



Guideline for ‘Measurement Strategy Patterns’

**Ensuring that COSMIC size measurements
may be compared**

VERSION 1.0

March 2013

Version control

Date	Reviewer(s)	Modifications / Additions
March 2013	Measurement Practices Committee	First version for public distribution

Acknowledgements

Version 1.0 authors and reviewers 2013 (alphabetical order)		
Alain Abran, École de Technologie Supérieure, Université du Québec, Canada	Diana Baklizky, ti Metricas Ltda, Brazil	Juan J. Cuadrado-Gallego, University of Alcalá, Madrid, Spain
Jean-Marc Desharnais*, École de Technologie Supérieure, Université du Québec, Canada	Cigdem Gencel, Free University of Bozen/Bolzano, Italy	Peter Fagg, Pentad Ltd., United Kingdom
Arlan Lesterhuis, Netherlands	Bernard Londeix, Telmaco Ltd, United Kingdom	Pam Morris, Total Metrics, Australia
Luca Santillo, Agile Metrics, Italy	Hassan Soubra, ESTACA, France	Charles Symons*, United Kingdom
Sylvie Trudel, Université du Québec à Montréal, Canada	Gerhard Ungerer, Random Bit Corp., USA	Frank Vogelesang, Ordina, Netherlands
Chris Woodward, CW Associates Ltd., United Kingdom		

* Author of this guideline

Foreword

The purpose of this Guideline is to describe, for each of several different types of software, a standard set of parameters for measuring software sizes, called a 'measurement strategy pattern', which we abbreviate to 'measurement pattern' for convenience. Measuring sizes using a standard measurement pattern should help ensure consistency across all the purposes for which the sizes may be used, e.g. performance measurement, benchmarking and project estimating,

A measurement pattern is actually a template for a standard set of parameters (type of functional users and level of decomposition for a given functional domain) that must be determined during the Measurement Strategy phase of the COSMIC measurement process. Consistent use of measurement patterns will therefore not only help ensure comparability of measurements but will support and save time in the Strategy phase of a measurement.

Chapter 1 describes in more detail why it is so important to ensure functional size measurements are comparable. It then defines a measurement pattern and describes how they are documented.

The next three chapters describe the measurement patterns first by functional domain and then by other characteristics within these domains.

- Business Applications (Chapter 2)
- Real-time Applications (Chapter 3)
- Infrastructure Software (Chapter 4)

Chapter 5 shows a summary of all the patterns and how they relate.

Chapter 6 gives examples of how measurement patterns may be used to decide which sizes to measure, and deals with reporting measurement results.

The reader is assumed to be fully familiar with the COSMIC FSM method.

All information about COSMIC and documentation on the COSMIC method is available for free download from www.cosmicon.com.

The COSMIC Measurement Practices Committee.

Table of Contents

1	THE NEED TO ENSURE COMPARABILITY OF SIZE MEASUREMENTS.....	6
1.1	Measurement patterns.....	7
1.2	The terminology used for the measurement patterns.....	8
	1.2.1 <i>Functional domain</i>	8
	1.2.2 <i>Level of decomposition</i>	8
	1.2.3 <i>Suitability of the pattern for benchmarking</i>	9
1.3	Ensuring the comparability of project performance data	9
1.4	The symbols used in the measurement patterns	11
2	MEASUREMENT PATTERNS FOR BUSINESS APPLICATION SOFTWARE	12
2.1	An on-line business application	12
2.2	A business application processed in batch mode	13
2.3	An on-line business application built with a multi-layer (or multi-tier) architecture	14
3	MEASUREMENT PATTERNS FOR REAL-TIME APPLICATION SOFTWARE.....	16
3.1	A real-time application	16
3.2	Simple real-time embedded software	18
3.3	The operator workstation application of a real-time software system	19
4	MEASUREMENT PATTERNS FOR INFRASTRUCTURE SOFTWARE	20
4.1	Re-usable software components	20
4.2	Hardware device driver software	21
5	SUMMARY OF MEASUREMENT PATTERNS	22
6	USING MEASUREMENT PATTERNS IN PRACTICE.....	23
6.1	Example of measuring a project that delivers a mixed on-line/batch business application system	23
6.2	Example of an application that may be viewed as single- or multi-layered	24
6.3	A business application that uses re-usable components	25
6.4	Reporting measurement results obtained using measurement patterns.....	25
	Appendix - COSMIC CHANGE REQUEST AND COMMENT PROCEDURE	26

THE NEED TO ENSURE COMPARABILITY OF SIZE MEASUREMENTS

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, i.e. the COSMIC Function Point (CFP). These different sizes 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).

(Size will also probably vary depending on the timing of the measurement as the requirements evolve during the software development life-cycle, but we are not concerned with that aspect here.).

One of the most common purposes of measuring functional sizes using the COSMIC method is to measure the performance of the project that developed the software, e.g. using:

$$\text{Project productivity} = \text{output} / \text{input} = \text{software size} / \text{effort}.$$

Such measurements may then be used to compare productivity across different projects and to establish productivity benchmarks. When developing some new software, an estimate of the size of the new software and benchmark productivity data can be used as input to methods for estimating the effort to develop the new software.

Clearly if all these processes of performance measurement, estimating and benchmarking are to work satisfactorily, then the software sizes (and many other project parameters) must be measured in a way that ensures they are comparable across all the projects involved. The relationships between these activities are shown in Figure 1.1 below.

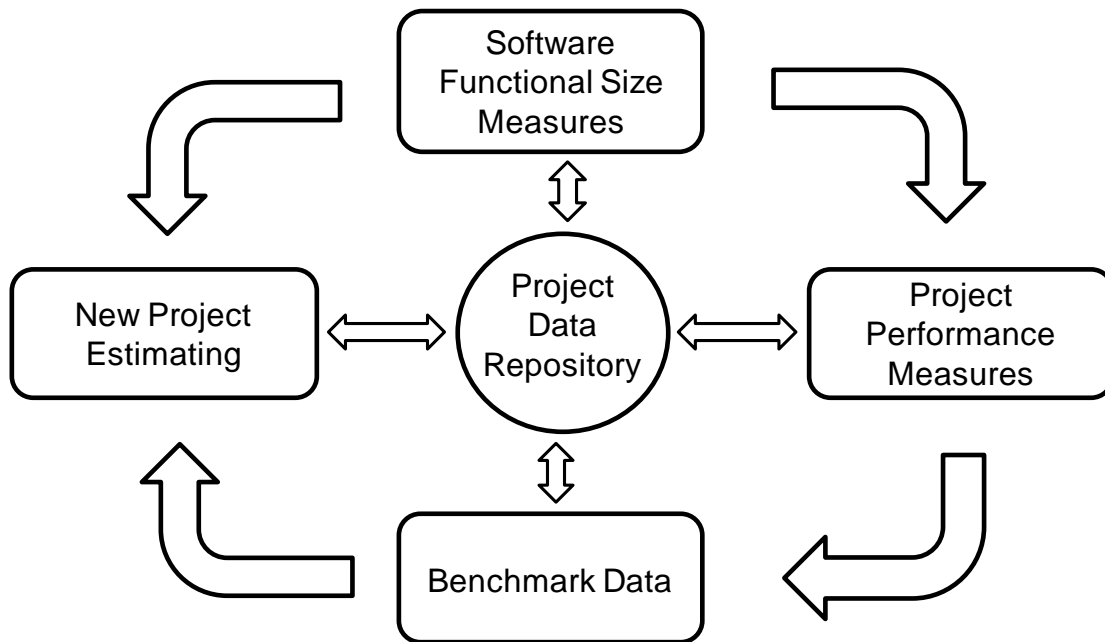


Figure 1.1 Inter-relationships that require consistent measurements

As noted above, a piece of software can have different sizes depending on the *purpose* of the measurement, even using the same Functional Size Measurement (FSM) method. For example, suppose an application that is distributed over several hardware platforms. It can be measured in two ways depending on the interests of different stakeholders.

- A manager may be interested in the productivity of the development project and need to measure the size of the delivered software 'as a whole' (as required by the customer, and as seen by the human user for the system).
- The developer of the new application, who must build the components, perhaps using different technologies, may wish to measure the sizes of the components separately and use benchmark data relevant to the separate technologies for estimating purposes.

The aggregation rules for the COSMIC method lead to the result that the size of the whole is less than the sum of the sizes of its components. In the example, functionality that the component on each platform 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' functionality seen by the human user.

As shown by this example, therefore, measurers must clearly identify and distinguish measurements of 'whole' pieces of software from measurements of software components.

In practice, several factors must be considered in the Measurement Strategy phase of the COSMIC process to help decide which size to measure. These factors must be recorded with a measurement so that its meaning is clearly understood and it will be correctly used in the future.

1.1 Measurement patterns

The purpose of this Guideline is to document a set of commonly-occurring situations faced by COSMIC measurers, i.e. effectively a set of re-usable, standard software size measurement 'patterns', together with their characteristics.

DEFINITION – Measurement (Strategy) Pattern

A 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.

In effect, a measurement pattern defines a standard combination of parameters that must be determined in the Measurement Strategy phase of a COSMIC measurement process. As it also defines the possible types of data movements, a measurement pattern provides a template for drawing the 'context diagram' for the software to be measured. Regular use of a limited set of measurement patterns should therefore speed up the measurement process, especially in the Measurement Strategy phase, as well as ensure 'like-for-like' size comparisons.

For each pattern we discuss 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. (How benchmark figures are developed and how sizes and benchmark data are used in estimating is not dealt with in this Guideline.)

The set of measurement patterns given in this Guideline is not intended to be (and never will be) complete. Users of the COSMIC method should develop their own local measurement patterns if the standard patterns described here do not meet their size comparability needs.

1.2 The terminology used for the measurement patterns

Terms used in this Guideline are either standard COSMIC terms, or are taken from the 'COSMIC/ISBSG Concise Data Collection Questionnaire' (v2011-06-01), obtainable from www.cosmicon.com/portal. (ISBSG is the International Software Benchmarking Standards Group.)

1.2.1 Functional domain

COSMIC distinguishes four functional domains that are also used in ISBSG data collection, namely:

- Business Application.
- Real-time Application
- Infrastructure Software
- Mathematically-intensive Application.

Only the first three domains are relevant to this Guideline, since the measurement patterns of these three types are distinguished in part by their different types of functional users. The measurement patterns are independent of whether the software to be measured is mathematically-intensive or not.

1.2.2 Level of decomposition

As illustrated by the examples in the discussion of the need for comparability, size measurements can only be comparable if made at the same level of decomposition. COSMIC defines a 'level of decomposition' as:

'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.'

There is also a Note 2 to the definition that states: 'Size measurements of the components of a piece of software may only be directly comparable for components at the same level of decomposition'.¹

For the purposes of this Guideline, we will define measurement patterns for a 'whole' piece of software and at two levels of decomposition:

- 'Whole' (or 'Monolithic'). The pattern for the software to be measured, typically a business or real-time application, reveals its functionality 'as a whole', i.e. without revealing any details of its physical breakdown into components.
- 'Major Component'. The pattern for the software to be measured reveals the functionality of the components of the software at the first 'Level 1' of decomposition. This pattern is typically used to reveal the major components of a business application that is built with a 'multi-layer' or 'multi-tier' architecture, where the major components may or may not be distributed over different hardware platforms
- 'Minor Component'. The pattern reveals the functionality of a piece of software to be measured that is a component, re-usable or not, at any level of decomposition below the Major Component level. This pattern is applicable to any minor component, regardless of the functional domain (business, real-time or infrastructure) in which it is used.

1.2.3 Suitability of the pattern for benchmarking

'External Benchmarking': Comments under this heading concern whether sizes measured using this pattern should be comparable across multiple organizations and whether resulting project performance data are suitable for submission to benchmarking organizations such as the ISBSG.

'Internal Benchmarking': Comments under this heading discuss whether any special care must be taken to ensure comparability of size measurements made using this pattern, even within one organization.

The comment of 'Suitable' for either external or internal benchmarking relates only to a COSMIC software size measured using the associated pattern.

1.3 Ensuring the comparability of project performance data

Ensuring the comparability of performance data such as project productivity does not, of course, depend only on ensuring comparability of the size measures. Many other parameters will need to be considered to ensure 'like-for-like' comparisons of project performance. Examples include:

- the same project 'type', i.e. for a new development, an enhancement of existing software, or a re-development,
- ensuring the project effort is measured in the same way,
- comparable project characteristics, such as the development life-cycle style (e.g. waterfall versus agile), staffing constraints, etc,
- the same technology (hardware platform, programming language, etc) is used,
- the same processing mode (on-line versus batch processing of business applications),
- comparable non-functional requirements such as system performance, availability, response time targets, etc,

¹ This Note 2 to this definition has been updated in light of Method Update Bulletin 9 on 'Layers' (published November 2012), which proposes to delete the term 'peer component' from the COSMIC vocabulary as unnecessary.

- comparable risk factors, e.g. requirements stability, use of new technology, etc.

The first of a series of five ISO standards on software project benchmarking has now been published². 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.

Another obvious point to consider is that a single project may deliver several separate pieces of software. For example, a project could deliver three products: a new application, an enhancement to an existing application that interfaces with the new application, and some software that will be used once to convert data to a new format. The concept of 'project productivity' is therefore very often not just a matter of simply dividing the size of one piece of delivered software by the effort to develop it.

Software projects can deliver such a variety of different types of products that it is neither possible nor desirable to define standard measurement patterns for all combinations of project products. When a project delivers a mix of different types of products and there is an aim to produce project performance measures that will be comparable with other measures, the challenge will be to determine which products can be sensibly measured using a standard or locally-defined pattern. It must also be possible to record or to allocate the associated project effort to each product and to record the other relevant parameters (as above).



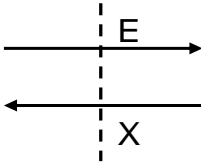
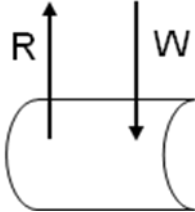
How to do this in general for software projects that have multiple deliverables is beyond the scope of this Guideline.

As always, the key to successful measurement is to follow the COSMIC Measurement Strategy process by first determining the purpose of a measurement and how it will be used. This will determine the scope of the piece(s) of software to be measured, and the measurement pattern(s) to be used.

² ISO/IEC 29155-1. Systems and software engineering - Information technology project performance benchmarking framework - Part 1: Concepts and definitions.

1.4 The symbols used in the measurement patterns

Key to symbols used in the patterns

Symbol	Interpretation
	The piece of software to be measured (box with thick outline)
	Any functional user of the software being measured. (Functional users are referred to as 'users' in the diagrams, for simplicity).
	The arrows represent <u>all</u> the Entry (E) and Exit (X) data movements crossing a boundary (the dotted line) between a functional user and the functional processes of the software being measured. The Entry data movements will include triggering Entries for the functional processes.
	<p>The arrows represent <u>all</u> the Read (R) and Write (W) data movements between the functional processes of the software being measured and persistent storage.</p> <p>(The standard flowchart symbol for 'data storage' is used to emphasize that persistent storage is an abstract concept. The software being measured is required to make some data persistent or to retrieve some data from persistent storage. The data storage symbol indicates that the software does not interact directly with physical hardware storage.)</p>

MEASUREMENT PATTERNS FOR BUSINESS APPLICATION SOFTWARE

The three most common patterns for business application software are:

- On-line business applications considered as a whole (see section 2.1)
- Batch processed business application (2.2)
- On-line business applications considered as components of a multi-tiered implementation (2.3)

2.1 An on-line business application

The first pattern shown in Figure 2.1 should be used when the need is to determine the size of an on-line business application 'A', seen as a whole, with humans and other interfacing applications as its functional users.

We assume that the Functional User Requirements (FUR) of any on-line business application A to be measured define the functionality needed by the business, including the human interface requirements for the customer of the software. The FUR to be measured do not describe functions provided by the operating environment or by 'control commands' (see the COSMIC 'Measurement Manual', 3.0.1, section 4.1.10).

The size of this application is denoted as S_A . For sizes measured for other types of business application patterns in this chapter, we will discuss whether their measured sizes may be sensibly compared against S_A .

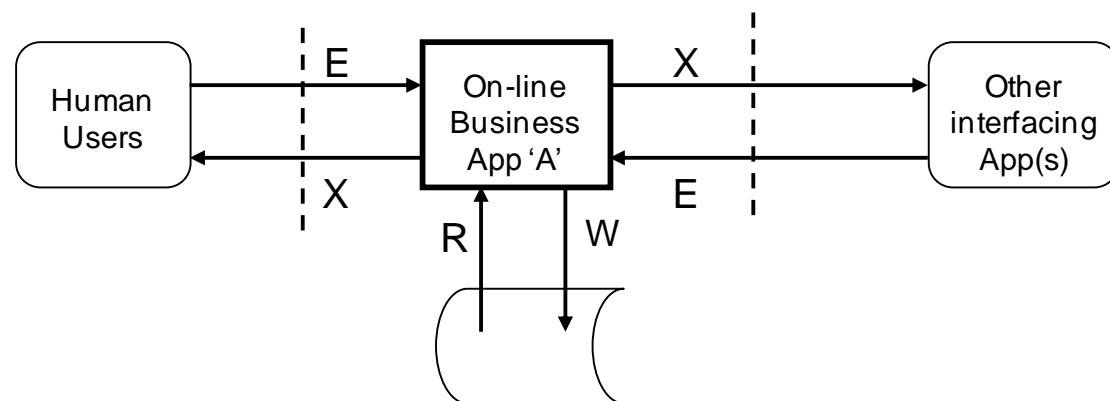


Figure 2.1 - Online business application seen as a whole

Level of decomposition: The Whole (i.e. no decomposition)

Suitability for external benchmarking: Suitable. This is the reference pattern for all other size measures in the domain of Business Application software.

Suitability for internal benchmarking: Suitable

2.2 A business application processed in batch mode

The pattern for an application B that processes data in batch mode is shown in Figure 2.2.

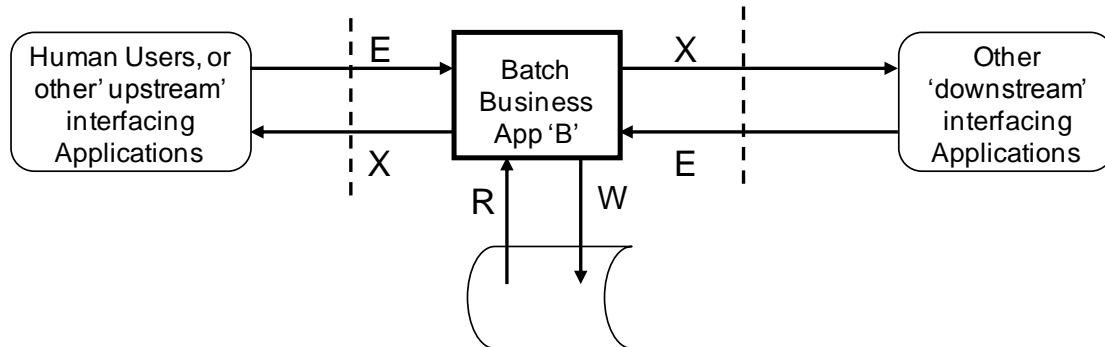


Figure 2.2 - Business application processed in batch mode

Any human users of a batch process do not, of course, interact directly with the software as in the on-line pattern shown in section 2.1. However, human users provide the input data for the functional processes of the application and receive the output data from the batch processing.

'Upstream' interfacing applications are functional users that send data to (or that make a batch of input data available for) the application B for batch processing. 'Downstream' applications are functional users 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 is no input data (e.g. to produce a series of reports), see the 'Guideline for sizing Business Application Software'.

Level of decomposition: The Whole (i.e. no decomposition).

Suitability for external benchmarking: Suitable. The size S_B of the batch-processed application B may be sensibly compared against the size S_A of the reference On-line Business Application software in section 2.1. 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 product comprising an on-line and a batch component, see Example 1 in Chapter 6.)

2.3 An on-line business application built with a multi-layer (or multi-tier) architecture

As an example, the pattern for an on-line business application that is developed using a multi-layer (or multi-tier) architecture (in this case 3 layers) is shown in Figure 2.3. The three major components are the 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.

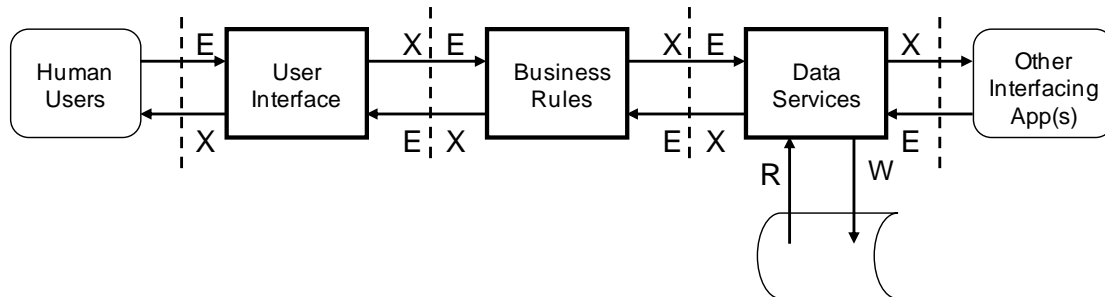


Figure 2.3 - Online business application built on multi-tiered architecture

Each major component can be measured separately in the normal way, giving sizes S_{UI} , S_{BR} and S_{DS} respectively.

Level of decomposition: '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), the size of the 'whole' application, S_C , may be sensibly compared against the size S_A of the reference On-line Business Application software in section 2.1. (The size S_C of the 'whole' application is the sum of the sizes of the three major components ($S_{UI} + S_{BR} + S_{DS}$) **less** the size contribution from all Entries and Exits crossing the two boundaries between the three major components.)³

For purpose b), the size of each of the three major components, S_{UI} , S_{BR} and S_{DS} 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

³ An alternative approach for purpose (a) that is **NOT RECOMMENDED** would be to measure the project work output as the simple total of the sizes of the three components ($S_{UI} + S_{BR} + S_{DS}$) and to use this total to compute a project productivity figure. The approach is not recommended as the proportion of the total size due to the number of data movements between the components can vary significantly from one application to another. This adds another difficulty to the task of finding good comparators.

component. There is no standard way of doing this, so external benchmarking at the major component level has this inherent problem.

These two possible benchmarking aims and the use of two possible language/technology combinations for the major components (all the same, or different) lead to three possible combinations for how to compare against external benchmarks, as in the table below.

Purpose of External Benchmarking	Programming Language/Technology for each Component	
	Same	Different
a) Compare the performance of the project to develop the whole software system, ignoring the multi-layer architecture	OK to compare project performance against other 'whole project' results as in section 2.1 and 2.2	This is a 'multi-platform' project in ISBSG terminology. Need to find other projects with a similar language /technology mix to ensure comparability
b) Compare the performance of the individual sub-projects to develop each separate major component	Compare the performance of each sub-project against other projects or sub-projects to develop major components, each using the same language /technology mix. Be careful to ensure 'common effort' has been dealt with in the same way by the comparator projects (i.e. either all ignored or all allocated)	

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.

MEASUREMENT PATTERNS FOR REAL-TIME APPLICATION SOFTWARE

Two measurement patterns are defined for application software of real-time systems:

- A patterns for a real-time application considered as a whole (section 3.1), or for a simple real-time embedded application (section 3.2).
- The human operator workstation of a real-time application (section 3.3)

If the task is to measure separately the major components of a distributed real-time application, the pattern of section 3.1 should be used for each major component as appropriate, except that the pattern of section 3.3 should be used if there is a major component of the distributed application that is a human operator workstation.

The factors to consider when there is a goal to benchmark (externally or internally) the performance of developing the various major components of a distributed real-time application are the same as those given in section 2.3 for a distributed business application.

3.1 A real-time application

This pattern should be used when the need is to determine the size of a typically large-scale, real-time application, seen as a whole, with hardware devices and other interfacing software as its functional users.

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, etc,

The typical pattern for a real-time application is shown below. The pattern shows the real-time application interacting with many possible types of functional users:

- 'dumb' hardware input and output devices that only provide and accept data respectively
- 'intelligent' hardware devices, i.e. devices that can receive data and do something with it, or that must be told what data to send
- other interfacing software.

Note that triggering Entries may come from any of these functional users except dumb output devices.

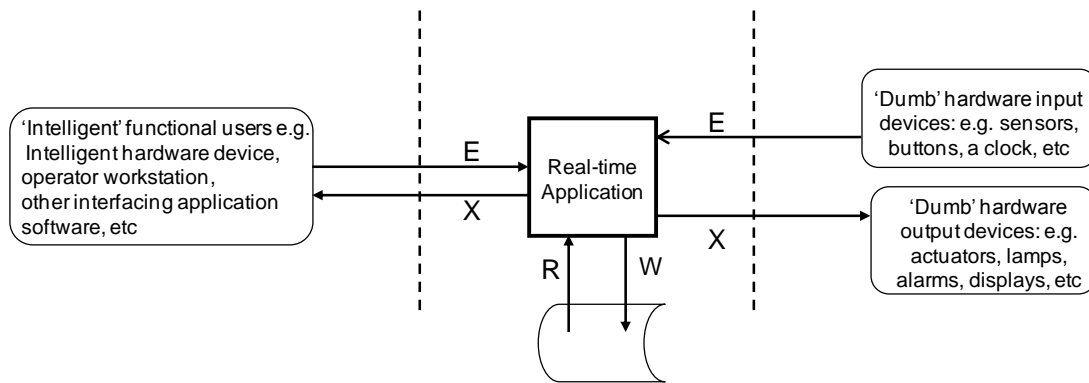


Figure 3.1 - A real-time application seen as a whole

The real-time application may interact with its hardware and software functional users either directly or 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.

Level of decomposition: The Whole (i.e. no decomposition)

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

Suitability for internal benchmarking: Suitable.

3.2 Simple real-time embedded software

It is impossible to define 'simple' real-time embedded software precisely, because the technology evolves continuously. The intention of this category is that 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,
- parts of a vehicle (e.g. the air conditioning, lighting, tire pressures, etc.),
- a simple, single elevator, and such-like.

The distinction between software in this category and 'real-time application software' described in section 3.1 is only a matter of scale, i.e. software size. Software in this category may typically have to support a limited number of 'dumb' input/output devices. A microprocessor with embedded software may be part of a large-scale distributed real-time process control system, e.g. to control a power plant. Conversely, large-scale real-time systems as described in section 3.1 may have to interface with thousands of 'intelligent' input/output devices containing embedded software and/or many other software systems.

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. Increasingly such microprocessors 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 4.2.

The general pattern for simple real-time embedded application software that interacts directly with 'dumb' hardware input or output devices is the same as Figure 3.1 in section 3.1. This same pattern applies even if the simple real-time embedded application interacts with its 'dumb' hardware users via an operating system.

A simple embedded application may also communicate with other application software functional users as part of a distributed real-time application.

Level of decomposition: The Whole (i.e. 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.

Suitability for internal benchmarking: Suitable.

3.3 The operator workstation application of a real-time software system

The pattern for the application software of a human operator workstation used to monitor or control a real-time application such as described in section 3.1 is shown in Figure 3.2 below.

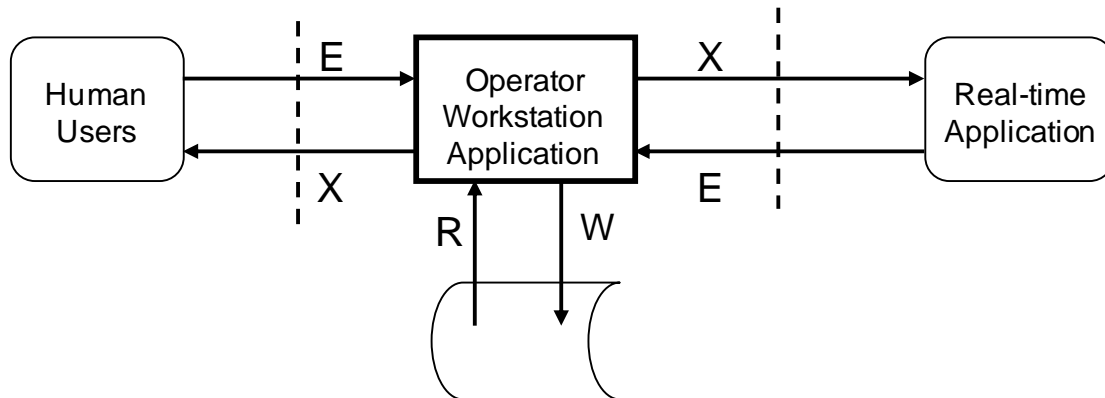


Figure 3.2 - The human operator workstation of a real-time software system

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 pattern is essentially the same as that for an on-line business application as shown in section 2.1. 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: The Whole (i.e. no decomposition)

Suitability for external benchmarking: Suitable. This pattern is effectively identical to the pattern for an on-line business application with humans and interfacing applications as its users (see section 2.1). However, 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.

Suitability for internal benchmarking: Suitable.

MEASUREMENT PATTERNS FOR INFRASTRUCTURE SOFTWARE

Two types of measurement patterns are described

- For re-usable software components
- For the software to drive various types of dumb and intelligent hardware devices

4.1 Re-usable software components

This category includes re-usable software such as object-classes and components of a Service-Oriented Architecture⁴. These are 'minor components' in the terminology explained in section 1.2.2, usually at the lowest level of decomposition.

The pattern for a re-usable software component is shown in Figure 4.1 below.

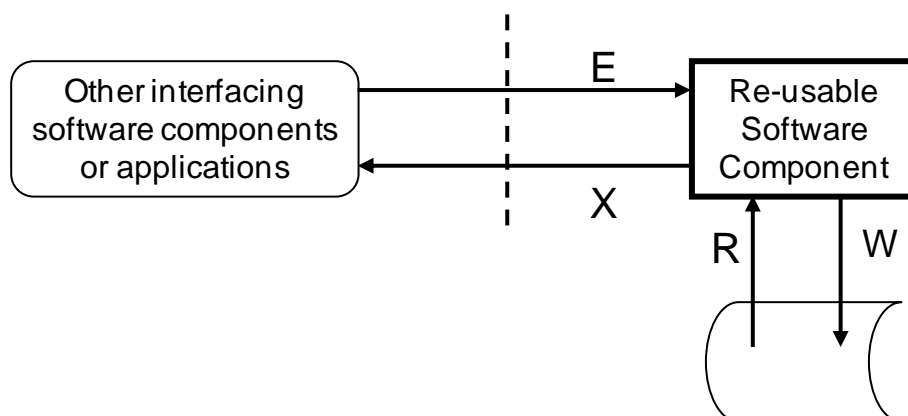


Figure 4.1 - Re-usable software component

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.

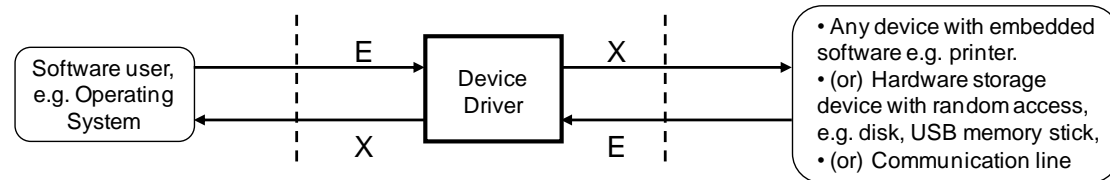
Suitability for internal benchmarking: Suitable.

⁴ See the 'Guideline for sizing Service-Oriented Architecture Software'

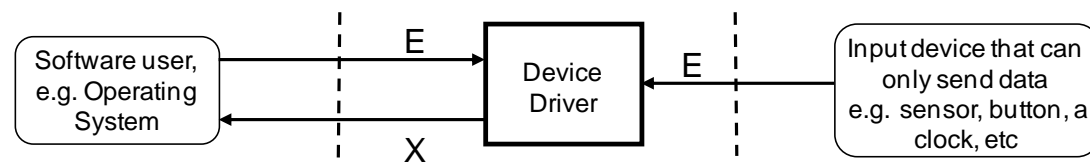
4.2 Hardware device driver software

Three categories of device-driver software can be defined. Their respective patterns are shown in Figure 4.2 below.

a) An 'Intelligent' Input / Output Device



b) A 'Dumb' Input Device



c) A 'Dumb' Output Device

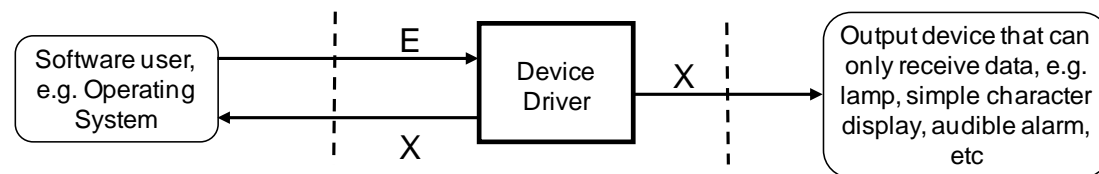


Figure 4.2 - Hardware device driver software

Level of decomposition: These are 'minor components' that cannot be decomposed further.

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

Suitability for internal benchmarking: Suitable.

SUMMARY OF MEASUREMENT PATTERNS

The proposed set of measurement patterns can be summarized as in the table below. The numbers refer to the sections of this Guideline where the measurement pattern is described.

Level of Decomposition (or Scale)	Functional Domain		
	Business Application	Real-time Application	Infrastructure Software
Whole (or 'monolithic')	2.1 (On-line app.) 2.2 (Batch app.)	3.1 for large and small-scale applications (see also section 3.2). Distinguish different scales of apps locally (3.3 for an operator workstation)	(Currently no pattern is defined, but a generalized version of 4.1 should be usable.)
Major component	2.3		
Minor component	4.1	3.1 or 4.1	4.1 or 4.2 a), b) & c)

The main points to learn from this table are:

- Although nine measurement patterns are proposed that should meet all of the most common project performance and benchmarking needs, many organizations will specialize in developing software for one specific functional domain, so will not need all of these patterns
- It is very important to distinguish performance measurements made at different levels of decomposition. The three 'whole/major/minor' levels defined in this Guideline are quite commonly distinguished in the domain of business application software. However, in the domains of real-time and infrastructure software it is harder 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.

USING MEASUREMENT PATTERNS IN PRACTICE

The purpose of this chapter is to give examples of how measurement patterns 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 these patterns should be reported.

6.1 Example of measuring a project that delivers 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 patterns from sections 2.1 and 2.2 are used to model the on-line front-end and the batch back-end respectively.

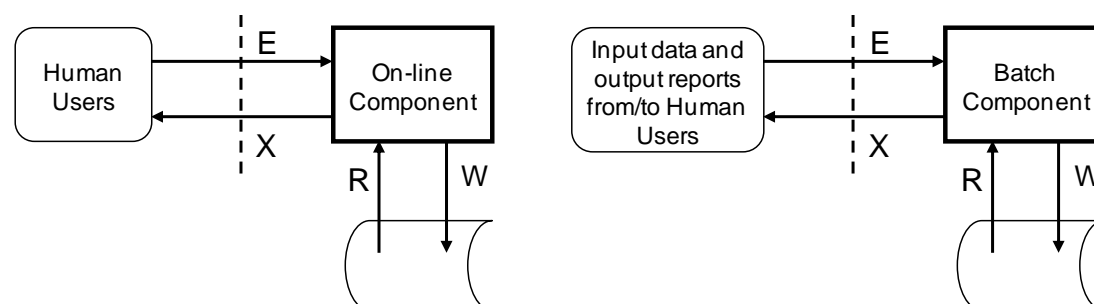


Figure 6.1 - Example of a mixed on-line/batch business application system

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 process types that:

- process each order type and update the customer's account (the orders provide the triggering Entries for each of these functional processes),
- produce the customer statements and the management reports (the triggering Entry for this part of the batch back-end, assuming no input data is needed, is effectively the 'tick of a clock' that starts the process).

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. Suppose the measured sizes for the on-line and batch parts are S_O and S_B respectively

- Derive separate productivity measurements for each part of the application using the sizes S_O and S_B 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 S_O and S_B 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, taking into account 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 measurement scopes.)

6.2 Example of an application that may be viewed as single- or multi-layered

An on-line business application is developed using various layered architecture models. Which measurement patterns should be used for which purposes?

- a) The purpose is to measure the application as a whole. The software lies wholly in the application layer of a computer system. Its functional users are humans and perhaps other interfacing applications. Use the measurement pattern of section 2.1
- b) This same application happens to be developed using a three-layer (or 'three-tier') architecture. The application's three major components are the user interface, the business rules and the data server, each in its own layer. The purpose is to measure the three components separately. Use the measurement pattern of section 2.3.
- c) Each of the three major components is developed to include re-usable components that were developed according to a Service-Oriented Architecture which has its own layer structure. The purpose is to measure any of the re-usable components separately. Use the measurement pattern of section 4.1.

Note: it would **not** make sense to define a measurement purpose and it would be a **mistake** to define a single measurement scope covering:

- any two components of the software with the three-layer architecture in case b) above. (Either measure the software as a whole, as per case a) or measure each of the three components separately, as per case b). The scope of a measurement must be confined to one layer
- or two or more components in the SOA architecture of case c) above. (Measure each component separately as per case c). The functional user of a SOA component can only 'see' the functionality of that component; it cannot 'see' any functionality beyond that component.)
- or a size that necessitated combining any of the above three software architectures.

6.3 A business application that uses re-usable components

A pattern for a business application that uses two SOA (Service-Oriented Architecture) components **could** be drawn as shown in Figure 6.2 below

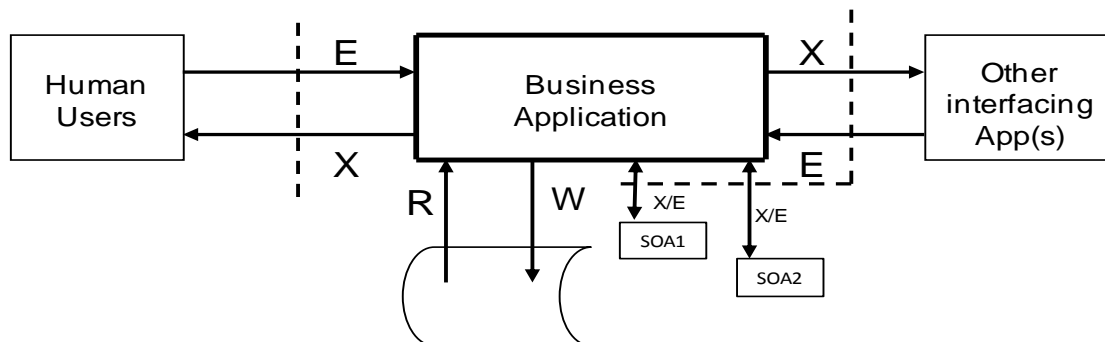


Figure 6.2 - A business application that uses re-usable components

Drawing the pattern this way implies that the SOA components are functional users of the Business Application. Using this pattern, the size of the application must include all the Entries and Exits for all interfacing SOA components.

Whether or not this model is the correct one to use depends on the **purpose** of the measurement.

If the purpose is to estimate the effort to develop the business application (excluding the SOA components) then this model could be useful. The functionality to be developed will include the data movements across the interface to the SOA components, but not the functionality that these components provide

Alternatively, if the purpose is to measure a size of the whole business application for external performance benchmarking and the SOA components 1 and 2 are being re-used (i.e. they already existed) then the SOA components should be considered as **within** the scope of the software being measured. The data movements across the internal interfaces to these components should be **ignored**. The reference pattern of sections 2.1 or 2.2 should be used - **not** the model of Fig. 6.3. The re-use of SOA components should be recorded and taken into account in the benchmark comparisons.

The reason for this is that the only way to get fair comparisons of performance across projects developing whole new business applications is to measure sizes from the viewpoint of the same types of functional users (humans and other interfacing applications) as per the patterns of section 2.1 or 2.2. In this view of a whole business application, the SOA components are not functional users of the business application. They are part of, i.e. within the scope, of application A.

For measuring the size of the SOA components, see section 4.1.

6.4 Reporting measurement results obtained using measurement patterns

Section 5.2 of the Measurement Manual gives the Rules for reporting COSMIC functional size measurements. In light of the proposals in this Guideline, the two parameters that must be added to this list concern whether the software executes in on-line or batch mode and some measure or account of the re-usable software that was used.

It would also be helpful to record the measurement pattern used for the measurement as defined in this Guideline, or the local pattern that was used.

APPENDIX - COSMIC CHANGE REQUEST AND COMMENT PROCEDURE

The COSMIC Measurement Practices Committee (MPC) is very eager to receive feedback, comments and, if needed, Change Requests for this guideline. This appendix sets out how to communicate with the COSMIC MPC. All communications to the COSMIC MPC should be sent by e-mail to the following address:

mpc-chair@cosmicon.com

Informal general feedback and comments

Informal comments and/or feedback concerning the guideline, such as any difficulties of understanding or applying the COSMIC method, suggestions for general improvement, etc should be sent by e-mail to the above address. Messages will be logged and will generally be acknowledged within two weeks of receipt. The MPC cannot guarantee to action such general comments.

Formal change requests

Where the reader of the guideline believes there is a defect in the text, a need for clarification, or that some text needs enhancing, a formal Change Request ('CR') may be submitted. Formal CR's will be logged and acknowledged within two weeks of receipt. Each CR will then be allocated a serial number and it will be circulated to members of the COSMIC MPC, a world wide group of experts in the COSMIC method. Their normal review cycle takes a minimum of one month and may take longer if the CR proves difficult to resolve. The outcome of the review may be that the CR will be accepted, or rejected, or 'held pending further discussion' (in the latter case, for example if there is a dependency on another CR), and the outcome will be communicated back to the Submitter as soon as practicable.

A formal CR will be accepted only if it is documented with all the following information.

- Name, position and organization of the person submitting the CR.
- Contact details for the person submitting the CR.
- Date of submission.
- General statement of the purpose of the CR (e.g. 'need to improve text...').
- Actual text that needs changing, replacing or deleting (or clear reference thereto).
- Proposed additional or replacement text.
- Full explanation of why the change is necessary.

A form for submitting a CR is available from the www.cosmicon.com site.

The decision of the COSMIC MPC on the outcome of a CR review and, if accepted, on which version the CR will be applied to, is final.

Questions on the application of the COSMIC method

The COSMIC MPC regrets that it is unable to answer questions related to the use or application of the COSMIC method. Commercial organizations exist that can provide training and consultancy or tool support for the method. Please consult the www.cosmicon.com web-site for further detail.