Advancing Functional Size Measurement – which size should we measure?

Charles Symons

'COSMIC' (The Common Software Measurement International Consortium)

Abstract

A tape measure is calibrated showing a standard unit of measure, e.g. the centimetre. But if you use a tape to measure an irregularly shaped object, then you have a choice of sizes that can be measured.

The same is true for a piece of software, which has many possible functional sizes. 'Traditional' or '1st generation' Functional Size Measurement (FSM) Methods such as the IFPUG method and its variants, the MkII FPA method, etc., have not properly recognised this problem because they were designed to be applicable only in the domain of business application software – though the problem exists to a degree even using these methods in this domain.

The '2nd Generation' COSMIC method was designed to be applicable to measure sizes of any 'data-rich' software, i.e. of business and real-time software. It has been applied to measure functional sizes of complex real-time software architectures such as found in large-scale telecoms and avionics systems. In such environments the measurer is faced with sizing requirements that may be spread over multiple layers of software, that are expressed from different viewpoints and at different levels of granularity, and that will execute on multiple technical platforms. Consequently, many valid functional sizes can be measured for such software, all with the same unit of measure.

Clearly, if this problem were to remain unresolved, it would be impossible to be sure that size measurements from different sources could be meaningfully compared, even on the same piece of software.

This paper will introduce the measurement challenges and will propose a set of concepts and standards designed to ensure the comparability of functional size measurements from different sources. This is vital if such measurements are to be used for reliable comparisons of performance, and for estimating and benchmarking. The proposals are valid for any FSM Method. Adoption of these concepts and standards by the software metrics community should lead to greater credibility and acceptance of FSM Methods, with potentially enormous benefits for the software industry.

These concepts and standards have been developed by members of the COSMIC Measurement Practices Committee, on whose behalf this paper is presented.

1. Introduction

Perhaps the three best-known methods of measuring a functional size of software are the so-called '1st Generation' IFPUG [1] and MkII FPA [2] methods and the '2nd Generation' COSMIC [3] method. All three methods assume that the functional user requirements[1] of some software to be

[1] This paper uses the terminology of the standard ISO/IEC 14143/1:2007 'Functional Size Measurement – Part 1 Definition of Concepts', notably for terms such as 'functional user requirements', functional size', 'user', 'BFC', etc. The paper assumes a basic understanding of functional size measurement methods and the terminology used.

measured can be analysed so as to identify certain types of 'base functional components' (or BFC's) defined by the method. Each method assumes a size unit of measure e.g. a 'function point' and assigns a fixed number of size units to each BFC-type. The functional size of a given piece of software according to the method is then obtained by adding up the sizes of all the BFC's identified in the software's functional user requirements (or 'FUR').

As an example, all three methods define a type of component of functionality that is essentially the same, but is given a different name in each method, i.e. the 'elementary process' (IFPUG), the 'logical transaction' (MkII FPA) and the 'functional process' (COSMIC). Each method then classifies or sub-divides this component so as to define the method's BFC-types which are unique to each method. The size units that are attributed to the BFC's are also unique to each method, but the underlying approach of 'functional size measurement' (FSM) methods is the same for all methods.

All FSM Methods therefore behave like any measurement method. They define a general size concept ('functional size', a size based purely on functionality, ignoring any technical or quality requirements) and a unit of measure[2], e.g. 'a COSMIC function point', where the latter varies with the method. So far, so good.

However, a piece of software is like an irregular object and, as we shall discuss below, it has many possible sizes for its different dimensions. The general problem may be illustrated by supposing two pieces of software A and B and two associated functional size measurements S_A and S_B, measured using the same FSM Method. The question we must be able to answer is 'is the ratio S_A / S_B a valid and fair measure of the relative sizes, or is it a meaningless because the sizes S_A and S_B are of different dimensions?'

FSM Methods *in general*, have hitherto given only limited guidance on 'which' size to measure in various circumstances.

The IFPUG method, for example, distinguishes procedures for measuring

- The work-output of a development project (the size of the functionality delivered with the first installation of the software when the project is complete)
- The work-output of an enhancement project (the size of the functionality of some existing software that has been changed when the project is complete)
- an application (the size of the functionality of an installed application, which is updated every time the application is enhanced)

The MkII FPA method makes a similar set of distinctions.

The COSMIC method, up to its current publicly-available version 2.2, recognized that more parameters need to be considered to define a particular type of measurement. But with growing experience in use of the method, a better understanding has been gained of the challenges of functional size measurement allowing more refinements to be introduced. The next version 3.0 of the method[3], to be published in the first half of 2007, will include what COSMIC now believes to be a much more comprehensive set of parameters that must be considered when defining 'which' functional size should be measured of a given piece of software.

---

[2] To be strictly accurate 1st Generation FSM Methods do not define a *unit* of measure. Neither the IFPUG method nor the MkII FPA method define *one* 'Function Point'. For example, IFPUG BFC's are assigned sizes from 3 to 15 Function Points

[3] From version 3.0 of the method, its name will be simplified from the 'COSMIC-FFP' method to the 'COSMIC' method. the name of the unit of measure is being changed to a 'COSMIC Function Point', abbreviated as 'CFP'

The process of defining this set of parameters for a given measurement is known as determining the 'Measurement Strategy'.

The purpose of this paper is to give a preview of the Measurement Strategy concepts and the process as it will be described in the COSMIC Measurement Manual version 3.0 when it is published. At the time of writing this paper, the description given below has not been completely finalised, but it is believed that the final description will not change in any important respects.

The ideas proposed herein have been developed by members of the COSMIC Measurement Practices Committee, on whose behalf this paper is presented.

It is important to recognise that the Measurement Strategy parameters and its process are totally independent of the COSMIC method's view of 'how' to measure a functional size of a piece of software. The process, the parameters and the proposed standards of the Measurement Strategy that determine 'which' size to measure should be valid in principle for any FSM Method, regardless of 'how' it measures a functional size. (Some of the parameters given below are described as 'defined in the COSMIC Measurement Manual v3.0', only because the definitions have been taken from that publication.)

Another important point to emphasise is the economic importance of these Measurement Strategy parameters and proposed standards. Functional size measurements are routinely used as input to project estimating methods and as measures of work-output for software projects. It is highly desirable therefore for all parties involved in FSM measurement (e.g. suppliers and users of FSM Methods, benchmarking services, estimating tools, etc) to agree on these parameters and proposed standards. Only with such agreements will it be possible to be sure that any two functional size measurements from different sources can reliably be compared, even if made with the same FSM Method.

2. The parameters of a Measurement Strategy for a piece of software

A given piece of software can have many possible functional sizes depending on several factors. The choice of which size to measure depends entirely on the Purpose of the measurement. As an example, consider two Purposes

A. A distributed application system is to be developed whose major components will execute on different technical hardware. The measurement purpose is to obtain sizes of each component separately that can be input to an estimating tool that will take into account the different technical platforms when estimating the total effort to develop the application

B. The same application as in A will be developed partly from new code and partly from re-used or packaged software. The purpose is to measure the proportion of re-used functionality of the whole application (i.e. ignoring its breakdown into components) in order to compare the actual re-use against a target for re-use

The total size of the application resulting from these two purposes will not be identical and the sizes of the constituents will of course be quite different.

Setting a Measurement Strategy involves determining three main parameters for the piece of software to be measured

- the scope of the piece of software
- the viewpoint of the user of the software

- the level of granularity of the functional user requirements of the software to be measured.

Although these parameters should be determined in this sequence, each depends on the measurement purpose. Determining the purpose is therefore the first and most important step of setting a Measurement Strategy.

2.1 The 'scope' of the software being measured

The 'scope' of a functional size measurement is defined in the COSMIC Measurement Manual *as "the set of functional user requirements to be included in a specific functional size measurement instance".* The scope is determined by examining the two groups of parameters below.

2.1.1 The 'level of decomposition' of the software to be measured

This parameter is defined in the COSMIC Measurement Manual v3.0 as *''any level of division of a piece of software showing its components, sub-components, etc".*[4]

The reason this parameter is significant is that functional sizes of the components of a piece of software 'X' cannot be simply added up to obtain the size of the whole piece 'X'. This is because the size of any one component will include an allowance for the messages it exchanges with other components (according to all FSM Methods). When the sizes of the major components of a piece of software X are measured separately and these sizes are added up, their total will exceed that of the size of X measured as a whole, due to the size contributions of the inter-component message exchanges.

Clearly the finer the sub-division of the whole X (i.e. the more components it is divided into), the greater will be the disparity between the sum of the sizes of the components and the size of the same software X measured as a whole.

Ideally one would like to standardise certain levels of decomposition, but these are very difficult to define precisely. Possible candidates for standard levels of decomposition that must be useful in practice, e.g. for estimating are:

- A whole application
- A major component of a distributed application that executes on a single hardware platform, that is different from the platform on which other major components execute
- A re-usable object-class

The problem with this set is the starting point. What one organization defines as an 'application' might not be considered as such by another organization. Other possible hierarchies of levels of decomposition are even harder to define precisely. If two organizations happen to define their hierarchies using the same terms such as 'system', sub-system', 'program', and 'module', there can be no certainty that these levels correspond at any levels since these terms can be interpreted in many ways.

2.1.2 The possible components of the delivered size

[4] The reader's attention is drawn to the definitions of two terms that must be carefully distinguished. 'Level of decomposition' relates to the software itself. 'Level of granularity' (defined later in this paper) relates to the description, e.g. the functional user requirements, of the software

For any piece of software at any level of decomposition as defined above, the functionality of the software delivered by a project team (i.e. the team's 'work-output') may consist of

- Newly developed functionality
- Changes to existing functionality
- Existing functionality that has been re-used, unchanged[5]

This classification into three groups is important in practice for purposes such as estimating, because each group will normally be associated with a different productivity.

The functional sizes that can be measured and that should be distinguished are as follows

a) The size of newly developed functionality
b) The size of functionality that has been changed
c) The size of changes to functionality that has been changed
d) The size of re-used, existing functionality that is delivered unchanged
e) The size of functionality after it has been changed.

We can now see the limitations of the advice given by the IFPUG and MkII FPA methods on measurement of 'different types of sizes'. Both methods define the work-output of a 'development project' as newly-*delivered* functionality, without distinguishing whether it is newly-developed, or reused, existing functionality.

Both methods define the work-output of an 'enhancement project' as the sum of added, modified and deleted functionality. However, the IFPUG method measures size b) above of an enhancement whereas the MkII FPA method, which could measure size b), recommends instead measuring size c) as a better measure of the work-output of an enhancement project (as does the COSMIC method).

A high proportion of software projects nowadays involve some re-use of existing software in the form of application packages, or of 'COTS' (Commercial Off-The Shelf) software, or of re-usable object-classes, together with some newly-developed or changed software. Understanding the sizes of these various contributions to the total delivered size becomes increasingly important for performance measurement and estimating.

2.2 The 'viewpoint' of the 'user' of the software being measured

The 2007 edition of the ISO/IEC 14143-1 standard on functional size measurement concepts defines the 'user' of a piece of software as *"any person or thing that communicates with or interacts with the software at any time".*

If we now take a typical piece of real-time application software such as the embedded application of a multi-function printer/copier machine, we can see that the following can be interpreted as 'users' of this application according to this definition.

.

- Any human operator of the machine who interacts *indirectly* with the application via buttons, displays lights and such-like
- Any of the hardware devices that interact *directly* with the application

[5] It might be argued that this third category ought to be split between

- existing functionality that has been re-used unchanged, and
- changes to functionality that, before the project, was re-used, unchanged functionality

However, most often re-usable functionality is not changed, since doing so negates the benefits of re-usability. We therefore ignore this possible fourth category.

- Any peer application, such as the software of a PC which sends files to be printed, assuming the printer/copier is networked
- An operating system, if there is one, on which the printer/copier application relies.

The meaning of the term 'user' is therefore not altogether satisfactory in that for functional size measurement purposes, the operating system is almost never thought of as a user of an application. The reverse is true. An application is invariably a user of an operating system (assuming there is one). The Functional User Requirements or 'FUR' of an application would never normally include the requirements of the operating system as a 'user'. For one thing, the FUR of an operating system are common to all applications that use it.

This leaves us with three types of possible users of this application, namely human operators, hardware devices and peer applications.

Now consider the functionality of the printer/copier from the viewpoint of the human operator versus that of the hardware devices. Even without any detailed knowledge of the internal working of this device, it should be clear that each type of user 'sees' different functionality. Specifically, a human operator cannot 'see' all of the functions that the application must provide for the machine to work. Human operators may guess how certain functions work, such as the detectors of paper jams, ink running out, paper present/absent, the automatic tests that run on start-up, all the adjustments that follow from pressing a 'darker/lighter' button when copying, etc etc. But these functions result from direct interactions between the application and the many and various engineered hardware devices that the application drives and many of these interactions are invisible to the human user.

If we were to measure the functional size of such an application from the human operator-user viewpoint and again from the viewpoint of the engineered hardware devices as users of the application, then we would find a much smaller functional size from the human viewpoint.

Similarly, peer applications see an even more limited set of functionality (via the defined Application Program Interface) than the human user.

We conclude that the definition of the 'user' of a piece of software to be measured, and his/its 'viewpoint' needs to be more differentiated if any resulting functional size measurements are to be correctly interpreted. (This problem does not really exist for 1st Generation FSM Methods. These were designed to measure only 'whole' business applications and changes to them and they provide rules for measuring from the viewpoint of human and peer application users. The need to solve this problem became apparent when using the COSMIC method which was designed to be applicable for business and real-time software at any level of decomposition, for which the users can be any of the three types given above.)

The simple solution defined in the COSMIC Measurement Manual version 3.0 is to introduce the concept of a 'functional user', defined as *"a (type of) user that is a sender or intended recipient of data in the functional user requirements of the software to be measured".*

This concept of the 'functional user' makes a direct link to the 'FU' in 'FUR'. In the vast majority of practical applications of FSM, the measured functional sizes are used for purposes related in some way to the effort to create or to modify the software to which the FUR apply. So if the purpose is to estimate the development effort of some new piece of software, the FUR to be measured will be those that the developer must satisfy.

If, therefore, the purpose were to measure the development effort of the printer/copier application, the FUR to be measured would be those where the functional users are the hardware devices that the application must interact with directly. Measuring a set of FUR that described only the indirect interactions of the application with a human user would give a functional size that would

be too small, which had no relation to the development effort and would thus be unsuitable for estimating.

Sometimes, however, it is of great interest to measure a piece of embedded application software from the viewpoint of the human user as operator, rather than from the viewpoint of the hardware devices. Toivonen, for example, used the COSMIC method to measure the functionality offered by two mobile phones to their human functional users. He then related the functional sizes to the respective memories provided with the phones so as to compare their relative efficiency in their use of storage for the functions provided to human users [4].

Suppliers of benchmarking services and of estimating tools to the business application community have taken it as obvious that the functional sizes they require are those corresponding to the associated project development or enhancement effort, and they usually recommend 1st Generation FSM Methods to measure those sizes. These methods were designed to measure business application software, only from the human user and peer application viewpoints, so the measurement methods are consistent with the related services.

The dangers are clear, however, when suppliers of such services and tools extend their offering to the domain of real-time software. Not only are the 1st Generation FSM Methods stretched beyond their original design goals, but these methods give no guidance on which functional user viewpoint to assume in cases where there may be a choice.

Users of 2nd Generation FSM Methods such as COSMIC that are designed to measure functional sizes in both the business and real-time domains should adopt a standard that any functional size measurement must be accompanied by a definition of the type of functional user from whose viewpoint the measurement was made.

2.3 The 'level of granularity' of the FUR of the software being measured

As the FUR of a piece of software evolve early in the life of a development project, often the need for more and more requirements is discovered and so the *extent* of the software appears to grow. This phenomenon, known as 'scope creep', is well understood, and is NOT what we are talking about here. In this section we assume the *overall* scope of the FUR to be fixed. We will examine how the FUR of a piece of software to be measured are developed *in more detail* as the project progresses.

A 'level of granularity' has been defined in the COSMIC Measurement Manual v3.0 as follows.

"Any level of magnification of the description of a piece of software (e.g. a statement of its requirements, or a description of the structure of the piece of software) such that each increased level of magnification of the description reveals the software's functionality at an increased and comparable level of detail.

As an analogy to illustrate this definition, consider three maps of a nation's road system.

- Map A shows only motorways and main highways
- Map B shows all motorways, main and secondary roads (as in a motorist's atlas),
- Map C shows all roads of all types (as in a set of local district maps).

These maps reveal the details of the national road network at three different levels of granularity, each with their different map scales.

Note that the overall scope of the maps (the nation) is the same in each case. If the purpose is to measure the total size of the national road network, then this can only be accurately measured using Map C.

Similarly with software. Suppose, as an example, we document the FUR of a piece of software, e.g. a simple order-processing system, from the viewpoint of its human functional users as the FUR evolve through four levels of granularity. The structure of the system might be summarised as follows.

1. System Level: The order processing system

2. Sub-system Level: Customer maintenance, product maintenance, order registration, invoicing and goods despatch sub-systems

3. Functional Process Level: (or Elementary Process[6] or Logical Transaction Level)

   For the Customer maintenance sub-system, the functional processes could be
   - Add customer
   - Modify customer
   - Delete customer
   - Enquire on customer

   Similarly for the other sub-systems, each being 'magnified' to expose its functional processes.

4. Functional Process Component Level:

   When a functional process is magnified to the level of granularity at which functional sizes can be accurately measured, the Base Functional Components (or BFC's) revealed depend on the FSM Method. For example, the 'Add customer' functional process measured on the COSMIC method would probably be analysed to consist of four 'data movements' (the BFC-type of the COSMIC method).

   Enter customer (for the entry of data about the new customer)
   Read customer (to check that the entered data is not for a customer that already exists)
   Write customer (to move the validated customer data to persistent storage)
   Exit messages (for error messages, or to indicate successful completion)

   So the functional size of this functional process is 4 CFP (COSMIC Function Points)

All of the above should be familiar and obvious to any experienced user of any of the main 1st or 2nd Generation FSM Methods. The critical level of granularity on which these FSM Methods all rely to ensure that their rules for sizing are unambiguous is that of the 'functional process' (or 'elementary process' or 'logical transaction', depending on the method). As stated above, although the three methods have different definitions for this standard component of functionality, they are all striving to define the same concept. If there were no common agreement on the functional processes of a piece of software that must be measured, there would be no hope of achieving repeatable measurements with any FSM Method.

[Functional sizes can of course be measured early in the life of a project, when the FUR have only been defined at a high level of granularity such as, say, the sub-system level. But it seems to be impossible to define any level of granularity higher than that of a functional process in a way that everyone could interpret in the same way. How do you define a 'sub-system' for example in any universally meaningful way?

[6] The IFPUG method also requires the identification of 'Logical Files' at the Elementary Process level which, on further magnification are broken down to the 'Record Type' Level for measurement. These parallel levels of Base Functional Components are ignored for this discussion

Therefore, to measure a functional size of a sub-system, say, of some software being developed, the artefacts of the sub-system must be measured in some local way and a local process must be worked out and calibrated locally to scale those measurements to the level of granularity of functional processes and their components. Only functional sizes measured at, or scaled to, the level of functional processes and their components have any hope of being repeatable and universally understood. The existing COSMIC Measurement Manual, version 2.2 gives examples of methods of scaling sizes measured on software artefacts at high levels of granularity to sizes at the level of functional processes; version 3.0 will give more examples.]

The COSMIC method's definition of a functional process, for example, is as follows:

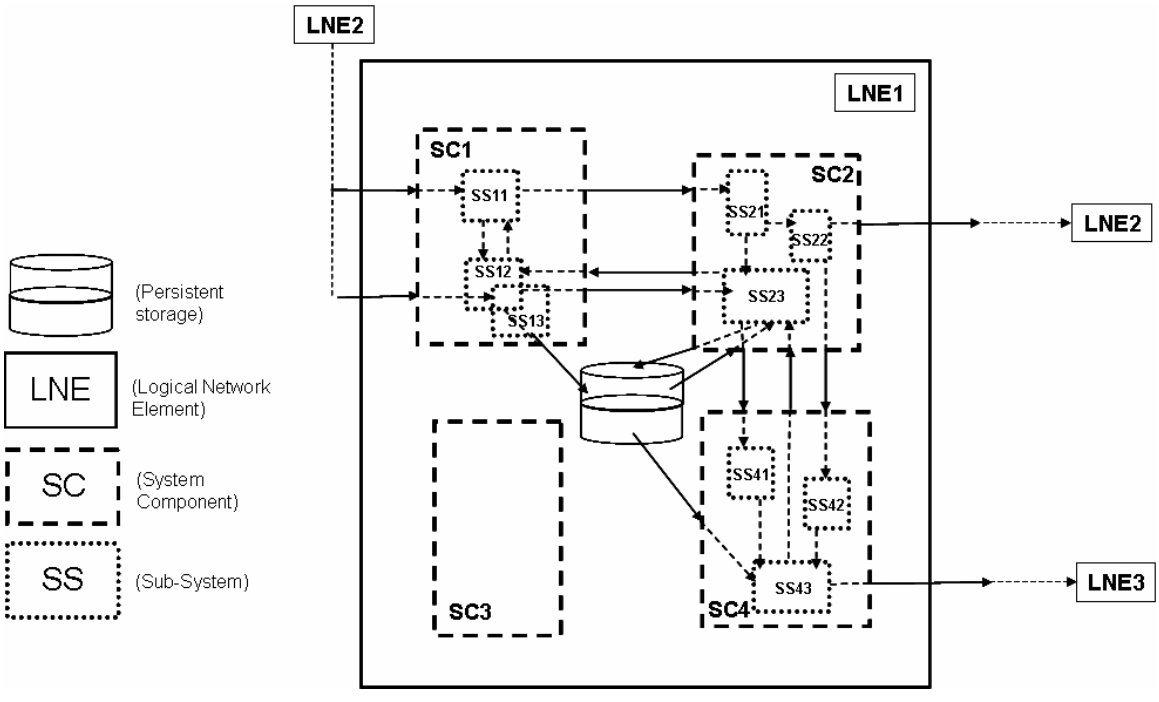
"An elementary component of a set of Functional User Requirements comprising a unique cohesive and independently executable set of data movement types. It is triggered by a movement of data (an Entry) from a functional user that informs the software that the functional user has identified a triggering event. It is complete when it has executed all that is required to be done in response to the triggering event type.

NOTE In addition to informing the software that the event has occurred, the Entry triggered by the event may include data about an object of interest associated with the event,"

A key feature of this definition is that a functional process is triggered by a (single) functional user (-type) detecting a (single) event (-type). A functional user cannot be, for example, a 'department' staffed with humans that handle multiple types of functional processes, or a 'control panel' that has many types of instruments as functional users.

But what if the functional user is a piece of software? We have already seen from 2.1 above that a piece of software can be decomposed in multiple ways, e.g. from a 'system' or an 'application', down to an 'object-class' or to a 'module', depending on the technology. If a functional user can be a piece of software and the latter has no unambiguously definable level of decomposition, it follows that the level of granularity of any associated functional processes is not absolutely definable.

This conclusion is further illustrated by the following example from a major manufacturer of telecoms equipment and associated software. The diagram shows part of the functional user requirements (FUR) of a 'Logical Network Element' (LNE) and the details of the FUR as they are magnified at two lower levels of granularity. The point of interest of the analysis approach of this example is that the LNE is also <u>decomposed</u> at each level of granularity.



The diagram shows, at the highest level of granularity, a single functional process of Logical Network Element 1 (LNE1). As far as this functional process is concerned, LNE1 has two functional users at the same level of granularity, namely LNE2 and LNE3. These users are peer pieces of software. Some data enters LNE1 from LNE2 and some data is sent by LNE1 to LNE2 and to LNE3. Some data is also sent to and retrieved from storage by LNE1.

At one lower level of granularity, the diagram shows that LNE1 is decomposed into four System Components, namely SC1 – SC4. In other words, at the SC level, there is no longer one measurement scope (of LNE1), but four scopes, one for each SC. At this level, the functional users of each System Component are either other System Components in LNE1 or are System Components within LNE2 and LNE3 (the diagram does not illustrate this latter aspect).

The single functional process at the LNE1 level has been decomposed into three functional processes, one in each of the System Components SC1, SC2 and SC4. (We now see that SC3 does not participate in the functional process at the LNE1 level.)

At the lowest level of granularity, we see that each System Component is decomposed into a number of Sub-systems. In total there are now nine measurements scopes within the one LNE. At this level, the functional users of any one Sub-system are either other Sub-systems within LNE1 or are Sub-systems in LNE2 or LNE3 (the latter are not illustrated). The single functional process at the LNE1 level has now been decomposed into nine functional processes at this lowest level of granularity, one in each Sub-system.

At each level of granularity, some data is moved to storage and some is retrieved from storage. The diagram shows which components of LNE1 are involved in this functionality as we decompose to lower levels of granularity.

This diagram therefore illustrates that when a pure software architecture is viewed at different levels of granularity and is also decomposed at each level into its components, such that the functional users are always other pieces of software, functional processes can be defined at *any* level of granularity.

The result of this analysis also shows that the size of the functionality shown in the diagram must increase as the detail of more components and functional processes is revealed at lower levels of granularity/decomposition. This 'growth' is analogous to what we have seen in the example of the road maps. As we move from a large-scale map to one of smaller scale showing more roads, so the size of the road network appears to increase, although the unit of measure for all maps (e.g. the kilometre) is the same.

This finding is extremely important for organizations involved in comparative benchmarking or in the supply of or use of estimating tools. When the functional users of some software to be measured include humans or engineered devices, it is possible to define the level of granularity of functional processes unambiguously because everyone can recognise a single human or a single engineered device. But in a context where the functional users are only pieces of software, there is no one unique level of granularity/decomposition at which functional processes can be defined as a basis for proceeding with measurement.

Within the local context of the telecoms manufacturer that provided this example, it is possible to set an unambiguous local standard for the level at which sizes are measured. For this

manufacturer, separate project teams start to develop software at the Sub-system level; Sub-systems are autonomous applications. It is therefore at this level of granularity/decomposition that the telecoms manufacturer wishes to measure functional sizes for project estimating purposes.

But other organizations might have quite different ideas on what is a 'Sub-system'. So if measurements made in pure software architectures must be compared from different organizations, very great care must be taken to check that the measurements have been made at comparable levels of granularity and of decomposition.

2.4 The Measurement Strategy Process

The four elements of the measurement strategy process should be carefully considered before starting a measurement to ensure that the resulting size can be properly interpreted. The four elements are:

a) establish the purpose of the measurement

b) define the scope of each piece of software to be separately measured, considering all the parameters decribed above

c) establish the viewpoint of the functional users of each piece of software that is to be measured

d) establish the level of granularity of the software artifacts to be measured and how, if necessary, to scale from the sizes of the artefacts to sizes at the level of granularity of functional processes

Some iteration may be needed around steps (b), (c) and (d) when requirements are evolving and new details indicate the need to refine the definition of the scope(s) to be measured.

The great majority of functional size measurements are carried out for a purpose that is related to development effort in some way, e.g. for developer performance measurement, or for project estimating. In these cases, defining the measurement strategy should be very straightforward. The purpose and scope are usually easy to define, the functional users are the users for whom the developer must provide the functionality, and the level of granularity at which the measurements are required is that at which single functional users detect single events.

But not all measurements fit this common pattern, so the measurement strategy parameters must be carefully defined in each case.

3. Summary and Conclusions

To ensure that functional size measurements can be interpreted unambiguously, there is a need for various standards that help define 'which' size of a piece of software has been measured. Such standards should be adopted by suppliers and users of FSM Methods, estimating tools, benchmarking services and such-like.

Standards are needed in three areas:

1. To define standard measurement scopes
   - standard levels of decomposition, e.g. an 'application', a major component of an application, an object-class
   - standard types of delivered work-output (a new development, a set of changes to existing software, some existing software that is re-used, unchanged) and their respective possible size measurements

2. To require that any functional size measurement should be accompanied by a statement of the viewpoint of the functional user (-types) from which the measurement was made

3. To require that all functional size measurements be made, if possible, at the level of granularity at which functional processes and their components can be unambiguously identified. Where this is not possible, i.e. in pure software architectures where all the functional users of the software to be measured are other pieces of software, local standards must be established to ensure the comparability of the levels of granularity of measurements between those sharing data

It is important to remember that these elements of a Measurement Strategy are not tied to the COSMIC FSM Method, but should be common to all FSM Methods. It is only the broader applicability and flexibility of the COSMIC method that has required these elements to be considered more carefully than with 1st Generation FSM Methods.

References

[1] Function Point Counting Practices Manual, release 4.2, The International Function Point Users Group, 2004

[2] MkII Function Point Analysis, Counting Practices Manual, version 1.3.1, The United Kingdom Software Metrics Association, September 1998

[3] The COSMIC Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761:2003), version 2.2, January 2003

[4] Toivonen, H., 'Defining Measures for Memory Efficiency of the Software in Mobile Terminals', International Workshop on Software Measurement, Magdeburg, Germany, October 2002