to update the display unit using parameters appropriate for this display unit.

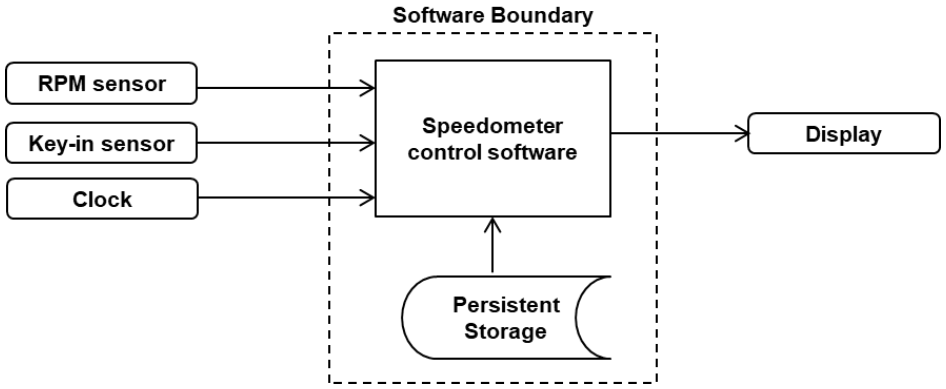


Figure 2.1 – Context diagram for the speedometer software

The context diagram shows the four functional users of the speedometer software, namely three input devices (the rpm sensor, the key-in sensor and the clock) and the one output device (the driver display).

There are two events that need to be responded to by the speedometer software (i.e. are triggering events), They are the key-in event and the 5 millisecond clock tick. Hence the speedometer control software has two functional processes, FP1 and FP2.

- FP1 initializes the speedometer control software on the event of 'key-in' detected by the key-in sensor, which includes reading the parameter data for the display;
- FP2 measures the speed on the event of the tick generated by the clock every 5 ms and sends the speed to the display.

2.1.3 Different processing paths, one functional process.

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

2.2 Objects of interest and data groups

2.2.1 The functional user as object of interest

EXAMPLE 1. The speedometer control software in Example 2 of section 2.1.2 has the 'RPM sensor' as a functional user. This sensor sends a data group to the software, which has one attribute 'current rpm'. The object of interest of this data group could be considered as the drive shaft or the rpm sensor. It is often the case in real-time software that a functional user (the RPM sensor in the example) is also the object of interest of a data group that it sends (i.e. it is sending data about itself).

EXAMPLE 2: A data group entering software from a physical device informs about the current state of the device. In this case the device is the object of interest (and functional user) and the data group conveys its state, such as that a valve is open or closed, leading to the start of a functional process. The physical device is the object of interest. Similarly, a data group output to a device, such as to switch a warning lamp on or off conveys state data about the lamp object of interest.

EXAMPLE 3: Suppose a temperature sensor A sends a measure of the current temperature of a material for processing by a functional process. The sensor provides information about its own state and is thus object of interest of the information data group.

2.1.2 Other examples of objects of interest

EXAMPLE 1: A message-switch software system may receive a message data group as input and route it forward unchanged as output, as per the FUR of the particular piece of software. The attributes of the message data group could be, for example, 'message ID, sender ID, recipient ID, route code and message content'; the object of interest of the message is 'message'.

EXAMPLE 2: A reference data structure, represents objects of interest whose attribute-values are given in tables found in the FUR, and which are held in permanent memory (ROM memory, for instance) and accessible to most of the functional processes found in the measured software.

EXAMPLE: 3: Files, commonly designated as ‘flat files’, represent objects of interest mentioned in the FUR, which are held on a storage device.

2.3 Data attributes.

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

2.4 Data movements.

2.4.1 The various ways of receiving or getting data

EXAMPLE 1. Figure 2.2 shows the various ways in which real-time software can receive or ‘get’ a data group from its functional users (which varies with their capabilities) and from persistent storage.

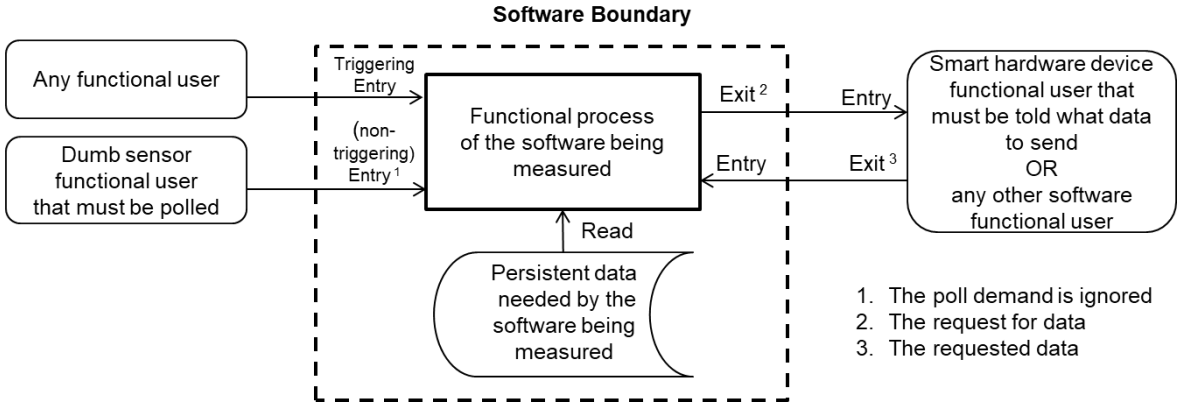


Figure 2.2 - The various ways in which a functional process can receive or get data

2.4.2 Identifying data groups and data movements

EXAMPLE 1: Suppose a functional process sends to one of its functional users, such as an ‘intelligent’ hardware device or another peer piece of software, some parameters for an enquiry or the parameters for a calculation, or some data to be compressed. The response from the functional user is obtained via the functional process issuing an Exit, followed by the receipt of an Entry data movement.

EXAMPLE 2: For a clock-tick event occurring every 3 seconds, identify an Entry moving a data group of one data attribute. The object of interest (and functional user) is the clock, the data group conveys the state of the clock.

2.4.3 Types and occurrences of functional users and data groups.

EXAMPLE 1: A functional process is required to accept different data groups from two different seismometers (functional users) each responding to the same event e.g. a test explosion. Identify two Entries.

EXAMPLE 2: Suppose a process control system for a machine that produces a flat product such as paper or a plastic film. The machine has an array of 100 identical sensors across the direction of movement of the product to detect breaks or holes in the product. The functional process that must check for breaks or holes receives the same data from each sensor. The position of e.g. a hole in the product can be determined from the position in the string of data values sent from the array of sensors, The processing of data from all the sensors in the array is identical. Identify one

functional user for all the sensors and one Entry for the data obtained from all the sensors by the functional process.

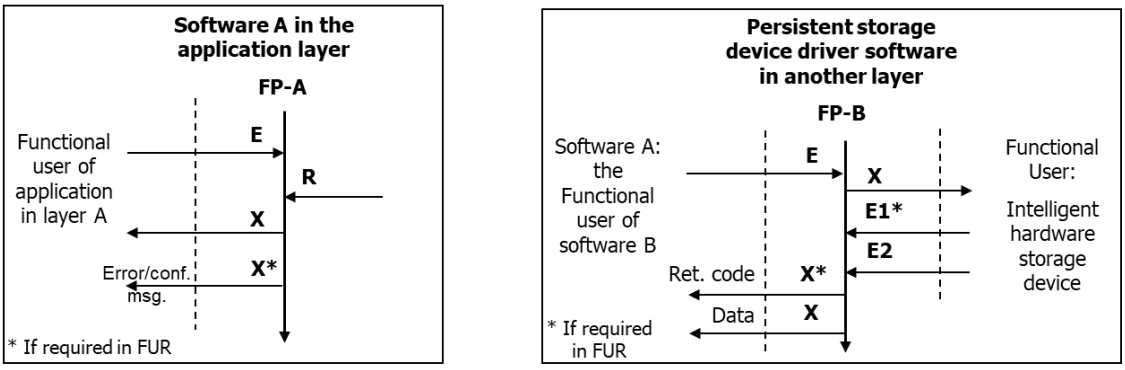
2.4.4 Data groups in a layered software architecture.

EXAMPLE 1: This example concerns the piece of software A in Figure 2.3 that is required to retrieve a stored data group. Consider a separate piece of software B that is the device driver for the intelligent hardware storage which holds the data group that the software A is required to access. (Ignore the probable presence of an operating system for simplicity; the operating system effectively transmits application requests to the device driver software and returns the results of requests.)

The two pieces of software are in different layers in an architecture such as shown in Figure 1.1. Software A is in e.g. the application layer, and software B is in a device driver layer. Physically, there is probably a hierarchical relationship between the two pieces and (ignoring the operating system) a physical interface between software in the two layers, as shown for example in Figure 1.1. However, the models of the functional processes of software A and B are independent of the nature of the relationship between the layers, which may be hierarchical or bi-directional.

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

Suppose that an enquiry functional process FP A of the software A needs to retrieve a stored data group. Figure 2.3 (a) shows the COSMIC model of this enquiry. Figure 2.3 (b) shows the functional process FP B of the software B in the device driver layer that handles the physical retrieval of the required data from a hardware storage device (such as a disk or USB memory stick).



Figures 2.3 (a) and (b) – Solution for a Read issued by software A in the application layer to software B in the device driver layer.

Figure 2.3 (b) shows that the Read request of the software A is received as a triggering Entry to the functional process FP B, which passes on the request as an Exit to the hardware device. The response of the latter depends on the particular hardware device. The device may just return the requested data, shown as Entry E2 in Figure 2.3 b). The device may also issue a separate error message describing the success or the reason for the failure of the request, e.g. 'data not found', or disk error', shown as Entry E1* in Figure 2.3 b). FP B returns the data to the software A as an Exit. FP B also normally issues a 'return code' describing the success or reason for the failure of the request. (Although the return code may be physically attached to the returned data, it is logically a different data group to that of the returned data – it is data about the

outcome of the request process). For FP A no Entry for these messages is identified, as the Read data movement accounts for the retuned data and error messages, as Reads and Writes account for any associated reporting of error conditions. For FP A, an Exit is identified for an error/confirmation message, if required.

Note: in practice, there may be more data movements between the device driver software and the intelligent hardware device than are shown in Figure 2.3 b). For example, this Figure does not show the effect of the device driver measuring a timeout for non-response from the hardware.

EXAMPLE 2: Suppose a functional process of a real-time process control software system is required to poll an array of identical dumb sensors. At the application level, the request for the data by the functional process and the receipt of the data is accounted for by one Entry. (Since the sensors are identical one Entry is identified and counted.)

Suppose further that the request for the data must in practice be passed to a piece of device driver software in a lower layer of the software architecture, which physically obtains the required data from the sensor array as illustrated in the layered architecture of Figure 1.1. The functional processes of the process control software and of the device driver software for the dumb sensors would be as shown in Figures 2.4 (a) and (b) below.

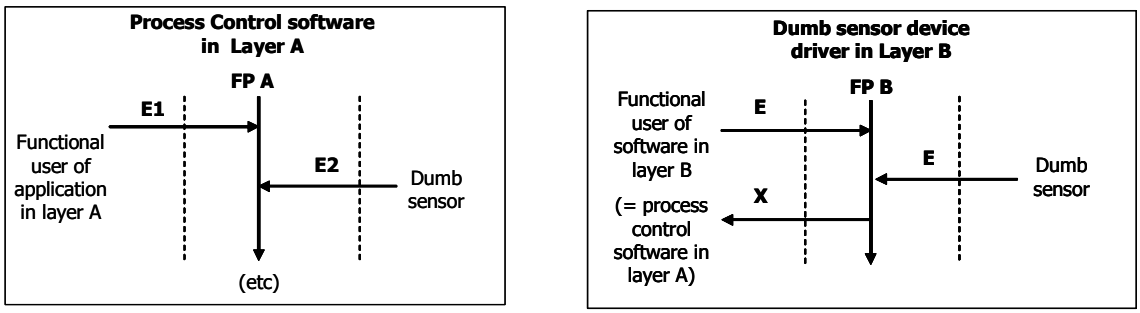


Figure 2.4 (a) and (b) – Solution for a poll of dumb sensors.

Figure 2.4 (a) shows that the software functional process FP A is triggered by an Entry E1 e.g. from a clock tick. This functional process then obtains data via Entry E2 from the dumb sensor array to receive the multiple occurrences of the sensor readings. The dumb sensors are also functional users of the process control software. (The device driver software is hidden at this level.)

Figure 2.4 (b) shows the model for the software that drives the dumb sensor devices. It receives data via an Entry from the process control software (probably in practice via an operating system) as the trigger of a functional process FP B. This functional process obtains the required data via an Entry E from its functional user, the dumb sensor array.

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

The apparent mis-match between the one Entry E2 from a dumb sensor to the process control software and the Entry followed by an Exit data movement of the device driver software is due to the convention that an Entry from a dumb sensor is considered to

include any ‘request to enter’ functionality since the dumb functional user has no capability of dealing with any message from a functional process.

2.5 Measuring the components of a distributed software system.

EXAMPLE: The pieces of software to be measured are assumed to have a ‘client/server’ relationship, i.e. where one piece, the client, obtains services and/or data from the other piece, the ‘server’, in the same or a different layer. Figure 2.5 shows an example of such a relationship, in which the two pieces are major components of the same application. In any such client/server relationship, the FUR of the client component C1 would identify the server component C2 as one of its functional users, and vice versa. The same relationship would exist and the same diagram would apply if the two pieces were separate applications, or if one of the pieces were a component of a separate application.

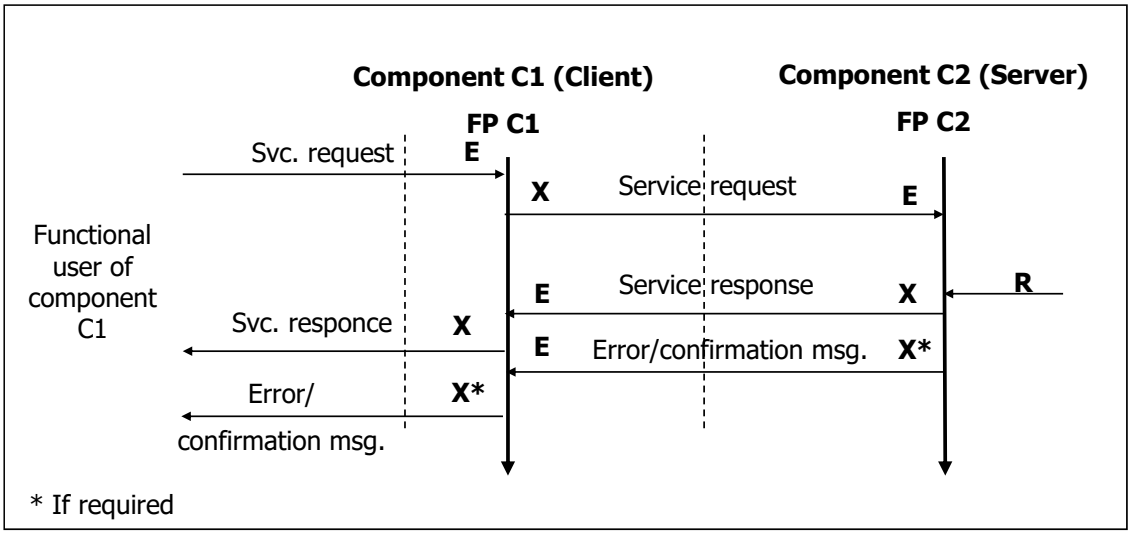


Figure 2.5 – Data exchanges between client and server components.

Physically, the two components could execute on separate processors; in such a case they would exchange data via the respective operating systems and any other intermediate layers of their processors in a software architecture such as shown in Figure 1.1. But logically, applying the COSMIC models, the two components exchange data via an Exit followed by an Entry data movement. All intervening software and hardware is ignored in this model.

Figure 2.5 shows that a functional process FP C1 of the client component C1 is triggered by an Entry from a functional user (such as a human) which consists, for example, of the parameters of the enquiry. The FUR of component C1 will recognize that this component must ask the server component C2 for the required data, and must tell it what data group is required.

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

as an Entry. FP C1 then passes the data group on as an Exit to satisfy the enquiry of its functional user.

Taking into account the possible error/confirmation message issued by the client, this enquiry therefore requires 6 data movements (i.e. 6 CFP) to satisfy the enquiry request for component C1 and 4 CFP for component C2. This compares with the 4 CFP (1 x E, 1 x R and 2 x X) that would have been required for component C1 if it had been able to retrieve the data group from persistent storage via a Read as shown in Figure 2.3 (a).

Component C2 will probably use the services of some storage device driver software in another layer of the software architecture to retrieve the data from the hardware, as in Figure 2.3 (b).

2.6 Re-use of software.

EXAMPLE 1: Several functional processes in the same software being measured may need to obtain data from the same sensor (common movement of same data group) or may need to carry out the same scale conversion calculation, e.g. from Fahrenheit to Centigrade (common data manipulation). Measure this common functionality.as part of each functional process that requires it.

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

EXAMPLE 3: For a computer operating system, the user that starts the operating system is a bootstrap program that is started when the computer power is switched on. This user must be ignored.

EXAMPLE 4: A modern vehicle has a distributed system of Electronic Control Units (ECUs) to control many functions, e.g. engine management, brakes, air-conditioning, etc. In the AUTOSAR architecture, in a distributed system, the ‘Network Management’ (NM) module, which is always running, is responsible for activating the ECUs that are connected together via a network (‘bus’). This NM module also handles the coordinated switching between the ECU operating states: Normal Operation, Low Power and Sleep. Therefore, it is the NM that wakes up or puts to sleep ECUs. When measuring any ECU application software, this NM functionality is being reused and should be ignored.

2.7 Error or fault messages

2.7.1 General examples

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

EXAMPLE 2. The FUR of the software of Figure 2.4 may also state that the FP’s A and B must handle an error condition when the device driver software fails to obtain the data from one or more of the array of dumb sensors. A dumb sensor cannot, by definition, issue an error message. The device driver FP B will, most likely, obtain a

string of values from the array of dumb sensors, e.g. state 1, state 2, state 3, no response, state 5, no response, state 7, etc. and will issue this string as an Exit to the FP A of the application where it is received as an Entry. No separate error message should be identified as an Exit from FP B of the device driver software, nor as an Entry to FP A of the process control application.

EXAMPLE 3: Suppose a busy airport has multiple radar stations to control incoming air traffic on a runway. In a particular case, the radar station software is asked to report the number of aircraft in the 45-to-135-degree quadrant around runway 09L. Two of the radar stations report 2 aircraft and the third radar station reports 5. The software will report 2 aircraft to the traffic controllers as a majority response to the request, but will also report a separate warning message to the controllers and to the radar engineers that this is a majority decision and that one station reports 5 incoming aircraft.

The object of interest of the first message is 'Incoming air traffic on runway 09L'.The object of interest of the error message could be 'Radar station disagreement'

2.7.2 Error messages intended for hardware or software functional users.

EXAMPLE 1. 'In-line' error messages that are intended for hardware or software functional users. If a data group describing a particular object of interest in a message issued by the software being measured may include an indication of a fault or of an error in place of the normal valid data, this fault/error indication describes the same object of interest as the normal valid data. Hence this data group is moved by only one Exit, i.e. the fault/error indication is <u>not</u> identified as a separate Exit.

> Output= {VAL1, VAL2, VAL3, …, VALn, ERROR} where 'VAL' indicates a valid value.

EXAMPLE 2. Separate error messages that are intended for hardware or software functional users. If the software being measured issues the reason for a fault or error condition as a separate message, then a separate Exit may be identified. To be measured as a separate Exit, the error message must describe a different object of interest than the message containing the normal valid data and/or the error message must be intended for a different functional user than the user that would receive the normal valid data. The general case is:

Output1= {VAL1, VAL 2, VAL3, …, VAL n, ERROR}

Output2= {Sensor_failure, Internal_error, General Failure, …}

2.8 Changes to software.

See section 3.2 for an example of changes to software.

2.9 Data manipulation

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

3. COMPREHENSIVE EXAMPLES

3.1 PLC software for controlling a process in a chemical factory

Requirements

A process in a chemical factory is controlled by a PLC. The process consists of filling a tank with a liquid, heating the liquid and then emptying the tank when a temperature is reached that is pre-set in the temperature sensor device.

In the following description of the requirements of the process (system) control we assume that all mentioned functionalities are allocated to the PLC software, unless stated otherwise.

- The process is started by a human operator pressing a start button connected to the PLC which controls all the subsequent steps.
- The software issues a command to open the inlet valve of the tank and the tank fills with liquid under gravity.
- When the tank is full (‘high level reached’ is detected by the high level sensor) the software receives a message from this sensor and sends commands to close the inlet valve and to start the heater to heat the liquid.
- When the software is informed that the pre-set temperature is reached, it sends commands to stop the heater, open the outlet valve and start the pump to empty the tank.
- The pump continues emptying until ‘low level reached’ is detected by the low level sensor. On receipt of a message from this sensor, the software sends a command to stop the pump.
- During the entire the process, the process status (‘Filling’, ‘Heating’, ‘Pumping’) is shown on an operator display controlled by the software. When the process is finished, the software causes an audible alarm to sound and the message ‘Process finished’ is shown on the display.
- When the process is started and whilst the process is running, the PLC software polls the valves, the heater and the pump asking for their status at regular intervals to detect any fault conditions.
- If the PLC software is informed that an error is detected, it starts the audible alarm and displays a message to the operator showing the device(s) concerned. If an operator receives an error message, the operator deals with it manually, outside the software system.
- The polling frequency is determined by signals (‘ticks’) from a clock.

Context diagram

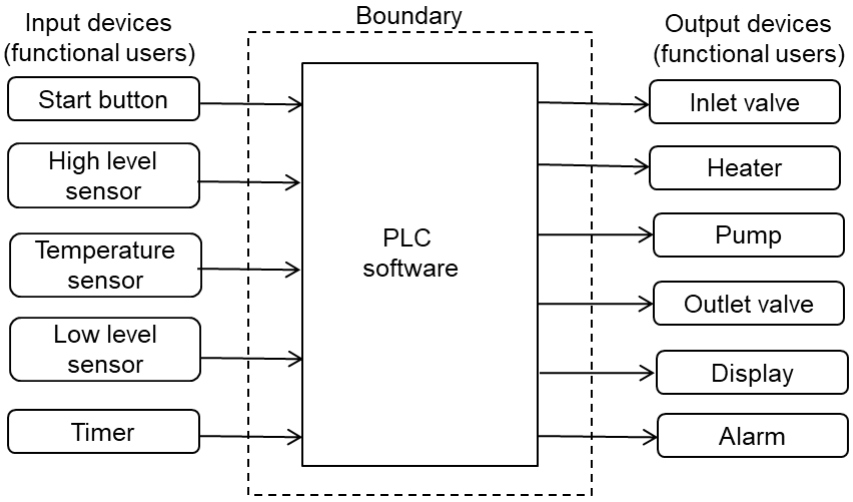


Figure 3.1 - Chemical Factory Process: PLC software Context Diagram

Analysis

The measurement assumes all the hardware devices that interact directly with the software are its functional users, as shown in the context diagram. The PLC does not have an operating system.

In the requirements as described above, the software is not decomposed in any way and is not a component of another piece of software. The level of granularity of the requirements is at the 'functional process level of granularity', i.e. the level of individual functional users and events (rather than groups of these). The triggering events and corresponding functional processes are:

Table 3.1 - Chemical Factory, triggering events and functional processes

Triggering event	Functional user that initiates the functional	Corresponding Functional process
Start button pushed	Start button	Start process/Fill tank
High level reached	High level sensor	Heat liquid
Pre-set temperature reached	Temperature sensor	Stop heating/Empty tank
Low level reached	Low level sensor	Finish process
Clock tick (= time to poll)	Clock	Fault check

The data groups consist of the signals from the start button, the sensors and the clock to the software and the signals from the software to the actuators, valves and the devices for the operator.

The object of interest of each data group entering the software is the functional user that sent the group (i.e. the functional user is sending data about itself). Similarly the object of interest of each data group that leaves the software is the functional user that receives the group (i.e. the functional user is being sent data about itself). For instance, the data movements starting or stopping the pump move data groups that specify the (desired) states of the pump. The pump is therefore the object of interest of these data groups.

The software determines the process status to be displayed from the triggering Entry for each functional process (except the Fault Check process). For instance, from the

start button signal the software determines that the current status is 'Filling' and displays this status.

The functional processes of the PLC software are as follows. The data movements (abbreviated as DM), the data groups moved and an explanation are shown for each functional process. We assume that the devices that the software polls to determine their status are ‘dumb’, i.e. the software inspects the state of these devices, which requires only one Entry per device type for the poll.

Functional process: Start process/Fill tank

DM	Functional User / Object of interest	Data Group
Entry	Start button	Start process message
Exit	Inlet valve	Open inlet valve command (to start entering liquid)
Exit	Clock	Start clock command (for fault detection at regular intervals)
Exit	Display	Display status command (‘Filling’)

The size of this functional process is 4 CFP.

Functional process: Heat liquid

DM	Functional User / Object of interest	Data Group
Entry	High level sensor	Tank full message
Exit	Inlet valve	Close inlet valve command (to stop liquid entering)
Exit	Heater	Start heating command
Exit	Display	Display status command (‘Heating’)

The size of this functional process is 4 CFP.

Functional process: Stop heating/Empty tank

DM	Functional User / Object of interest	Data Group
Entry	Temperature sensor	Pre-set temperature reached message
Exit	Heater	Stop heating command
Exit	Outlet valve	Open outlet valve command
Exit	Pump	Start pump command (to start emptying the tank)
Exit	Display	Display status command (‘Pumping’)

The size of this functional process is 5 CFP.

Functional process: Finish process

DM	Functional User / Object of interest	Data Group
Entry	Low level sensor	Low level reached message
Exit	Pump	Stop pump command
Exit	Outlet valve	Close outlet valve command
Exit	Display	Display status command (‘Finished’)
Exit	Audible alarm	Sound alarm command (to inform operator)
Exit	Clock	Stop clock command

The size of this functional process is 6 CFP.

Functional process: Fault check

DM	Functional User / Object of interest	Data Group
Entry	Clock	Clock tick (to start fault check process)
Entry	Inlet valve	Inlet valve status (from polling)
Entry	Outlet valve	Outlet valve status (from polling)
Entry	Heater	Heater status (from polling)
Entry	Pump	Pump status (from polling)
Exit	Audible alarm	Start alarm command (if device fault(s) detected)
Exit	Display	Display faulty device(s) command (if there is a fault[2])

The size of this functional process is 7 CFP.

The total software functional size of the PLC software is 4 + 4 + 5 + 6 + 7 = 26 CFP.

3.2 Measurement of a change to the PLC software

Requirements

In the Example of section 3.1 it has been decided to remove the audible alarm device and to adapt the software accordingly.

Analysis

The commands from the software to the alarm device can be removed, i.e. the Exit data movements of the audible alarm data group in the last two functional processes must be removed. The functional size of the change is 2 CFP. The resulting software functional size will be 24 CFP once the change is made.

3.3 Timing functionality

Measuring timing functionality requires clear specifications on what functions are allocated to the hardware part of the functionality, and what are specifically allocated to the software part.

EXAMPLE 1. Timing functionality needed, for example, to control a pre-set time interval can be implemented in several ways, with different divisions between the hardware and software:

- A hardware clock generates pulses (‘clock ticks’) at regular defined intervals each of which triggers a functional process of the software. The software keeps track of the pulses, may convert them to seconds or minutes if needed, and increments the elapsed time until the pre-set time is reached.
- A hardware timer both generates and keeps track of the pulses and transforms them into seconds, minutes etc., in an internal register if needed. The software may start the timer which informs the software when the desired time is reached. This mechanism is used in the next Example 2.

[2] In this case, the display is shown on the context diagram as the functional user (not the human operator that reads the display). This Exit is therefore not an ‘error/confirmation message’, as defined in the Measurement Manual.

EXAMPLE 2. A web-server must access a customer information system to retrieve some customer data. In addition to handling this request, the server starts a monitoring process to check that the request for customer information is handled within a set time. The aim is to ensure that the human user who seeks the customer information is not left hanging indefinitely if the customer information system fails to respond. Figure 4.2 shows a message sequence diagram for a simple example (no re-tries) of how this might be done via the interactions of the functional processes of the four participants, which are functional users of each other:

- Web-server (functional user 1)
- Customer information system (functional user 2)
- Monitor (functional user 3)
- Real-time Timer (functional user 4)

The web-server, after issuing the request to the customer information system, issues another message to the monitor, requesting it to respond if the given time-out period is exceeded. If the web-server receives the data from the customer information system within the time-out period, it tells the monitor to stop monitoring. Otherwise, if the web-server first receives a reply from the monitor that the time-out period has passed, the web-server issues a time-out message to the functional user that requested the customer data.

The monitor logs the request from the web-server and issues a request to the real-time timer, asking for a response within the given time-out period. (The timer may be implemented in hardware and/or software of the RTOS; it does not matter.) The monitor next receives either a message from the web-server to stop monitoring, or a message from the timer that the time-out period is complete. If the latter, the monitor sends a time-out message to the web-server. On completion, the monitor cancels the request from its log.

In Figure 4.2, the data movements of the timing functionality are shown as red dashed lines. The functionality requires 4 CFP for the web-server to request the monitoring function, in addition to the 2 CFP to obtain the customer data. The monitor requires 8 CFP to fulfill its requirement. (The ‘delete request’ Write is counted only once, although it may be issued at two alternative times, depending on whether the customer data are returned within the given time-out period or not.)

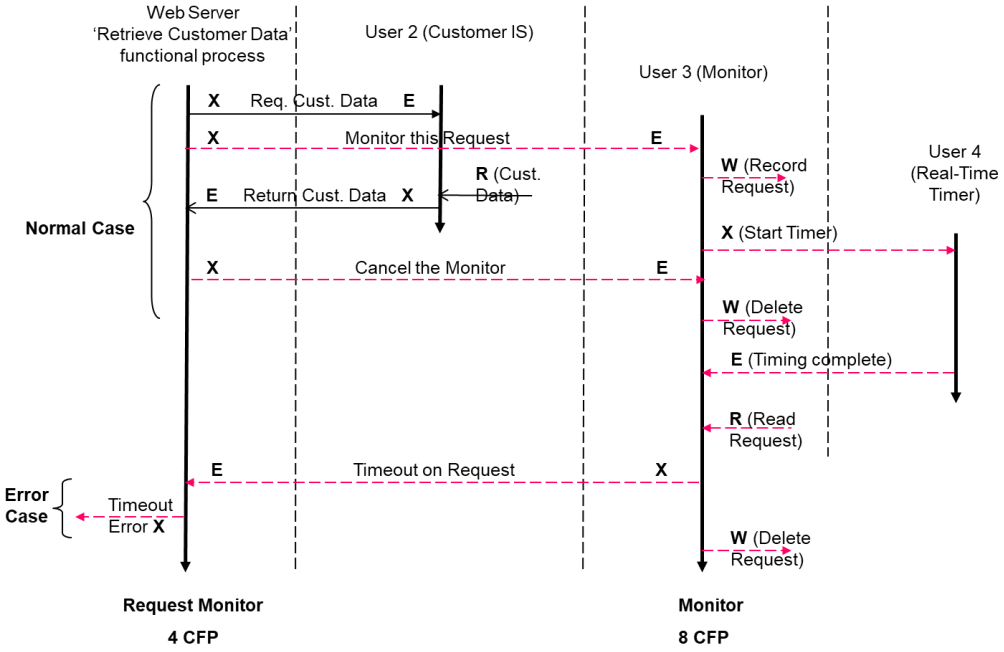


Figure - 3.2 The functionality for a web-server to monitor 'time-out'

3.4 Intruder alarm system

Outline statement of requirements

This case concerns a domestic intruder (or burglar) alarm system. Its main purpose is, when it is activated, to start one or two sirens (devices that make a loud noise) if a sensor detects a movement inside the house or if the front door is opened.

We do not have a statement of requirements, so we deduce the functionality available to normal house occupants and allocated to software from knowing how to use the system and by examining it physically. We are not interested in the functionality provided for the alarm maintenance engineer, nor in the functions to set-up the system when it is first installed.

The software supports the alarm system's human interface via a keypad and red/green LED's. The software also accepts data from a device that can sense whether the main front door of the house is open or not, and from several internal movement detectors. (The alarm system can handle any number up to 10 movement detectors. The number does not matter for this analysis as they are all identical and equivalent.) The alarm system also controls an internal and an external siren.

The alarm system is always powered 'on', but is not 'active', i.e. the movement detectors and the front door sensor are not working, unless the system is activated by the house occupant (the person normally resident in the house). When the system is activated, either the software waits in a state where it can receive signals from these sensors, or the software polls the sensors to obtain their state. We do not know which process is used and it does not matter for the functional size measurement.

To activate and de-activate the alarm system, the house occupant must enter the correct PIN (Personal Identification Number) within a pre-set time. The PIN is stored by the software and can be changed, so there must be some persistent storage. When the first digit of a PIN is entered, the internal siren is started; this siren is stopped on

entry of all digits of the correct PIN. If the wrong PIN is entered three times or if the correct PIN is not entered within the pre-set time, the external siren is also started.

There is a battery to provide continuity if the mains electricity power supply fails, so there must be a power voltage detector.

The green LED is illuminated when power is switch on. If a siren is started or if the mains power fails, the green LED is switched off and the red LED is illuminated.

As certain functions must be completed within pre-set times, there must be a clock or timer mechanism. For example, if the alarm system is activated before leaving the house, the occupants must leave and close the front door within a pre-set number of seconds; if not, the sirens are started. The external siren must not continue for more than the legal limit of 20 minutes.

We do not know how the clock/timer is implemented but assume a software implementation for simplicity, which starts whenever needed. The functionality to keep track of elapsed times is then a form of data manipulation, which we can ignore.

Measurement strategy parameters

Purpose of the measurement: To measure the functional processes of the embedded application software available to the house occupant for normal operation.

Measurement scope: The alarm system embedded application software functions available to the house occupant for normal operation. (We are not interested if there is an operating system)

Functional users: A context diagram shows the hardware functional users and how they interact with the software. Note that the movement detectors are all functionally identical, so do not need to be distinguished. The human user of the alarm system, referred to as ‘the occupant’ is not a functional user; he/she interacts with the application only via the keypad and the audible and visual signals.

Layer: Application layer.

Level of decomposition: ‘level 0’, i.e. no decomposition.

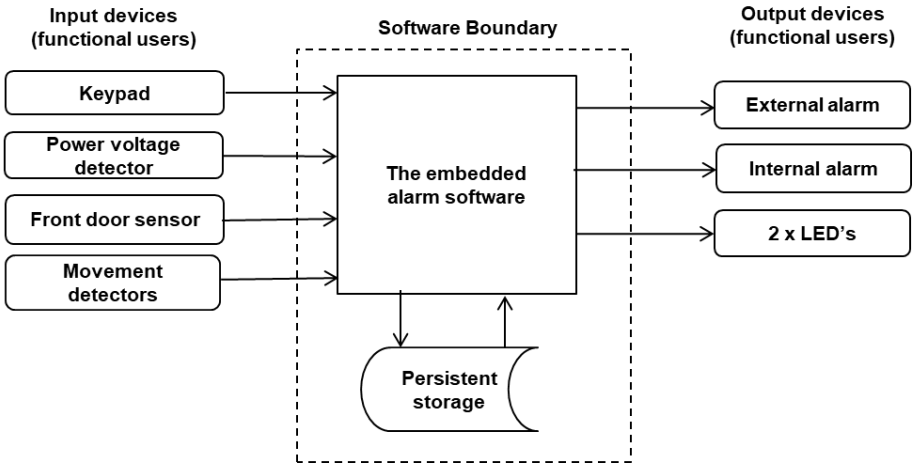


Figure 3.3 – The Intruder Alarm System Context Diagram

The functional processes:

After initial set-up, the alarm system application provides the occupant with nine functional processes. These can be identified by considering the events that the software must respond to.

1) The occupant wishes to change the existing PIN.

2) The occupant wishes to leave the house and activate the alarm system.

3) The front door sensor detects that the door has been opened whilst the alarm system is activated.

4) The occupant wishes to activate the alarm system whilst he/she is in the house, e.g. when retiring at night, out of range of the movement detectors.

5) The occupant wishes to deactivate the alarm system when inside the house, e.g. when getting up in the morning before moving within range of the movement detectors.

6) A movement detector signals a movement whilst the alarm system is activated (which starts the internal siren).

7) The occupant wishes to cancel the siren(s) and to deactivate the alarm system by entering the correct PIN following events 3) or 6).

8) The power voltage detector signals failure of the mains electrical supply

9) The power voltage detector signals restoration of the mains electrical power supply.

Analysis of an example functional process:

We analyze the event 3) on the list above (the front door is opened whilst the alarm system is activated). When the front door sensor detects this event, the internal siren starts; the correct PIN code must then be entered within a pre-set time to de-activate the system and to stop the internal siren. If the PIN code isn't entered before the pre-set time, or the wrong code is entered more than three times, the external siren also starts. The functional process has the following data movements.

Functional process: Possible intruder detected. Triggering event: Door opens whilst alarm system is activated.

DM	Functional User / Object of interest	Data Group
Entry	Front-door sensor	'Door open' message (triggering Entry)
Read	/ Occupant	PIN (from persistent storage)
Exit*	Green LED	Switch 'off' command
Exit*	Red LED	Switch 'on' command
Exit	Internal siren	Start noise command
Entry	Keypad	PIN (If the wrong code is entered, the user may enter the PIN two more times but the process is always the same so it is only measured once.)
**	Green LED	Switch 'on' command (after successful entry of PIN)
**	Red LED	Switch 'off' command
**	Internal siren	Stop noise command (after successful entry of PIN)
Exit	External siren	Start noise command (after three unsuccessful PIN entries, or if the PIN is not entered in time)
Exit	External siren	Stop noise command (after 20 minutes, a legal requirement)

NOTE: (*) The green and red LEDs are different types as they are subject to different functional user requirements, therefore identify two functional user types. (**) These are repeat occurrences of the Exits to the LED's and the internal siren earlier in the process, but with different data values ('on' instead of 'off', and vice versa).

The total size of this functional process is 8 CFP

3.5 Cooker software

A simple cooker can be set to cook for multiples of one minute, provided its door is closed.

Requirements

- When the power is switched on the cooker is in a ‘standby’ state. The cooker software can receive input from the door and from a start button, and can send signals to switch an internal light, and the heater, on or off. The software can also send signals to a timer to set the cooking time and can receive a signal from the timer when cooking is complete.
- Cooking starts with pressing the start button provided the door is closed. If the door is open pressing the start button has no effect.
- Opening the door during cooking turns the heater off.
- Whilst cooking or whilst the door is open, the cooker light is on.
- The cooking time is set in multiples of a minute.
- Each time the start button is pushed adds one minute to the cooking time.
- When the timer stops, either because the door is opened whilst cooking is in progress, or because the timer signals that cooking is completed, the timer resets itself to zero.
- The initialization of the cooker software is out of the scope of this case. Context diagram

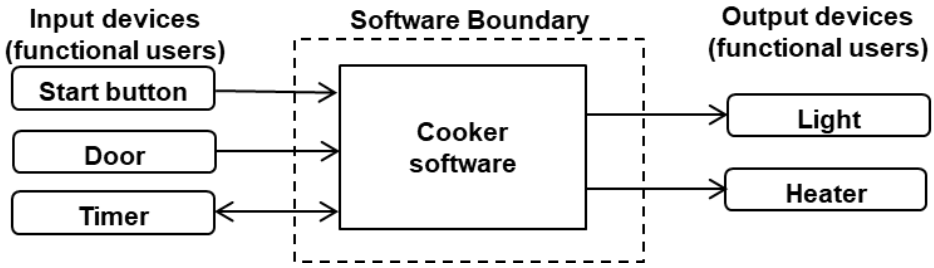


Figure 3.4 – Cooker Context Diagram

The state transition diagram of the cooker is shown in Figure 3.5. Boxes represent states and arrows represent the transitions from one state to another (possibly the same state). The events which cause the cooker to move between its states are (occurrences of) triggering events. These are prefixed by ‘TE’ and the functional users that sense the events by 'FU'.

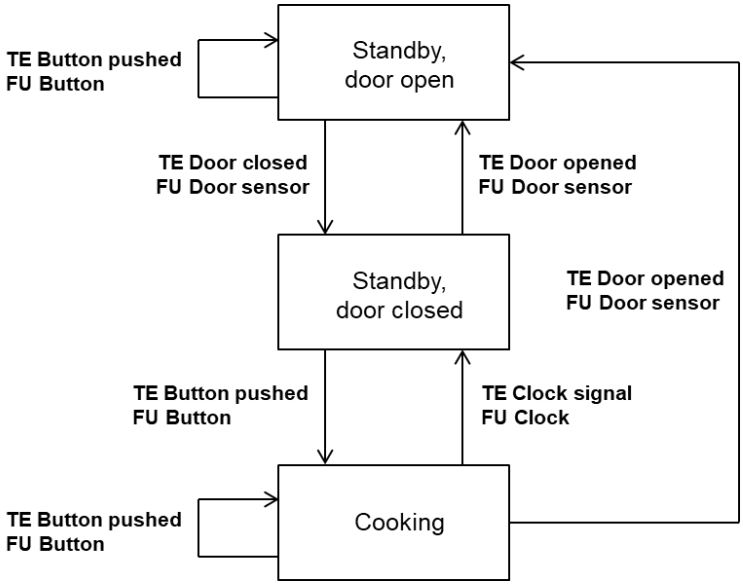


Figure 3.5 – Cooker State Transition Diagram

Analysis

The functional users of the cooker software on the input side are the door sensor and the push button. On the output side the functional users are the cooker light and the heater. The functional user that is on both the input and the output side is the timer. The object of interest of each data group entering the software is also the functional user that sent the group (i.e. the functional user is sending data about itself) and similarly the object of interest of each data group that leaves the software is also the functional user that receives the group (i.e. the functional user is being sent data about itself).

The events that actually trigger the software to start a functional process are as follows. As there is here a one-one correspondence between triggering events and functional processes, the same name is used for both.

Table 3.2 - Cooker, triggering events and functional processes

Triggering event	Functional user that initiates the functional process	Functional process
Door closed	Door sensor	Door closed
Button pushed	Push button	Button pushed
Timer signal (cooking ended)	Timer	Timer signal (cooking ended)
Door opened	Door sensor	Door opened

The functional processes of the cooker are as follows:

Functional process: Door closed

DM	Functional User / Object of interest	Data Group
Entry	Door sensor	Door closed signal (triggering Entry)
Exit	Cooker light	Switch ‘off’ command

The size of this functional process is 2 CFP.

Functional process: Button pushed

DM	Functional User / Object of interest	Data Group
Entry	Start button	Button pushed signal (triggering Entry)
Entry	Door sensor	Get door status
Exit	Heater	Heater ‘on’ command (if door closed)
Exit	Cooker light	Light ‘on’ command (if door closed)
Exit	Timer	Start or increment cooking time command (each push adds one minute to the cooking time if door closed)

The size of this functional process is 5 CFP.

Functional process: Timer signal (cooking ended)

DM	Functional User / Object of interest	Data Group
Entry	Timer	Timing stopped signal (triggering Entry)
Exit	Heater	Switch ‘off’ heater command
Exit	Cooker light	Switch ‘off’ cooker light command

The size of this functional process is 3 CFP.

Functional process: Door opened

DM	Functional User / Object of interest	Data Group
Entry	Door sensor	Door open signal (triggering Entry)
Exit	Cooker light	Switch ‘on’ cooker light command
Exit	Heater	Switch ‘off’ heater command
Exit	Timer	Stop timer command

The size of this functional process is 4 CFP.

The total functional size of the cooker software in the scope is 2 + 5 + 3 + 4 = 14 CFP.

Discussion

Note an important point about interpreting state transition diagrams. Not all state transitions correspond to separate functional processes. In this example there are seven state transitions but only four functional processes. Only events detected by or generated by a functional user external to the software can trigger a functional process. Each functional process must deal with all states and state combinations that it can encounter when responding to a given triggering event.

As an example, the triggering event ‘button pushed’ can occur when the cooker is in each of the three states. The event of the button being pushed takes place in the external world of the hardware and is entirely independent of the state of the machine. The one functional process that must handle the ‘button pushed’ event responds in three ways dependent on the state of the machine at the time the button is pushed namely:

- In the ’standby, door open’ state, it does nothing, i.e. it stops after having found that the door is open;

- In the ‘standby, door closed’ state, it sends signals to start the heater and switch on the light, and to start the timer for one minute of cooking;
- In the ‘cooking state’, it executes the same data movements as in the previous state but since the heater has already started and the light is already on, the effect is only to add one minute to the total cooking time.

In this example, we have assumed that the cooker can perform its functions by simply checking if the door is open or closed. In a more complex case, software may need to record the state of the machine and to update it in persistent storage every time the state changes. This would avoid the need for the software to determine the state of the machine each time a new event is signaled.

Similarly, the ‘door opened’ event can occur when the machine is in two states. The one corresponding functional process must deal with the two states.

3.6 Tire-pressure monitoring system

Requirements

- A tire-pressure monitoring system (TPMS) monitors the pressure of each of the four tires of a car.
- Each wheel has a sensor which obtains the pressure of its tire.
- As soon as the car’s electrical power supply is turned on, a clock activates the TPMS software once per second to retrieve the status of the four sensors, whether the car is moving or not. The sensors return their status, consisting of the sensor id (which identifies the particular wheel) and tire pressure.
- If the pressure is too low or too high - the values are in the software - the TPMS turns on the relevant red warning LED(s) at the dashboard (the sensor location is therefore relevant).
- If the pressure becomes normal again, the TPMS switches off the relevant red warning LED(s) at the dashboard.
- The TPMS electronic control unit (ECU), the clock, the tire pressure sensors and the dashboard LED’s are coupled by a CAN-bus (CAN = controller–area network).

The purpose of the measurement is to size the functionality of the TPMS.

Context diagram

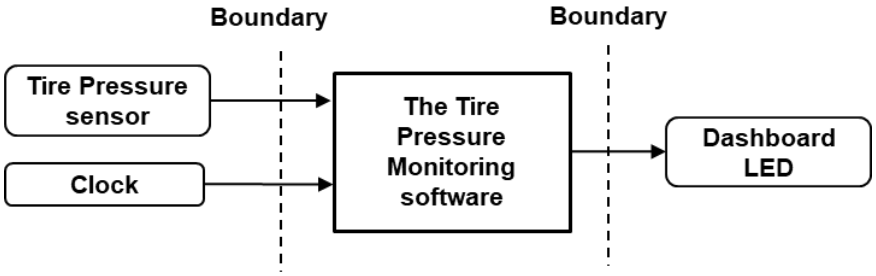


Figure 3.6 – TPMS, context diagram

Analysis

The four sensors are subject to the same FUR (‘obtain tire pressure’), therefore one functional user must be identified that represent these four occurrences: ‘Tire pressure sensor’ (short: Sensor). The same applies to the four LEDs (FUR: ‘turn on/off’), i.e. one functional user ‘Dashboard LED’ (short: LED). The third functional user is the clock.

The Sensor functional user sends the sensor ID and its value, the clock sends the clock signal, the LED functional user receives the LED ID and ‘on’ or ‘off’, all data that describe the functional user concerned. The three functional users are therefore objects of interest.

The CAN-bus controllers form a collection of software that together provides a cohesive set of services that the TPMS software can use and are therefore in a software layer that is separate from the layer in which the TPMS software resides. The network controllers are therefore not in the scope of this measurement. Note that if they were within the scope of the measurement, they must be measured separately as the controllers are software in another layer.

The software must respond to one triggering event, the clock signal that is sent every second, so consists of one functional process:

Table 3.3 - TPMS, triggering events and functional processes

Triggering event	Functional user that initiates the functional process	Functional process
Clock signal	Clock	Start TPMS software

There is no requirement to store or retrieve any persistent data. The table shows the data movements, the data groups moved for the functional process. Further explanation follows.

Functional process: Start TPMS software

DM	Functional User / Object of interest	Data Group
Entry	Clock	Start monitoring signal (triggering Entry)
Entry	Tire pressure sensor	Obtain tire pressure
Exit	Warning LED	Switch ‘on/off’ LED (if needed)

The size of this functional process is 3 CFP.

3.7 Automation of sizing real-time requirements

EXAMPLE. Automation of the measurement of requirements for real-time embedded software of vehicle Electronic Control Units modeled with the Matlab Simulink tool is described in [7] and n [8]. A concise English description of the method, copyright Renault, is available from the download section of www.cosmic-sizing.org [9].

Automation of the measurement of requirements expressed in UML (not specifically of real-time software) is described in [10].

3.8 Measurement of data manipulation-rich real-time software

EXAMPLE An example in this section illustrates that the assumption is reasonable that data manipulation functionality (or ‘algorithms’) of real-time software can be accounted for by the COSMIC method. The example does not, of course, prove that the assumption is always reasonable. For ways in which to deal with software for which it is known that certain areas of the functionality have a high concentration of data manipulation, see section 3.2.3 of this Guideline.

The distribution of algorithms in some avionics software

A large component of the software of a very complex real-time avionics system was measured using the COSMIC method [11]. The total size of the requirements (held in a modelling tool) for the component was over 8000 CFP. Implementation required over 80,000 lines of source code in the Ada language.

This one system component consisted of 33 sub-components. Within each sub-component, the number of lines of Ada code associated with each data movement was also counted. This is known as the 'NOLA', for 'number of lines of algorithm'. Hence the 'NOLA per Data Movement' could be calculated for each of the 8000+ data movements.

Figure 3.7 shows a histogram of the frequency of the 'NOLA per Data Movement', for all except five of the data movements. The five data movements with exceptionally high NOLA had 28 (x2), 36, 40 and 138 NOLA. (Example: the histogram shows that 20 of the 8000+ data movements had seven NOLA.)

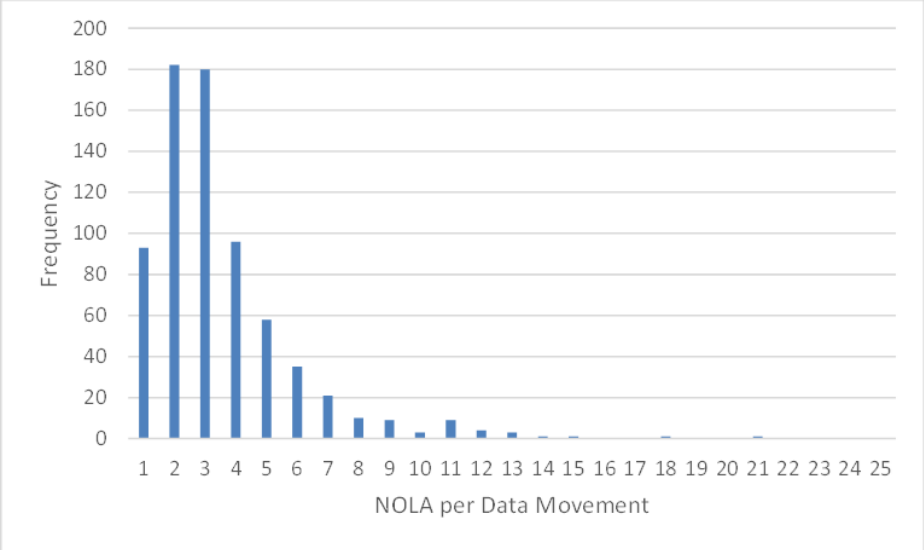


Figure 3.7 - Frequency of NOLA per Data Movement

The following parameters were derived from these data:

Parameter	Value
Median NOLA per Data Movement	2.4
Mean average NOLA per Data Movement	3.5
Data Movement upper size limit which accounts for 95% of the total NOLA	8 CFP
Data Movement upper size limit which accounts for 99% of the total NOLA	14 CFP

The data and the analysis indicate that the NOLA per Data Movement values have a limited range, apart from a very few exceptions. This finding supports the COSMIC method assumption that a count of data movements reflects the amount of data manipulation and thus is a good reflection of the functional size, at least for this particular piece of real-time software.

3.9 Sizing the memory requirements of vehicle Electronic Control Units

EXAMPLE. The study in [12] describes the application of the COSMIC method to size the software embedded in Electronic Control Units of Saab cars, manufactured in Sweden. The purpose of the study was to examine the relationship between the

COSMIC-measured functional size and the resulting memory space needed by the object code, measured in bytes. An extremely good linear correlation was found.

In the paper, the authors state: 'This paper shows that it is possible to obtain accurate code size estimates even for software components containing complex calculations, as long as the components contain similar complexity proportional to the number of component interfaces.'

Renault [12] also reported a good correlation of code size in bytes versus COSMIC-measured functional size in units of CFP, as in the graph below.

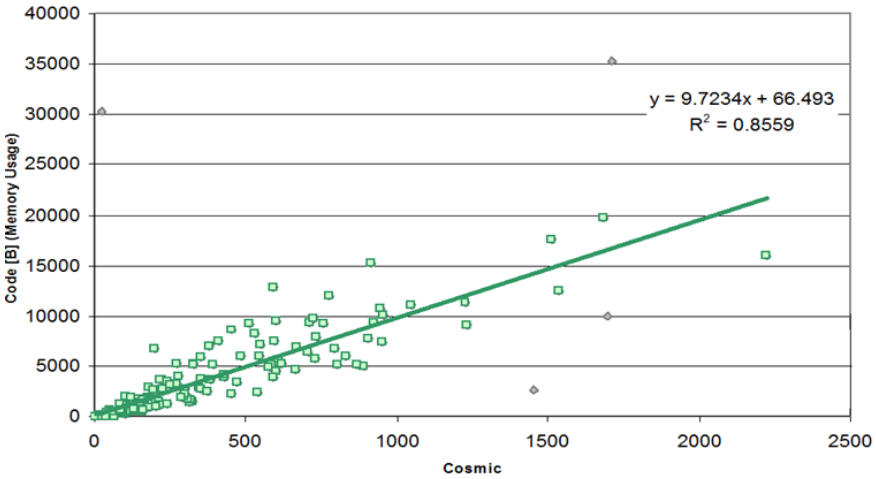


Figure 3.8 - Code size (bytes) versus COSMIC functional size (CFP) [12]

REFERENCES

All the COSMIC documents and the documents indicated by (*) listed below, including translations into other languages, can be obtained from the download section of www.cosmic-sizing.org.

[1] COSMIC, Measurement Manual, v5.0 Part 1, Part 2, Part 3 (*)

[2] ISO 14143:2007 Software Engineering – Software Measurement – Functional Size Measurement, Part 5, Determination of Functional Domains for Use with Functional Size Measurement.

[3] Toivonen, H., Defining measures for memory efficiency of the software in mobile terminals International Workshop on Software Metrics, 2002.

[4] COSMIC, Guideline for 'Measurement Strategy Patterns (*)

[5] COSMIC, Guideline on Non-Functional Requirements Practitioner / Expert (*)

[6] COSMIC, Early Sizing Practitioner / Expert Guides (*)

[7] Soubra, H., Abran, A., Stern, S., Ramdan-Cherif, A., 'Design of a Functional Size Measurement Procedure for Real-Time Embedded Software Requirements Expressed using the Simulink Model', International Workshop on Software Measurement - IWSM-MENSURA, Nara, Japan, 2011.

[8] Soubra, H., PhD thesis Automation de la mesure fonctionnelle COSMIC-ISO 19761 des logiciels temps-réel embarqué, en se basant sur leurs specifications

fonctionnelles. Ecole de technologie supérieure (ETS), Université du Québec and Université de Versailles at St-Quentin, in collaboration with Renault SAS.

[9] Renault, COSMIC Rules for Embedded Software Requirements Expressed using Simulink, 2012 (*).

[10] Swierczek, J, Automatic COSMIC sizing of requirements held in UML, COSMIC Masterclass, IWSM 2014, Rotterdam (*)

[11] Private client data.

[12] Stern, S., Gencel, C., Embedded software memory size estimation using COSMIC: a case study, International Workshop on Software Measurement – IWSM-MENSURA , Stuttgart (Germany), November 2010 - IWSM 2010 (*)

[13] Gencel, C., Heldal, R., and Lind, K., 'On the Conversion between the Sizes of Software Products in the Life Cycle', International Workshop on Software Measurement – IWSM-MENSURA, Stuttgart, November 2010.

COSMIC Measurement Manual for ISO 19761

Part 3c:
MIS Examples

June 2022

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)

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

June 2022 minor editing:

Text of Example 3.4 Expert System replaced by a link to its revision, the 'Expert System Measurement Case Study'.

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.

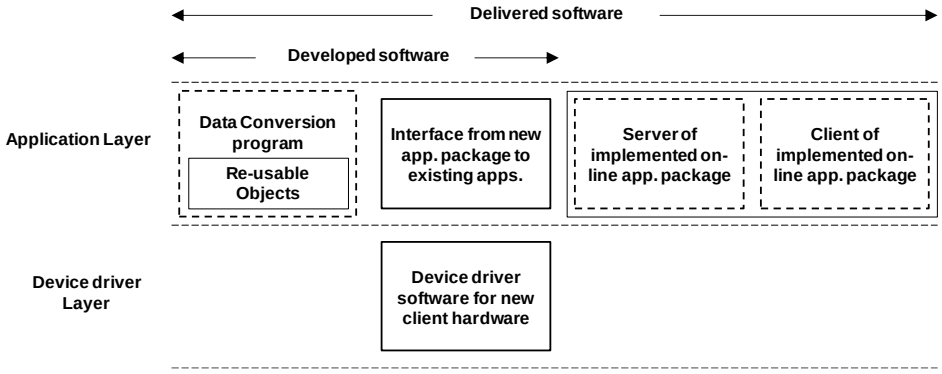


Figure 1.1 - The overall scope and the measurement scopes.

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:

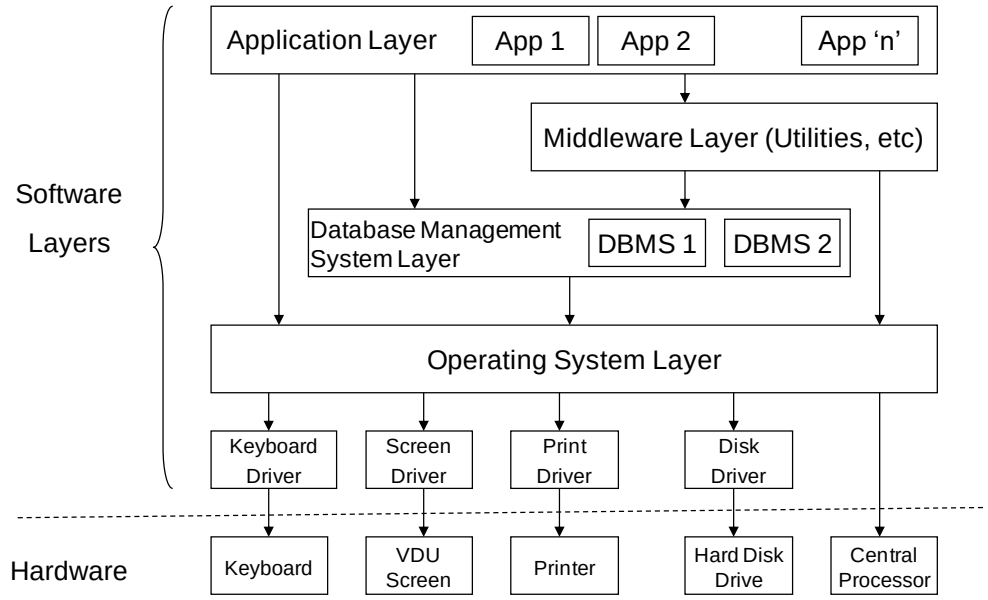


Figure 1.2 - Typical layered software architecture for a Business/MIS system.

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’.

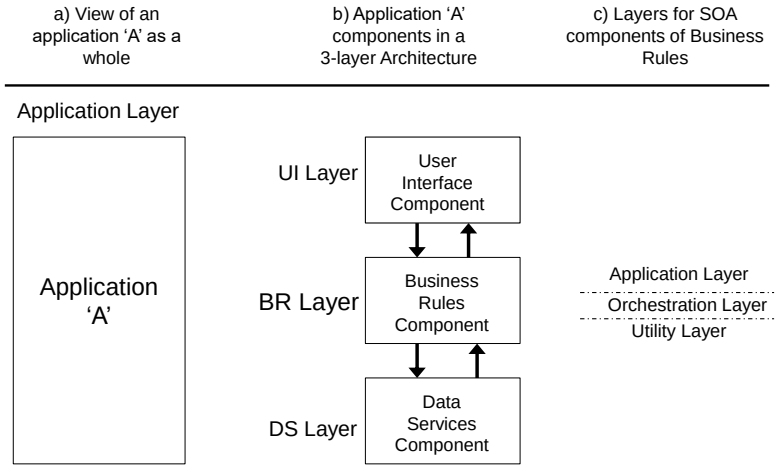


Figure 1.3 - Three views of the layers of an application.

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.

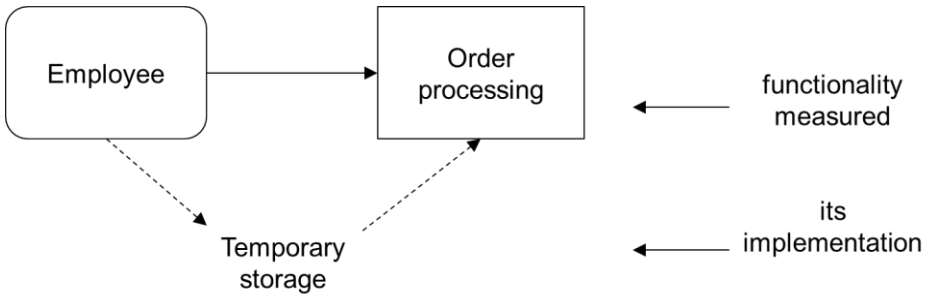


Figure 2.1 – Batch processing with temporarily storage.

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.

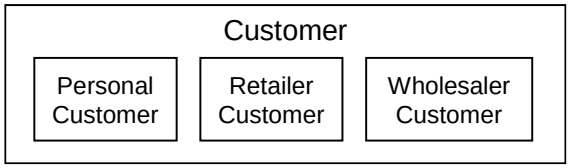


Figure 2.3 – Object of interest with its sub-types

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.

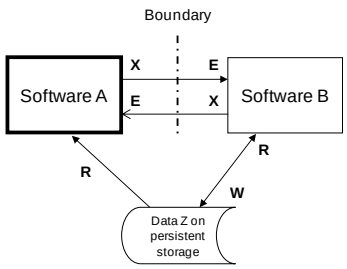


Figure 2.4 – Persistent data Z within the boundary of both software A and B for a Read.

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.

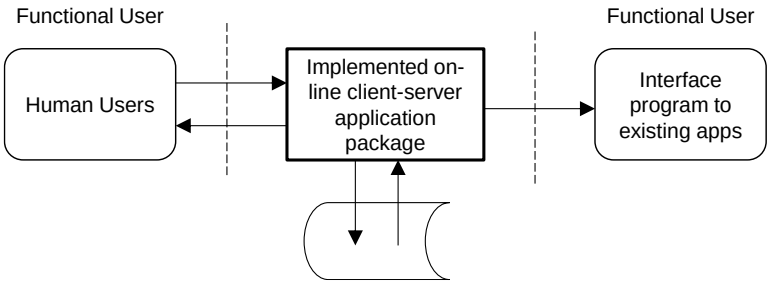


Figure 2.5 - Context diagram for the client-server application.

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):

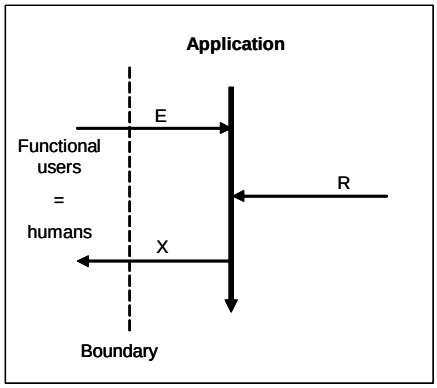


Figure 2.6 – Measurement scope is the whole application

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).

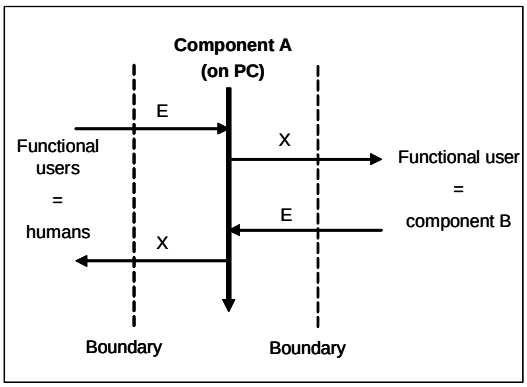


Figure 2.7 – Measurement scope is component A

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.

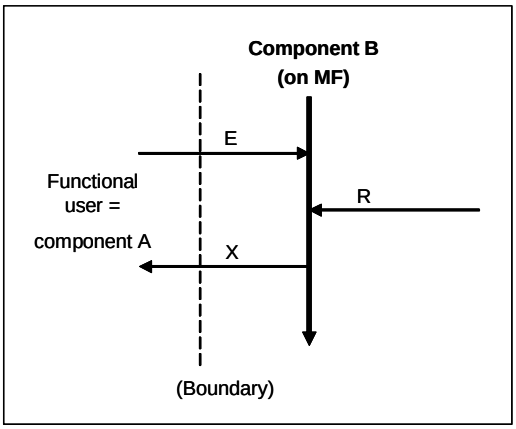


Figure 2.8 - Measurement scope is component B

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)

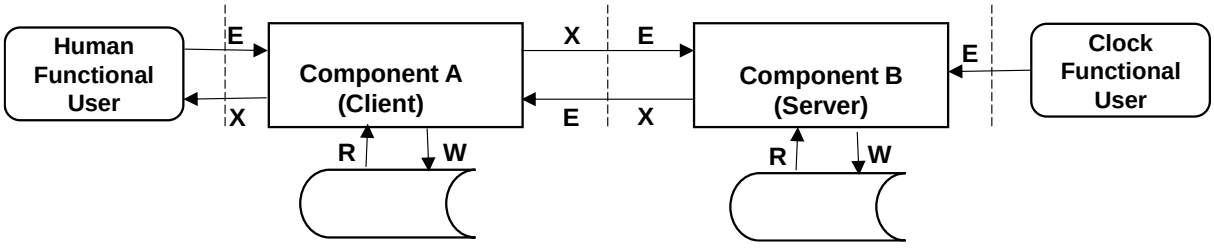


Figure 2.9 – Context diagram for cases b) and c).

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.

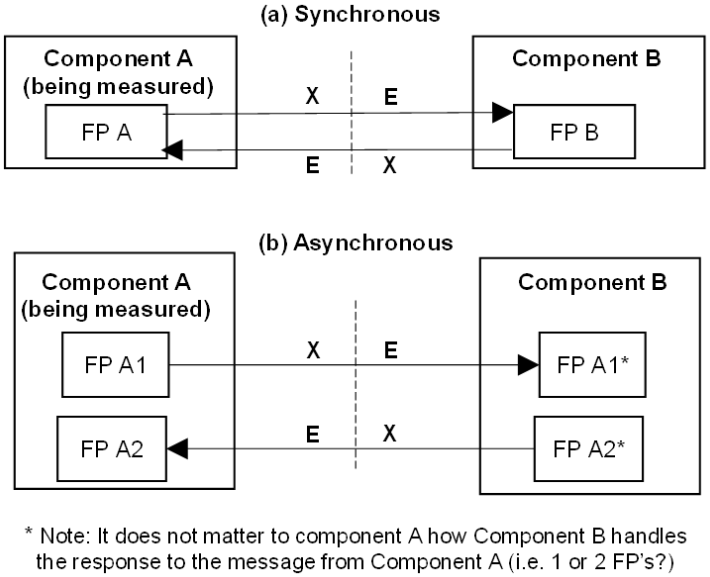


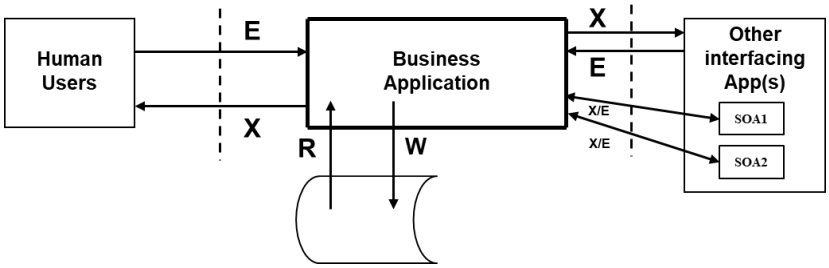
Figure 2.10 - COSMIC functional model of synchronous and asynchronous communications

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:

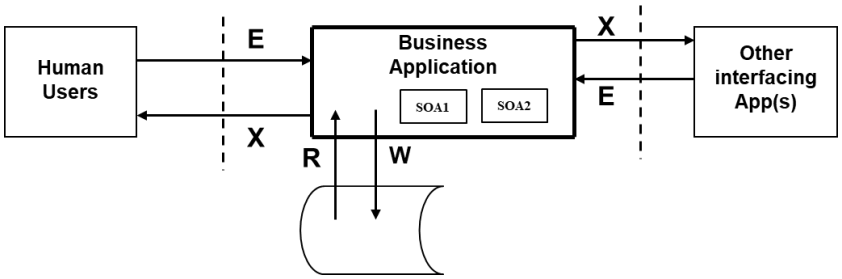
A. When the reusable components are *outside the scope* of the software being measured:

- 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 of a part of an application.

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

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

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

		employee total)

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

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

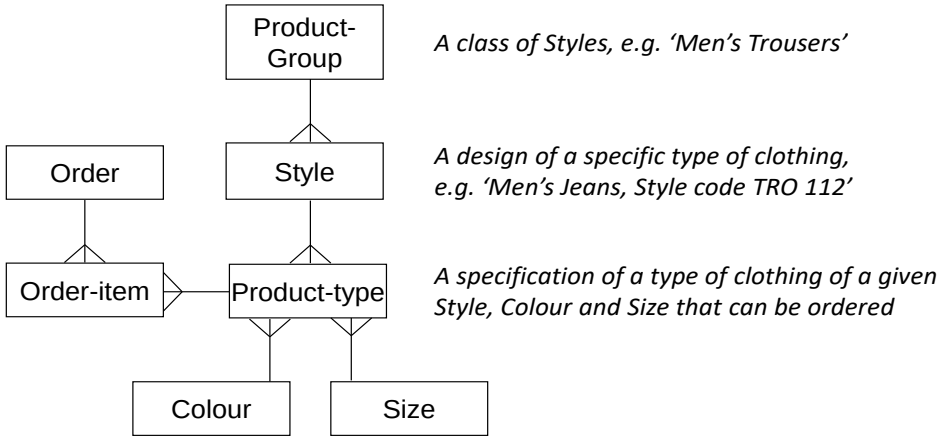


Figure 3.1 –The company's product structure and its product-types

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.

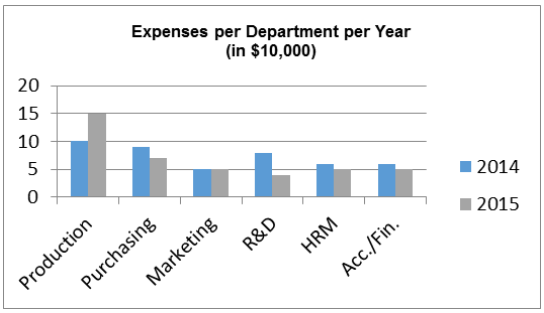


Figure 3.4 – Data in graphical form

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).

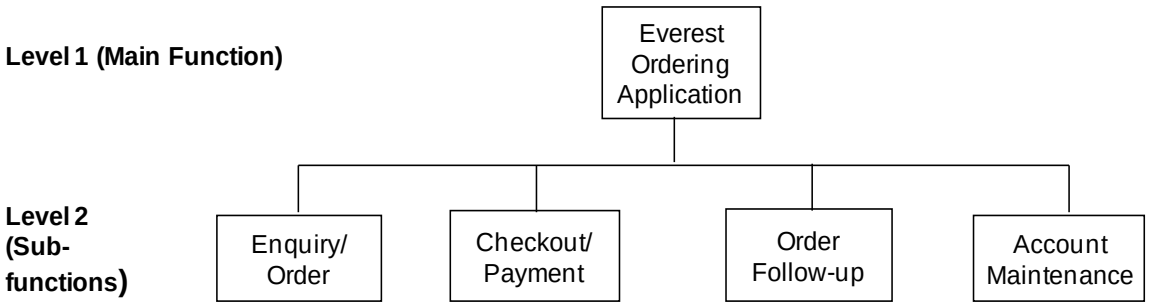


Figure 3.5 (a) - Analysis of the Everest Ordering System: the first two levels of granularity.

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.

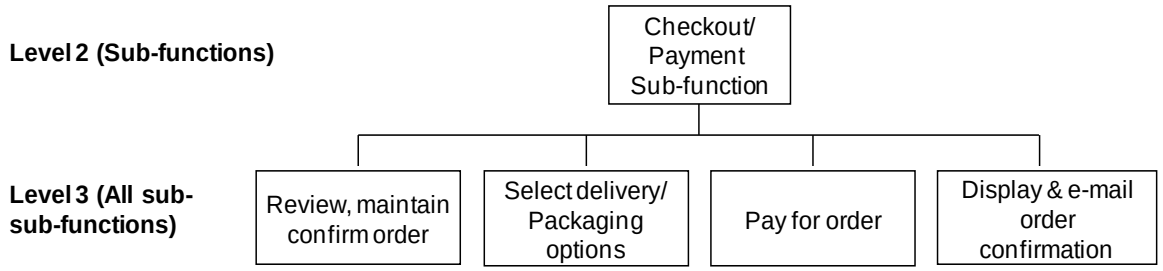


Figure 3.5 (b) - Analysis of the Checkout/Payment Sub-function.

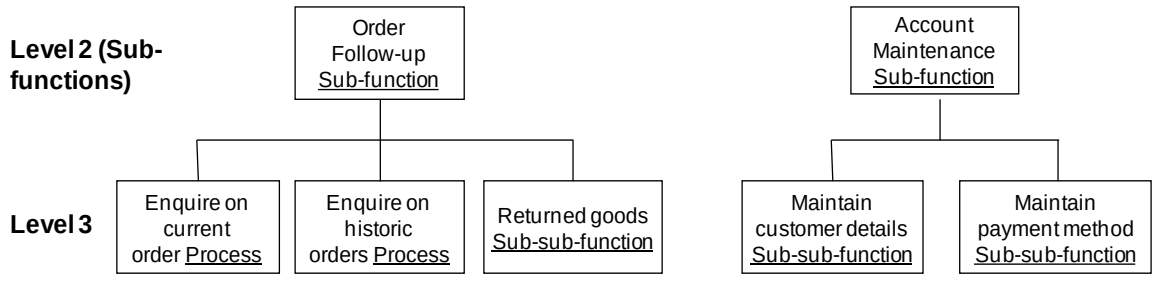


Figure 3.5 (c) - Analysis of the Order Follow-up and Account Maintenance Sub-function.

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.

COSMIC Measurement Manual for ISO 19761

Part 3d: Standardized Requirements with Big Data Cleaning Examples

Version 5.0

May 2022

Foreword.

The COSMIC Measurement Manual for ISO/IEC 19761:2011 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.

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

Purpose of this document.

The purpose of this document is to show COSMIC can be used to define structured requirements (called 'standardized FUR'), to specify Big Data Cleaning software examples with these requirements and that size follows, i.e. that a separate measurement of this standardized FUR is unneeded. See the Introduction to learn that standardized FUR is also useful when there is no need for size.

The COSMIC Group recommends that users study and master the COSMIC method, before applying the specification of standardized FUR.

May 2022 editing.

Section 5.4 Data movements for keys added. Section 5.7 Purpose mentions to merge 'relevant parts', however full tables are merged, specification of relevant columns added. Section 5.8 Requirement was added to display quartile values and enable verification of the results.

Editors.

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

Other members of COSMIC Measurement Practices Committee.

Jean-Marc Desharnais, Ecole de technologie supérieure – University of Quebec (Canada),
Peter Fagg, Pentad (UK),
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, IDC Metri (The Netherlands).

Table of Contents

1 INTRODUCTION.

The approach to a COSMIC measurement is to use functional user requirements (FUR) to determine functional size. To do so, the functional processes with their data movements must be identified. Specification of FUR and measurement are separate activities. However, if the FUR *themselves* are structured according to both COSMIC concepts, the size of such 'standardized FUR' is a consequence of that structure, so a separate measurement is unneeded.

Standardized FUR has several other interesting benefits:

- Applying the structure of the COSMIC concepts results in uniformly structured, 'standardized' specifications of FUR. If size isn't needed it can be ignored, without losing the other benefits.
- Standardized FUR can make user stories concrete. Assigning functional processes to user stories enables an early COSMIC estimate. This estimate can be refined when the data movements have been identified.
- Standardized FUR provides clearly structured specifications, facilitating verification of their completeness, correctness and consistency.
- As standardized FUR and its measurement coincide a measurement strategy is unneeded. It is replaced by a Requirements Agreement, a 'contract' between stakeholders and specifier. The stakeholders know what they get and whether the sizes are comparable to existing sizes. The specifier knows exactly what and how to specify.
- Standardized FUR's explicit structure of functional processes and data movements facilitates adaptation to new requirements and thereby supports 'living documentation'. It also provides an obvious input format for tools, including tools that generate testcases, test scripts and source code.

2 THE REQUIREMENTS AGREEMENT.

This section describes the key parameters that must be considered before the actual specification of requirements for a piece of software.

Capture

- the purpose of the specification and measurement exercises;
- the functional users of the software to be realized or changed, and the types of data movements that the software may handle;
- if applicable the layered architecture and components separately to be specified;
- the artefacts which will be needed for the specifications exercise.

Note. It is helpful to draw a context diagram of the software being specified.

3 APPLYING THE COSMIC CONCEPTS STRUCTURE.

3.1 Specification of a functional process

Specify

- triggering event (omit if the triggering event obviously follows from the name of the functional process)
- name of the functional process
- functional user(s)

3.2 Specification of a data movement

Note. Requirements that in a measurement are accounted for by the data movement functionality are indicated by '(*)'.

3.2.1 Specification of Entry functionality

Specify

- the data group of the triggering Entry and possibly other input data groups
- the functional user(s)
- (*) the required formatting and presentation manipulations along with all associated validations of the entered data attributes, to the extent that these data manipulations do not involve another type of data movement

If one of more Reads are required as part of the validation process, these are specified as separate Read data movements

3.2.2 Specification of Exit functionality

Specify

- the data group(s) moved to the functional user(s)
- (*) the required data formatting and presentation manipulations, including processing required to send the data attributes to the functional user, to the extent that these manipulations do not involve another type of data movement.

3.2.3 Specification of Read functionality

Specify

- the data group(s) to be retrieved from persistent storage
- (*) the logical processing and/or mathematical computation needed to read the data, to the extent that these manipulations do not involve another type of data movement

3.2.4 Specification of Write functionality

Specify

- the data group(s) to be stored to persistent storage

- the data group(s) to be deleted or validity to be ended
- (*) the logical processing and/or mathematical computation to create the data attributes to be written, to the extent that these manipulations do not involve another type of data movement.

3.3 Specification of functional changes

For details about measurement of changes, see Part 2 of the Measurement Manual, section ‘Measurement of the size of changes to software’.

4 EXAMPLE SPECIFICATION

(Abbreviations: TEv = triggering event, E = Entry, X = Exit, R = Read, W = Write, FU = functional user).

Note. Pay particular attention to updating the standardized FUR after realization or change of the software.

4.1 Example MIS specification

(from the C-Reg Case study)

FP: Add a Professor.			Size: 4 CFP
Specification	*Data group*		
Registrar (FU) enters details for the Professor	Professor details	E	1
C-Reg checks if the data describe a Professor who already exists and if so, displays an error message, else validates the entered data	Professor details	R	1
C-Reg creates a new Professor	Professor details	W	1
Display error message	Error Message	X	1

4.2 Example Real-time specification

(from the Rice cooker Case study)

FP: Start cooking (start button pressed)			Size: 3 CFP
Specification	*Data group*		
When the start button (FU) is pressed it sends a start signal to the software	Start signal	E	1
The software sends a Turn ON command to the Heater (FU)	Turn ON command to Heater	X	1
The software sends a Turn ON command to the Cooking Lamp (FU)	Turn ON command to Cooking Lamp	X	1

4.3 Example change of Real-time specification

(see the Rice cooker Case study)

To the first prototype a hardware ‘convenience timer’ has been added: a start signal can also be provided by this new timer. Cooking can start at a convenient point of time set by the human use (function implemented by the hardware). To prevent malfunction the software must ensure that any new start signal is ignored when the cooker is processing.

FP: Start cooking (TEv: button or convenience timer start) Size of change: 2 CFP			
Specification	*Data group*		
When the start button (FU) is pressed it sends a start signal to the software.	Start signal	E	-
At the set time the convenience timer (FU) sends a start signal to the software (data movement added).	Start signal	E	1
Ignore any start signal if the cooker has been started (data movement added).	Cooking mode	R	1
The software sends a Turn ON command to the Heater (FU)	Turn ON command to Heater	X	-
The software sends a Turn ON command to the Cooking Lamp (FU)	Turn ON command to Cooking Lamp	X	-

5 BIG DATA CLEANING EXAMPLE SPECIFICATION

5.1 The Requirements Agreement.

The Requirements Agreement for the Big Data cleaning software could be:

Purpose of the specifications exercise. Purpose of the specifications is twofold. The specifications describe the functionality of the software to be realized. In addition, the integrated functional size is input for estimating effort of its realization.

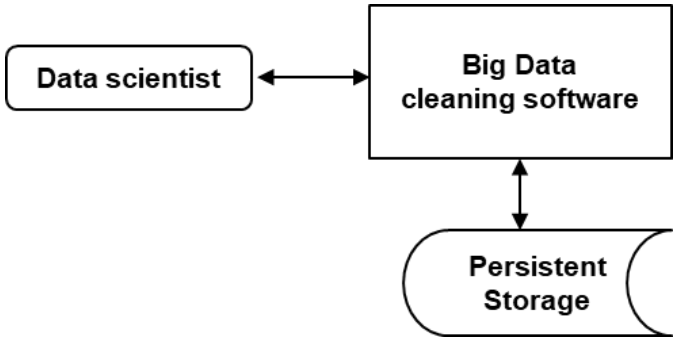
Functional users of the software. In the examples the data scientist is the only functional user.

Types of data movements that the software may handle. The types of data groups pertain to the data, their location (i.e. data set IDs and/or column IDs), defects, constraints and phenomena that the data scientist needs to cope with.

Layered architecture and components. Not applicable.

Artefacts needed for the specifications exercise. Not applicable. Usually the specifications exercise will be based on interview reports of data managers.

The context diagram of the software being specified.



5.2 Record cleaning operation details.

Purpose: Keep record of every cleaning operation (in plain text).

FP: Create cleaning operation.			**Size: 3 CFP**
Specification	*Data group*		
Data scientist (FU) enters details of a cleaning operation	Cleaning operation details	E	1
Store cleaning operation details	Cleaning operation details	W	1
Display error message	Error Message	X	1

FP: Display list of cleaning operations.			**Size: 4 CFP**
Specification	*Data group*		
Data scientist (FU) enters cleaning operation search term	Cleaning operation search term	E	1
Read cleaning operation details	Cleaning operation details	R	1
Display cleaning operation details	Cleaning operation details	X	1
Display error message	Error Message	X	1

FP: Display cleaning operation details.			**Size: 4 CFP**
Specification	*Data group*		
Data scientist (FU) enters cleaning operation ID	Cleaning operation details to be changed	E	1
Read cleaning operation details	Cleaning operation details	R	1
Display cleaning operation details	Cleaning operation details	X	1
Display error message	Error Message	X	1

FP: Update cleaning operation.			**Size: 3 CFP**
Specification	*Data group*		
Data scientist (FU) enters changed details of the cleaning operation	Cleaning operation changed details	E	1
Store changed cleaning operation details	Cleaning operation changed details	W	1
Display error message	Error Message	X	1

5.3 Test column data quality.

Note: in the following ‘constraint’ includes ‘text pattern’

FP: Check Column data quality			**Size: 6 CFP**
Specification	*Data group*		
Data scientist (FU) enters data set ID	Data set ID	E	1
Data scientist (FU) enters column ID	Column ID	E	1
Data scientist (FU) enters constraint to compare with	Constraint	E	1
Read value	Value	R	1
Check constraints and display violation	Constraint violation	X	1
Display error message	Error Message	X	1

5.4 Test relationship foreign key

FP: Test relationship foreign key			**Size: 8 CFP**
Specification	*Data group*		
Data scientist (FU) enters foreign data set ID	Foreign data set ID	E	1
Data scientist (FU) enters foreign key ID	Foreign key ID	E	1
Data scientist (FU) enters primary data set ID	Primary data set ID	E	1
Data scientist (FU) enters primary key to compare with	Key ID	E	1
Read foreign key value	Foreign key value	R	1
Read primary key value	Primary key value	R	1
Display key violation	Both data set IDs and foreign key	X	1
Display error message	Error Message	X	1

5.5 Test business rule.

Note: in the following prime customers (marked somehow) get a given percentage discount on an order.

FP: Test business rule			**Size: 6 CFP**
Specification	*Data group*		
Data scientist (FU) enters data set ID of prime customers	Data set ID of prime customers	E	1
Data scientist (FU) enters selection of prime customer	Customer type to be tested	E	1
Read customer data	Customer data	R	1
Read associated bill data	Customer bill data	R	1
Display business rule violation	Customer ID, bill ID, actual percentage	X	1
Display error message	Error Message	X	1

5.6 Test data violations in data set.

Purpose: to test a data set before processing it.

FP: Test data violations in data set			**Size: 6 CFP**
Specification	*Data group*		
Data scientist (FU) enters data set ID	Data set ID	E	1
Data scientist (FU) enters column ID	Column ID	E	1
Data scientist (FU) enters constraint to be tested	Constraint	E	1
Read value and test constraint	Value to be tested	R	1
Display violation	Value, violated constraint ID	X	1
Display error message	Error Message	X	1

5.7 Merging datasets.

Purpose: combining relevant parts of datasets to create a new file.

FP: Merging datasets			**Size: 10 CFP**
Specification	*Data group*		
Data scientist (FU) enters first data set ID to be merged	First data set ID	E	1
Data scientist (FU) enters linking column ID of first data set to be merged	Linking column ID of first data set	E	1
Data scientist (FU) enters column ID of first data set to be merged	Column ID	E	1
Data scientist (FU) enters second data set ID to be merged	Second data set ID	E	1

Data scientist (FU) enters linking column ID of second data set to be merged	Linking column ID of second data set	E	1
Data scientist (FU) enters column ID of second data set to be merged	Column ID	E	1
Read first data set	Record of first data set	R	1
Read second data set	Matching record of second data set	R	1
Merge records to new data set	Merged record	W	1
Display error message	Error Message	X	1

5.8 Detecting Outliers.

FP: Detect Outliers.			**Size: 6 CFP**
Specification	*Data group*		
Data scientist (FU) enters data set ID	Data set ID	E	1
Data scientist (FU) enters ID of column to be tested	Column ID	E	1
Read value	Value	R	1
Compute and display the quartiles Q1 and Q3 and the interquartile range	Quartile info	X	1
Read value	Movement of data accounted for by previous Read	-	-
Compare value with interquartile range and display it if outside interquartile range	Data group including outlier	X	1
Display error message	Error Message	X	1

5.9 Test for duplicates.

FP: Test for duplicates.			**Size: 5 CFP**
Specification	*Data group*		
Data scientist (FU) enters data set ID	Data set ID	E	1
Data scientist (FU) enters column ID(s) of combination of column(s) that should be different	Column ID	E	1
Read data set record	Data set record	R	1
Display duplicate if combined values equal	Data group with possible duplicate	X	1
Display error message	Error Message	X	1

5.10 Data transformation.

Transform data and store the result into an added column

FP: Data transformation.			**Size: 6 CFP**
Specification	*Data group*		
Data scientist (FU) enters data set ID	Data set ID	E	1
Data scientist (FU) enters the column ID of the data to be transformed	Column ID	E	1
Data scientist (FU) enters transformation rule	Transformation rule	E	1
Read value	Value	R	1
Transform value and store it	Transformed value	W	1
Display error message	Error Message	X	1

5.11 Test for missing value.

FP: Test for missing value.			**Size: 5 CFP**
Specification	*Data group*		
Data scientist (FU) enters data set ID to test for missing value	Data set ID	E	1
Data scientist (FU) enters the column ID to test for missing value	Column ID	E	1
Read data record	Value	R	1
Notify if value is missing	Data record ID	X	1
Display error message	Error Message	X	1