COSMIC-FFP OVERVIEW

COSMIC, the COmmon Software Measurement International Consortium, is a voluntary initiative of a truly international group of software measurement experts, both practitioners and academics, from Asia/Pacific, Europe and North America. The principles of the COSMIC-FFP method of measuring the functional size of software were first laid down in 1999. Field trials were successfully conducted in 2000/01 with several international companies and academic institutions. The process of developing an International Standard for the COSMIC-FFP method was started in 2001 and adopted and published by ISO in 2003. For further information about COSMIC visit www.cosmicon.com, or www.gelog.etsmtl.ca/cosmic-ffp, including for a free download of the COSMIC Implementation Guide to ISO 19761:2003, measurement bulletin updates, case studies, certification, etc.

The COSMIC-FFP measurement method is designed to be applicable to software from the following domains:

- Business application software which is typically needed in support of business administration, such as banking, insurance, accounting, personnel, purchasing, distribution or manufacturing. Such software is often characterized as "data rich", as its complexity is dominated largely by the need to manage large amounts of data about events in the real world.
- Real-time software, the task of which is to keep up with or control events happening in the real world. Examples would be software for telephone exchanges and message switching, software embedded in devices to control machines such as domestic appliances, lifts and car engines, for process control and automatic data acquisition, and within the operating system of computers.
- Hybrids of the above, as in real-time reservation systems for airlines or hotels for example.

For software, which:

- are characterized by complex mathematical algorithms or other specialized and complex rules, such as may be found in expert systems, simulation software, self-learning software, weather forecasting systems, etc.
- process continuous variables such as audio sounds or video images, such as found, for instance, in computer game software, musical instruments and the like.

it is possible, to define local extensions to the COSMIC-FFP measurement method

COSMIC-FFP Measurement Process Model

The derivation of functional size of the software being measured is independent of the effort required to develop or maintain the software, of the method used to develop or maintain the software and of any physical or technological components of the software.

The COSMIC-FFP measurement method is also designed to be independent of the implementation decisions embedded in the operational artifacts of the software to be measured. To achieve this characteristic, measurement is applied to the Functional User Requirements (or 'FUR') of the software to be measured - Figure 1.

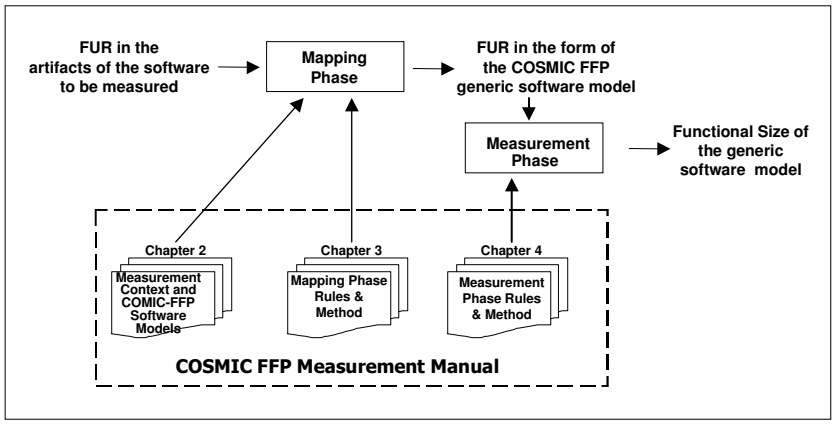


Figure 1 – COSMIC-FFP measurement process model – Version 2.2 (January 2003)

Extracting functional users requirements

The functionality delivered by software to its users is described through the Functional User Requirements (FUR). These state 'what' the software must do for the users and exclude any technical or quality requirements that say 'how' the software must perform. In practice, FUR sometimes exist in the form of a specific document (requirements specifications, for instance), but often they have to be derived from other software engineering artifacts. As illustrated in Figure 2, FUR can be derived from software engineering artifacts that are produced before the software exists (typically from architecture and design artifacts). Thus, the functional size of software can be measured prior to its implementation on a computer system.

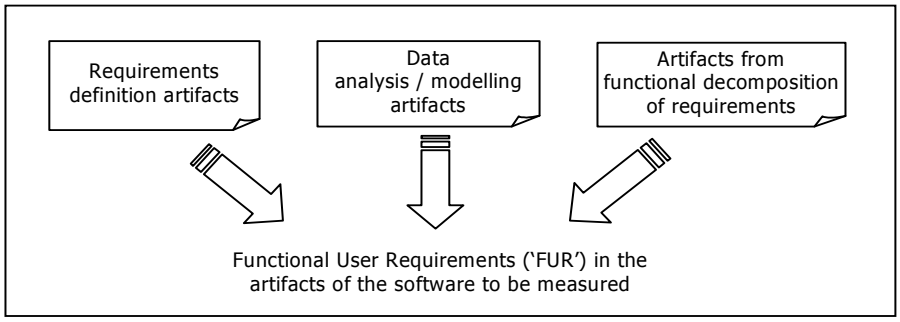


Figure 2 – COSMIC-FFP pre-implementation model

In other circumstances, software might be used without there being any, or with only a few, architecture or design artifacts available, and the FUR might not be documented (legacy software, for instance). In such circumstances, it is still possible to derive the software FUR from the artifacts installed on the computer system even after it has been implemented, as illustrated in Figure 3.

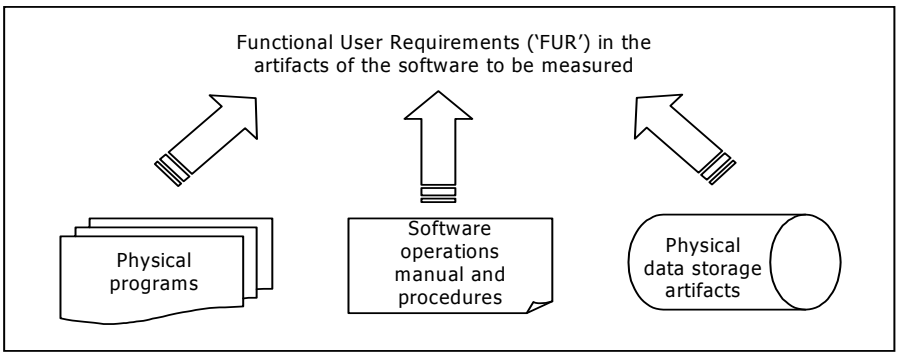


Figure 3 – COSMIC-FFP post-implementation model

COSMIC-FFP Mapping phase

The COSMIC-FFP mapping phase takes as input a statement of Functional User Requirements of a piece of software and, using a defined set of rules and procedures, produces a specific software model suitable for measuring functional size. The software model produced corresponds to the set of the FUR to be included in the specific FSM measurement exercise, as determined by the Purpose, Scope and Measurement Viewpoint of the measurement.

A key aspect of software Functional Size Measurement is the establishment of what is considered to be part of the software and what is considered to be part of the software's operating environment. Figure 4 illustrates the generic flow of data from a functional perspective from which the following can be observed:

- Software is bounded by hardware. In the so-called "front-end" direction, software used by a human user is bounded by I/O hardware such as a mouse, a keyboard, a printer or a display, or by engineered devices such as sensors or relays. In the so-called "back-end" direction, software is bounded by persistent storage hardware like a hard disk and RAM and ROM memory.

- The functional flow of data attributes can be characterized by four distinct types of movement. In the "front end" direction, two types of movement (ENTRIES and EXITS) allow the exchange of data with the users across a 'boundary'. In the "back end" direction, two types of movement (READS and WRITES) allow the exchange of data attributes with the persistent storage hardware.

- Different abstractions are typically used for different measurement purposes. For business application software, the abstraction commonly assumes that the users are one or more humans who interact directly with the business application software across the boundary; the 'I/O hardware' is ignored. In contrast for real-time software, the users are typically the engineered devices that interact directly with the software, that is the users ARE the 'I/O hardware'.

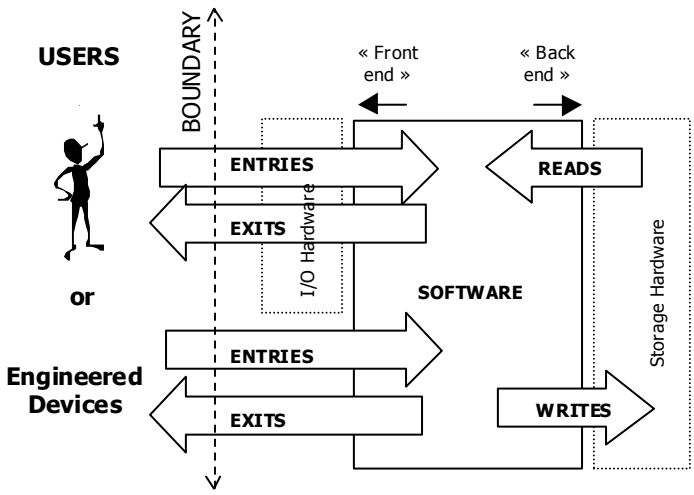


Figure 4 – Generic flow of data through software from a functional perspective

Figure 5 illustrates the generic software model proposed by the COSMIC-FFP measurement method. According to this model, software functional user requirements can be decomposed into a set of functional processes. Each of these functional processes is a unique set of sub-processes performing either a data movement or a data manipulation.

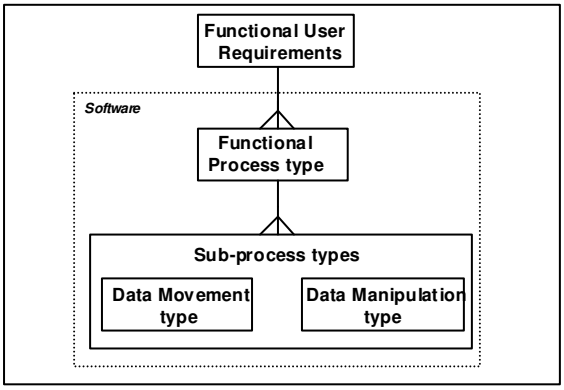


Figure 5 – A generic software model for measuring functional size

The generic COSMIC-FFP software model asserts that all software Functional User Requirements can be expressed in terms of 'functional processes', each triggered by an event in the world of the user. (See below for more on functional processes). The model further distinguishes four types of data movement: entry, exit, read and write, as defined by this measurement method. All data movements move data contained in exactly one data group. Entries move data from the users across the boundary to inside the functional process; exits move data from inside the functional process across the boundary to the users; reads and writes move data from and to persistent storage. These relationships are illustrated in Figure 6. Of course, as shown in Figure 5, software manipulates data as well as moving it. However, given that:

i) the COSMIC-FFP method is aimed at software from domains that are data movement-rich, rather than algorithm-rich and

ii) that how to measure the functional size of data manipulation is not al all clear, the method makes the simplifying assumption that each data movement type also accounts for the data manipulation associated with it.

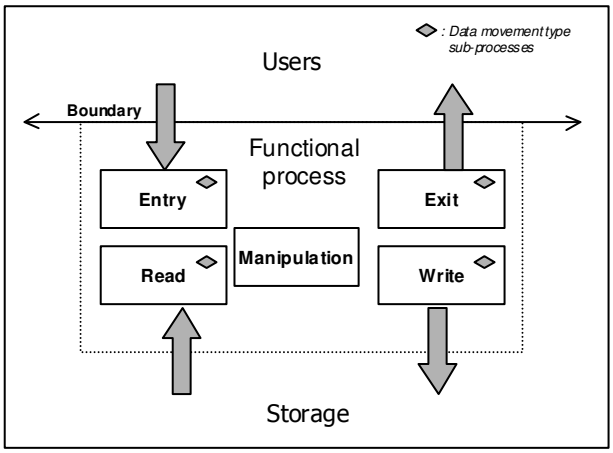


Figure 6 – The data movement types and some of their relationships

COSMIC-FFP measurement phase

The COSMIC-FFP measurement phase takes as input an instance of the COSMIC-FFP generic software model and, using a defined set of rules and procedures, produces a size based on the following principle: The functional size of software is directly proportional to the number of its Data Movements. Therefore:

- Each instance of a data movement is assigned 1 Cfsu (Cosmic functional size unit).
- The functional size of a functional process is defined as the arithmetic sum of the values of the measurement function, as applied to each of its data movements.
- The functional size of any required functional *change* to a piece of software is by convention the arithmetic sum of the functional sizes of all the added, changed and deleted functional data movements of that piece of software.

There is no upper limit to the functional size of a piece of software and, notably, there is no upper limit to the functional size of any of its functional processes.

Functional Size Measurement context

Before starting a measurement using the COSMIC-FFP method it is imperative to carefully define the Purpose, the Scope and the Measurement Viewpoint. This may be considered as the first step of the measurement process

There are many reasons to measure the functional size of software, just as there are many reasons to measure the surface areas of a house. In a particular context, it might be necessary to measure the functional size of software prior to its development, just as it might be necessary to measure the surface areas of a house prior to its construction. In a different context, it will be useful to measure the functional size of software some time after it has been put into production, just as it might be useful to measure the surface areas of a house after it had been delivered and the owner has moved in.

Likewise, measuring the functional size of software after it has been put into production entails a somewhat different measurement procedure when the required dimensions are extracted from the various artifacts. Although the nature of these artifacts differs, the dimensions, the unit of measure and the measurement principles remain the same.

Equally, it is important to define the Scope of the measurement, which is derived from the Purpose, before commencing a particular measurement exercise. Example: if the purpose is to measure the functional size of software delivered by a particular project team, it will first be necessary to define the Functional User Requirements of all the various components to be delivered by the team. These might include the FUR of software which was used once only to convert data from software which is being replaced. If then the purpose is changed to measure the size which the users have available once the new software is operational, this would be smaller, as the FUR of the software used for conversion would not be included in the scope of the measured size.

Finally, it is essential to define the Measurement Viewpoint, which again may follow from the Purpose. The Measurement Viewpoint, in general terms, determines the level of

detail that can be seen and therefore measured, within the Scope. The Measurement Viewpoint is highly significant, because in general measurements taken from different Measurement Viewpoints cannot meaningfully be compared or added together.

Purpose of a Measurement. Examples:

- To measure the size of the FUR as they evolve, as input to an estimating process
- To measure the size of changes to the FUR after they have been initially agreed, in order to manage project 'scope creep'
- To measure the size of the FUR of the total software delivered, and also the size of the FUR of the software which was developed, in order to obtain a measure of functional re-use
- To measure the size of the FUR of the existing software as input to the measurement of the performance of those responsible for maintaining and supporting the software.

The Purpose helps the Measurer to determine:

- the Scope to be measured and hence the artifacts which will be needed for the measurement
- the Measurement Viewpoint to be used for the measurement
- 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 COSMIC-FFP measurement method should be used, or whether a locally-derived approximation version of the COSMIC-FFP method should be used (e.g. early in a project's life-cycle, before the FUR are fully elaborated)

Scope of a Measurement. Examples:

- A contractually-agreed statement of FUR
- A Project Team's *delivered* work-output (i.e. including that obtained by exploiting existing software parameters and re-usable code, software used for data conversion and subsequently discarded, and utilities and testing software developed specifically for this project)
- A Project Team's *developed* work-output (i.e. including software used for data conversion and subsequently discarded, and utilities and testing software developed specifically for this project, but *excluding* all new functionality obtained by changing parameters and exploiting re-usable code)
- A Layer
- A Re-usable Object
- A Software Package
- An Application
- An Enterprise Portfolio

Specific Measurement Viewpoints.

As outlined above, for consistent measurement, measurers need to define and use consistently a very limited number of Measurement Viewpoints. The two Measurement Viewpoints that most obviously need to be standardized are the Measurement Viewpoint

of the 'End Users' of an item of application software and the Measurement Viewpoint of the Developer of the software to be provided to meet the FUR.

End User Measurement Viewpoint: A Measurement Viewpoint that reveals only the functionality of application software that has to be developed and/or delivered to meet a particular statement of FUR. It is the viewpoint of Users who are either humans who are aware only of the application functionality that they can interact with, or of peer application software that is required to exchange or share data with the software being measured. It ignores the functionality of all other software needed to enable these Users to interact with the application software being measured.

Developer Measurement Viewpoint:A Measurement Viewpoint that reveals all the functionality of each separate part of the software that has to be developed and/or delivered to meet a particular statement of FUR.

The Developer may see that a statement of FUR implies that more than one separate 'major component' has to be developed and/or delivered. This can arise if, due to the requirements, parts of the software have to be developed using different technologies, and/or will execute on different processors and/or belong to different layers of a given architecture, and/or are to be realized as separate peer items in the same layer'.

IDENTIFYING SOFTWARE LAYERS

Functional User Requirements may state explicitly, may imply, or the measurement analyst may infer, that the FURs apply to software in different layers or peer items. Alternatively, the measurement analyst may be faced with sizing existing software which appears to be in different layers or peer items. In both cases, let us assume that the Purpose, Scope and Measurement Viewpoint indicate that these layers must be measured separately. For example, the Purpose is estimating, where the layers will be developed using different technologies. We then need guidance to help decide if the FUR or the software comprises one or more layers or peer items. Layers may be identified according to the following definitions and principles. A layer is the result of the functional partitioning of the software environment such that all included functional processes perform at the same level of abstraction.

In a multi-layer software environment, software in one layer exchanges data with software in another layer through their respective functional processes. These interactions are hierarchical in nature; when considered in pairs, one layer is subordinate to the other. Software in a subordinate layer provides functional services to software in other layers using its services. The Measurement Method defines "peer-to-peer" exchanges as two items of software in the same layer exchanging data.

Layer identification is an iterative process:
The concept of software layers is a tool to help differentiate Functional User Requirements allocated at different levels of functional abstraction. There are many software architecture models in use. The layered model is used here to provide a functional view of the software. Other models could be used if they provide a functional view of the software, fully or partly.

IDENTIFYING SOFTWARE BOUNDARIES

This step consists in identifying the boundary of each piece of software (depending on the Measurement Viewpoint, e.g. each layer or each peer item within a layer if the 'Developer' Measurement Viewpoint) to be measured. The boundary is defined as a conceptual interface between the software under study and its users. The boundary of a piece of software is the conceptual frontier between this piece and the environment in which it operates, as it is perceived externally from the perspective of its users. The boundary allows the measurer to distinguish, without ambiguity, what is included inside the measured software from what is part of the measured software's operating environment.

A User is defined as any person or thing that communicates or interacts with the software (being measured) at any time. Users can be human beings, software or engineered devices.

Once identified, the candidate boundary must comply with the following principle: By definition, there is a boundary between each identified pair of layers where one layer is the user of another, and the latter is to be measured. Similarly, there is a boundary between any two distinct pieces of software in the same layer if they exchange data in 'peer-to-peer' communications; in this case each piece can be a user of its peer.

The following rules might be useful to confirm the status of a candidate boundary:

a) Identify triggering events, then identify the functional processes enabled by those events. The boundary lies between the triggering events and those functional processes.
b) For real-time or technical software, use the concept of layers to assist in the identification of the boundary (section 3.1).

IDENTIFYING FUNCTIONAL PROCESSES

This step consists in identifying the set of functional processes of the software to be measured, from its Functional User Requirements. A functional process is an elementary component of a set of Functional User Requirements comprising a unique cohesive and independently executable set of data movements. It is triggered by one or more triggering events either directly, or indirectly via an 'actor'. It is complete when it has executed all that it is required to be done in response to the triggering event (-type). An 'actor' is a User of the system being measured, acting as an intermediary to convey data about a triggering event to the functional process that has to respond to that event.

A triggering event is an event(-type) that occurs outside the boundary of the measured software and initiates one or more functional processes. In a set of Functional User Requirements, each event(-type) which triggers a functional process is indivisible for that set of FUR. (Clock and timing events can be triggering events; As far as software is concerned, an event has either happened, or it has not; it is instantaneous.

Once identified, candidate functional processes must comply with the following principles:

a) A functional process is derived from at least one identifiable Functional User Requirement,
b) A functional process is performed when an identifiable triggering event occurs,
c) A functional process comprises at least two data movements, an entry plus either an exit or a write,
d) A functional process belongs to one, and only one, layer,
e) A functional process terminates when a point of asynchronous timing is reached. A point of asynchronous timing is reached when the final (terminating) data movement in a sequence of data movements is not synchronized with any other data movement.

IDENTIFYING DATA GROUPS

This step consists in identifying the data groups referenced by the software to be measured. A data group is a distinct, non empty, non ordered and non redundant set of data attributes where each included data attribute describes a complementary aspect of the same object of interest. A data group is characterised by its persistence.

Data persistence is a characteristic used to help distinguish between the four types of sub-processes. Once identified, each candidate data group must comply with the following principles:

a) A data group must be materialized within the computer system supporting the software.
b) Each identified data group must be unique and distinguishable through its unique collection of data attributes.
c) Each data group must be directly related to one Object of interest described in the software's Functional User Requirements.

The data group definition and principles are intentionally broad in order to be applicable to the widest possible range of software. This quality has a drawback in the fact that their application to the measurement of a specific piece of software might be difficult. Therefore, the following rules, drawn from Functional Size Measurement practice, might assist in the application of the principles to specific cases.

APPLICATION TO BUSINESS APPLICATION SOFTWARE: Measurement practice has established that, in business application software, a data group is identified for each 'entity-type' In COSMIC-FFP, we use the term 'Object of interest' instead of 'entity-type' or 'TNF relation' in order to avoid using terms related to specific software engineering methods.

Examples: in the domain of management information software, an Object of interest could be 'employee' (physical) or 'order' (conceptual) – the software is required to store data about employees or orders.

Furthermore, data groups showing transient persistence are formed whenever there is an ad hoc enquiry which asks for data about some 'thing' about which data is not stored with indefinite persistence, but which can be derived from data stored with indefinite persistence. In such cases the transient Object of interest is the subject of the entry data movement in the ad hoc enquiry (the selection parameters to derive the required data)

and of the exit data movement containing the desired attributes of the transient Object of interest.

Example: we form an ad hoc enquiry against a personnel database to extract a list of names of all employees aged over 35. This group is a transient Object of interest. The entry is a data group containing the selection parameters. The exit is a data group containing the list of names.

APPLICATION TO REAL-TIME SOFTWARE. Real-time software measurement practice has established that data groups for this type of software often take the following forms:

- Data movements which are Entries from physical devices typically contain data about the state of a single Object of interest, such as whether a valve is open or closed, or indicate a time at which data in short-term, volatile storage is valid or invalid, or contain data that indicates a critical event has occurred and which causes an interrupt.
- A message-switch may receive a message data group as an Entry and route it forward unchanged as an Exit. The attributes of the message data group could be, for example, 'sender, recipient, route_code and message_content', and its Object of interest is 'Message'.

- Common data structure, representing Objects of interest that are mentioned in the Functional User Requirements, which are held in volatile memory and accessible to most of the functional processes found in the measured software,
- Reference data structure, representing graphs or tables of values found in the Functional User Requirements, which are held in permanent memory (ROM memory, for instance) and accessible to most of the functional processes found in the measured software,
- Files, commonly designated as "flat files", representing Objects of interest mentioned in the Functional User Requirements, which are held in a persistent storage device.

IDENTIFYING THE DATA MOVEMENTS

This step consists in identifying the data movements (Entry Exit Read and Write-types) of each functional process. A COSMIC-FFP data movement is a component of a functional process that moves one or more data attributes belonging to a single data group. A COSMIC-FFP data movement occurs during the execution of a functional process

APPLYING THE MEASUREMENT FUNCTION

This step consists in applying the COSMIC-FFP measurement standard to each of the data movements identified in each functional process. The COSMIC-FFP measurement standard, 1 Cfsu (Cosmic Functional Size Unit) is defined as the size of one elementary data movement.

According to this measurement function, each instance of a data movement (entry, exit, read or write) receives a numerical size of 1 Cfsu. The last step consists in aggregating the results of the measurement function, as applied to all identified data movements, into a single functional size value:

For each functional process, the functional sizes of individual data movements are aggregated into a single functional size value by arithmetically adding them together.

$$\text{Size}_{\text{Cfsu}}(\text{functional process}_i) = \sum \text{size}(\text{entries}_i) + \sum \text{size}(\text{exits}_i) + \sum \text{size}(\text{reads}_i) + \sum \text{size}(\text{writes}_i)$$

Note that the minimum size of a functional process is 2 Cfsu (There must always be one Entry and either a Write or an Exit) and theere is no upper limit to the size of any one fucntional process.

For any functional process, the functional size of changes to the Functional User Requirements is aggregated from the sizes of the corresponding modified data movements according to the following formula.

$$\text{Size}_{\text{Cfsu}}(\text{Change}(\text{functional process}_i)) = \sum \text{size}(\text{added data movements}) + \sum \text{size}(\text{modified data movements}) + \sum \text{size}(\text{deleted data movements})$$

The size of each piece of software to be measured within a layer shall be obtained by aggregating the size of the new and any changed functional processes within the identified FUR for each piece

Sizes of layers or of pieces of software within layers may be added together only if measured from the same Measurement Viewpoint.

Furthermore, sizes of pieces of software within any one layer or from different layers may be added together only if it makes sense to do so, for the Purpose of the measurement. (For example, if various major components are developed using different technologies, by different project sub-teams, then there may be no practical value in adding their sizes together.)

An example: A requested change to a piece of software might be: “add one new Functional Process of size 6 Cfsu, and in another Functional Process add one Data Movement, make changes to three other Data Movements and delete two Data Movements.” The total size of the requested change is 6 + 1 + 3 + 2 = 12 Cfsu.

LOCAL EXTENSIONS

The COSMIC-FFP Measurement Method is currently not designed to provide a standard way of accounting for the size of certain types of Functional User Requirements, notably complex mathematical algorithms or complex sequences of rules as found in expert systems. However, it may be that within the local environment of an organization using the COSMIC-FFP Measurement Method, it would be possible to account for this functionality in a way which is meaningful as a local standard.

For this reason, the COSMIC-FFP Measurement Method has provision for local extensions. In any functional process where there is an abnormally complex data manipulation functional sub-process, the measurer is free to assign his or her own locally-determined functional size units for this exceptional functionality. An example of a local extension standard could be:

In our organization, we assign one Local FSU for mathematical algorithms such as (list of locally meaningful and well-understood examples…). We assign two Local FSU's for (another list of examples), etc.

USING COSMIC-FFP EARLY IN THE LIFE CYCLE

It may be necessary in practice to determine a COSMIC-FFP size early in a project life-cycle before all detailed information has become available to produce a size according to the detailed rules given in the Measurement Manual.

In these circumstances we can use a locally-calibrated approximate version of the COSMIC-FFP method to obtain the early size estimate. Any approximate COSMIC-FFP method relies on finding a concept at a higher level of abstraction than the Data Movement, that can be assigned a size in Cfsu.

The first higher level concept above the Data Movement is the Functional Process. The simplest process for obtaining an approximate size of a new piece of software is therefore as follows. For the new piece of software:

1. Identify a sample of other pieces of software with similar characteristics to the new piece
2. Identify their functional processes
3. Measure the sizes of the functional processes of these other pieces with COSMIC-FFP
4. Determine the average size, in Cfsu, of the functional processes of these other pieces (e.g. = 8 Cfsu)
5. Identify all the functional processes of the new piece of software (e.g. = 40)
6. Based on the sample the early estimated size of the new piece of software is 8 x 40 = 320 Cfsu.

Such a process can be refined to give a more accurate result if instead, before step 4 above, the functional processes are sorted into categories giving equal contributions to the total size. In a real example for one component of a major real-time avionics system, it was decided to divide the functional processes into four quartiles of equal contribution to size. The average size of the functional processes in each quartile (and the names given to these quartiles) was:

‘Small’ 3.9 Cfsu

‘Medium’ 6.9 Cfsu

‘Large’ 10.5 Cfsu

‘Very Large’ 23.7 Cfsu

(To interpret these figures, for example, 25% of the total size of the component was accounted for by ‘Small’ functional processes whose average size was 3.9 Cfsu, another 25% of the total size by ‘Medium’ functional processes of average size 6.9 Cfsu, etc.)

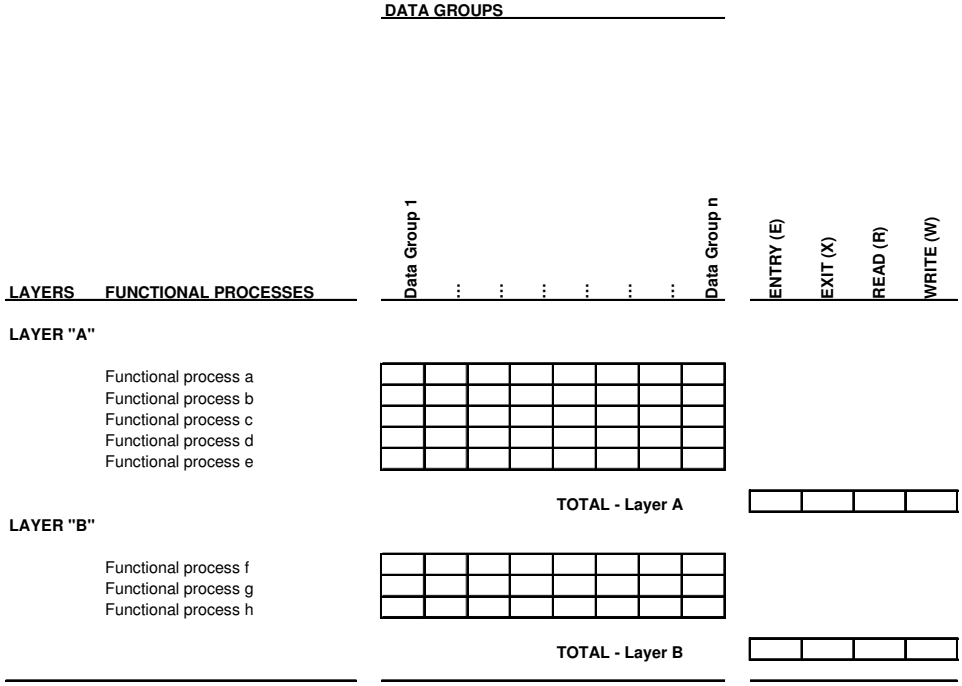
In step 5 of the above process, the functional processes of the new piece of software are identified and also classified as ‘Small’, ‘Medium’, ‘Large’ or ‘Very Large’.

In step 6, the average sizes listed above are then used to multiply the number of functional processes of the new piece of software, in each quartile respectively to get the total early estimated size.

N.B. Anyone wishing to adopt this process is strongly advised to calibrate their average sizes with local data relevant to the piece of software to be sized (and NOT to use the above averages). Much further research and measurement is needed before it will be possible to give ‘industry-average’ sizes of functional processes for various circumstances.

Measurement worksheet example

The matrix next can be used as a repository to hold each identified component of the generic software model to be measured. It is designed to facilitate the use of the measurement process.



- Each identified data group is registered in a column,
- Each functional process is registered on a specific line, grouped by identified layer.
- For each identified functional process, the identified data movements are noted in the corresponding cell using the following convention: “E” for an entry, “X” for an exit, “R” for a read and “W” for a write;
- For each identified functional process, the data movements are then summed up by type and each total is registered in the appropriate column at the far right of the matrix;
- The measurement summary can then be calculated and registered in the boxed cells of each layer, on the “TOTAL” line.