
# 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'.

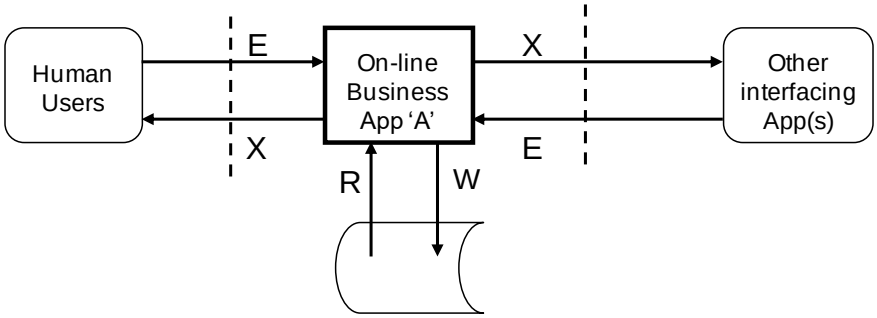



**Figure 2.1 – Whole online business application.**

### *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.

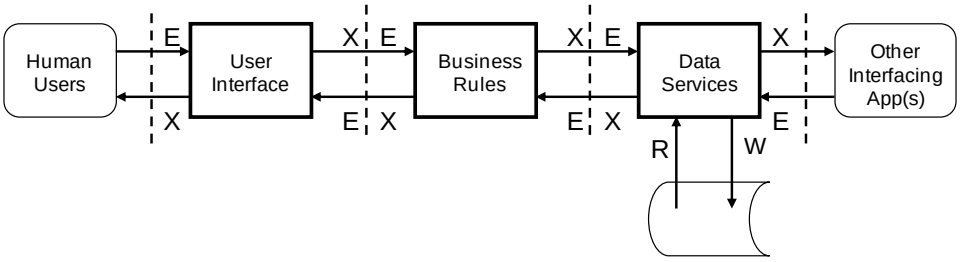



**Figure 2.2 – Major components of an online business application.**

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.

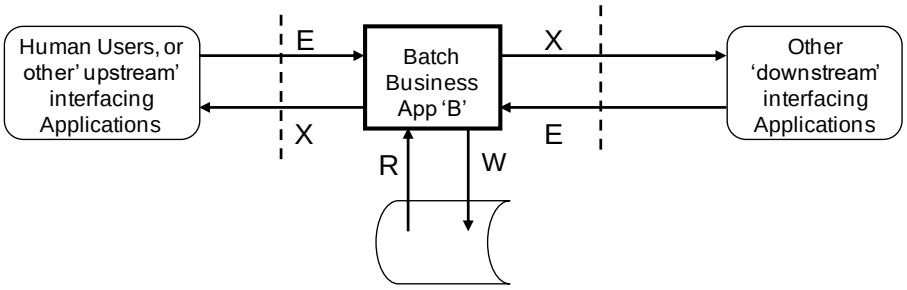



**Figure 2.3 - Business application processed in batch mode.**

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.

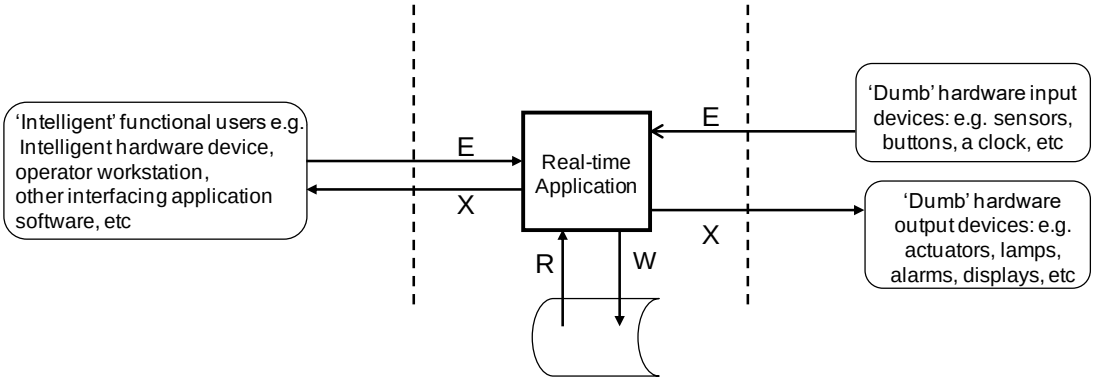



**Figure 2.4 - A real-time application to be measured as a whole.**

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.

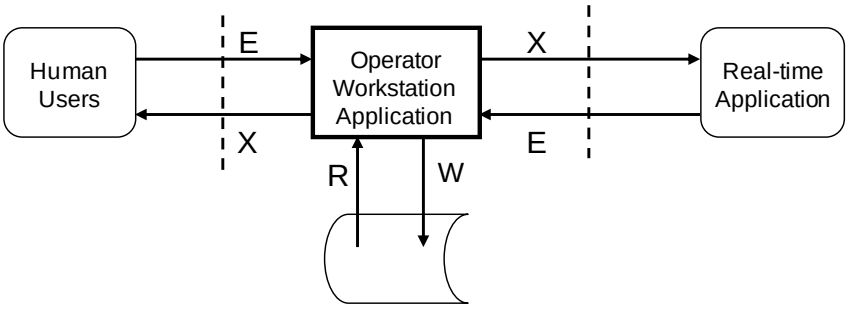



**Figure 2.5 - The human operator workstation of a real-time software system.**

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.

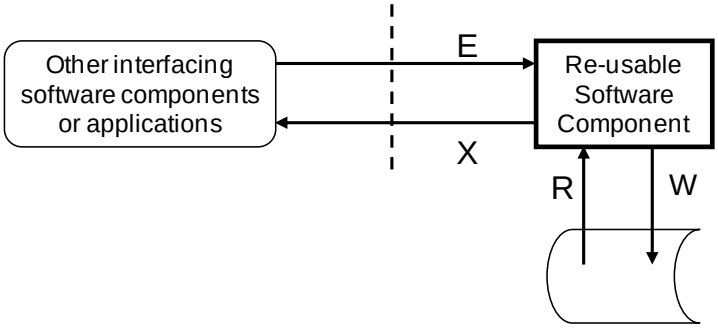


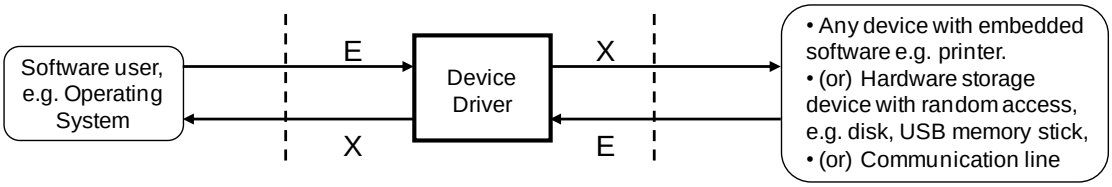

**Figure 2.6 - Reusable software component.**

### *2.3.2 Standard strategy for device driver software.*

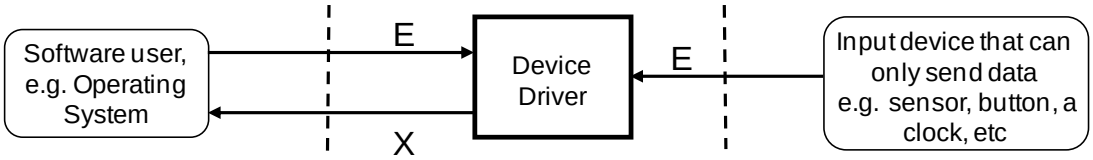
Three categories of device-driver software can be defined. Their respective context diagrams are shown below.

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

a) 'Intelligent' Input / Output Device




b) 'Dumb' Input Device




c) A 'Dumb' Output Device

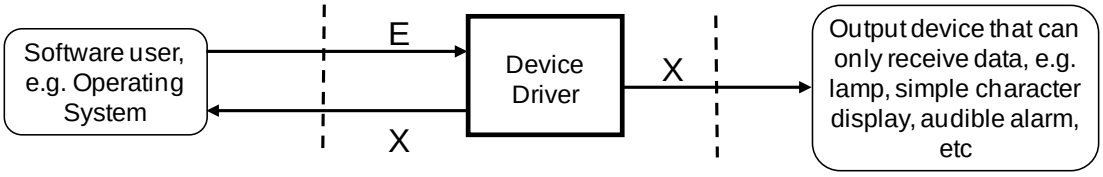



**Figure 2.7 - Device driver software.**

## 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.

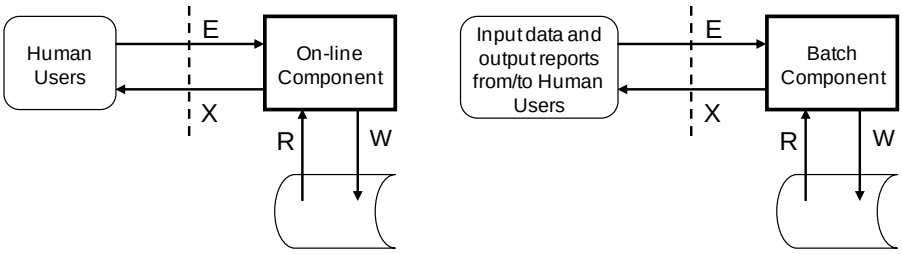



**Figure 3.1 - Example of a mixed on-line/batch business application system.**

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.