Early Software Sizing with COSMIC: Experts Guide

2nd Edition

February 27, 2020

(minor editing May 2020)

Acknowledgements

Acknowledgements: Editor & reviewers 2020 (alphabetical order).		
Alain Abran École de Technologie Supérieure Canada	Arlan Lesterhuis COSMIC the Netherlands	Bruce Reynolds Tecolote Research United States
Asma Sellami University of Sfax Tunisia	Hassan Soubra German University of Cairo Egypt	
Sylvie Trudel Université du Québec à Montréal Canada	Francisco Valdés Souto Spingere Mexico	Frank Vogelezang * METRI The Netherlands

* Editor

History of the versions of this document.

DATE	REVIEWER(S)	Modifications / Additions
July 2015	COSMIC Measurement Practices Committee	First public version of this document.
August 2018	Subject Matter Experts	Alignment to version 4.0.2 and issued for review by authors of relevant work.
February 2020	Subject Matter Experts	Revised version based on the 2019-2020 review comments.
May 2020	Subject Matter Experts	Minor editing- No version change for the document.

COSMIC documentation, including translations into other languages, is available at www.cosmic-sizing.org.

You can use the forum on cosmic-sizing.org/forums to post your questions and receive answers from the COSMIC world-wide community. The quality of any answers will depend on the knowledge and experience of the community member that writes the answer.

Commercial organizations exist that can provide training and consultancy or tool support - see www.cosmic-sizing.org for further details.

DOI 10.13140/RG.2.1.4195.0567

Foreword

The COSMIC method provides a standardized way of measuring a functional size of software. In practice, it is sometimes sufficient or necessary to only approximate a functional size:

- early in the life of a project, before the Functional User Requirements (FUR) have been specified down to the level of detail where the precise size measurement is possible;
- when there is insufficient time or resources to measure using the standard method and a quick approximate size will be acceptable;
- when the quality of the documentation of the actual requirements is not good enough for precise measurement.

Purpose of this Guide

The purpose of this Guide is to describe the current state of the art regarding early or rapid COSMIC functional size measurement using approximation techniques. This document describes several approximation techniques with their pros and cons, their recommended area of application and their validity.

The reader is assumed to be familiar with the standard COSMIC method. For those who need to use approximation techniques in practice, see the: Early Software Sizing with COSMIC: Practitioners Guide.

Chapter 1 describes reasons why it may be necessary to approximate functional sizes; how actual requirements are often expressed at varying levels of detail (known as 'levels of documentation') and some general principles on how to recognise and apply ways of sizing approximately such actual requirements.

The rest of the guide is divided into four parts:

- **Part I** Techniques for the Requirements Stage (chapters 2-6)
- **Part II** Techniques for the Feasibility Stage (chapters 7-10)
- **Part III** Techniques in the Research Stage (chapter 11)
- **Part IV** General Concepts

Readers of this Guide who are new to approximate sizing are strongly advised to first read Chapter 1 on the General Principles of approximate sizing and Part IV with the General Concepts.

Table of Contents

1

1. GENERAL PRINCIPLES OF APPROXIMATE SIZING.

1.0 Note on Terminology.

The COSMIC method measures the 'Functional User Requirements' (or FUR) of software. COSMIC uses this term to apply to requirements that are specified in sufficient detail for an approximate COSMIC Functional Size Measurement.

Approximate sizing techniques are designed to be applied when this level of detail is not (yet) available. In this Guide, we therefore refer to the 'actual requirements' as the subject that approximate sizing techniques are designed to measure. The term 'actual requirements' may include 'system' non-functional requirements. But many actual requirements that appear initially as 'system' non-functional evolve, as a project progresses, into 'software' functional requirements that can be sized by the same approximation techniques.

For the definition of general COSMIC terms used in this Guide, see the Measurement Manual [2]. For terms specific to this Guide, see the Glossary.

1.1 When is approximate COSMIC sizing needed?

Early in the software development lifecycle, requirements do not describe the full scope of functionality of the software. Over time, requirements will be detailed, and at times changed, as the software development lifecycle progresses. Early in the lifecycle an assessment of the functional size is needed to support cost or effort estimation.

Whatever the strength of a size approximation technique, there is really no way we can expect that technique to compensate for a lack of understanding of the software job to be done. Until a software specification is fully defined, it actually represents a range of solutions, and a corresponding range of software functional size. This uncertainty diminishes over time when the requirements are more fully understood and by applying proper risk management. It can be graphically represented as a *Cone of Uncertainty*.

The original conceptual basis of the *Cone of Uncertainty* was developed for engineering and construction in the chemical industry by the American Association of Cost Engineers in 1958 [4]. In the software field, the concept was picked up by Barry Boehm in 1981 [5]. Although the effect is usually presented on a logarithmic scale to yield a symmetrical cone, we prefer to show it on a linear scale to demonstrate that the uncertainty towards more functionality is far larger than the uncertainty towards less functionality.

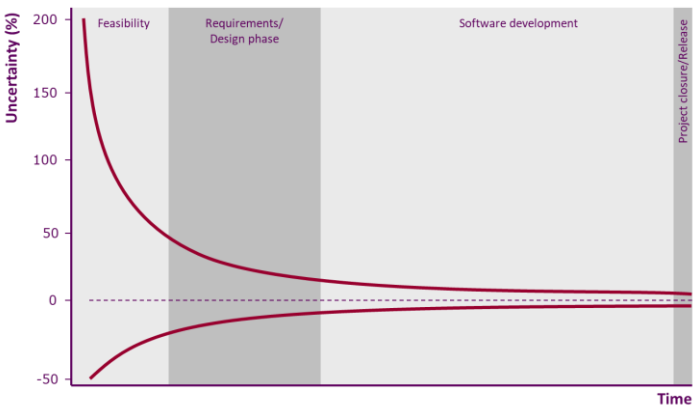

Figure 1.1 Cone of Uncertainty

The use of agile methods of developing software disrupts the smooth curves shown in Figure 1.1 since single projects are replaced by sprints. For each sprint parts of this curve apply. A

common approach in agile software development is to start with the development of an architecture for the new software. Once this is agreed, some priority parts of the architecture may be immediately specified in much greater detail, as sprints comprising a backlog of User Stories are defined. Approximate sizing techniques described in this Guide may be used at the architecture level (see more in Chapter 10) and certainly at the level of outlined User Stories. For more on using COSMIC sizing in Agile environments see [6].

There are three main circumstances in which an approximate[1] COSMIC functional size may be valuable:

- when a size measurement is needed rapidly and an approximate size is acceptable if it can be done much faster than with the standard method. This is known as 'rapid sizing';
- early in the life of a project before the actual requirements have been specified in some detail but insufficient for an accurate size measurement. This is known as 'early sizing';
- in general, when the quality of the documentation of the actual requirements is not good enough for an accurate size measurement.

Rapid sizing can be valuable when a very large piece of software or, say, a whole software portfolio needs to be sized but it would take too much time and money to measure accurately, and approximate sizes are acceptable.

Well before the FUR have been worked out in the detail needed for an accurate measurement, a project effort estimate is required. In such cases the actual requirements would typically exist at various levels of detail in artefacts that are not standardized in any way. For example, some actual requirements may exist at the Use Case level whilst others have been worked out in more detail. Furthermore, Use Cases themselves may express requirements at different levels of detail.

Whether measuring accurately or approximately, measurers should always try to get as much information and details on the description of the actual requirements. Assumptions can then be used to make the functional size measurement as accurately as possible.

1.2 Techniques to approximate functional size

1.2.1 General Principles

Steve McConnell described three levels of determining software size [7]:

- Count: If detailed information is available, the most accurate way of determining the size is to count. Adapting this to the COSMIC method means applying the standard method at the functional process level of documentation and counting the data movements, i.e. following the standard measurement process.

  Even at this level, requirements are very rarely sufficiently detailed that an accurate measurement is possible, so the measurer will need to make certain assumptions.
- Compute: If not enough information is available at the desired level of detail, count something that is available and then compute the answer by using calibration data. Adapting this to the COSMIC method means applying an approximation technique to a higher level of documentation and *scaling* this measurement to the functional process level of documentation.
- Judge: Experts can give an approximation of the size, based on a mental model established on experience. This expert judgment is the least accurate means of approximation. The accuracy can be strengthened if the expert judgment can be tied to

[1] Instead of describing this subject as approaches to 'approximate sizing', it might be more accurate to describe it as techniques to 'estimating sizes'. However, the word 'estimating' is strongly associated with methods of estimating project costs, effort or duration, etc. To avoid confusion, we therefore prefer to write about 'approximate sizing'.

concrete size information. Adapting this to the COSMIC method means *classifying* some available 'objects' (e.g. high-level statements of actual requirements or a list of Use Cases) and assigning a size based on the classification and knowledge of scaling factors established at the 'Compute' level.

1.2.2 Measurement scaling.

Scaling: Count each actual requirement and multiply the count or measurement by a number, the 'scaling factor', to determine its COSMIC functional size. A scaling factor is determined by a calibration process in which, from a representative number of actual requirements, the standard COSMIC functional size has been established.

The general principle of any scaling technique is to find some way of measuring the approximate size of locally-defined artifacts of actual requirements at a high level of documentation and then to measure the same requirements in units of CFP when they are known at the functional process level of documentation. A 'scaling factor' is a ratio that is used to convert measurements on locally-defined high-level artifacts to sizes expressed in CFP - see Table 1.1.

Level of documentation of the Actual Requirements	Sizing	Measurement result
Actual requirements at a high level of documentation derived from e.g.: • high-level statement of actual requirements for the software • architecture artifacts • high-level view of existing software expressed in locally-defined (countable) units e.g. Use Cases	An 'Approximate technique' to the COSMIC measurement method. Calibrated locally	The size of the locally defined unit, expressed in local units or in CFP ↑ scaling ↓ factor
The functional process level of documentation	COSMIC measurement method	Size in CFP

Table 1.1 – Scaling of sizes between different levels of documentation.

Scaling factors should be established locally. For more guidance, see chapter 13.

1.2.3 Localization (calibration).

Approximate sizing techniques are based on artifacts that are not standardized and may vary in their levels of functional details within organizations and across organizations. This implies that the scaling factors need to be calibrated locally. In this document, 'locally' implies that the environment in which the scaling factors for the approximation technique have been defined is representative of the environment the approximation technique is to be used in.

Guidance on localization is given in chapter 13.

1.2.4 Approximate sizing by classification and scaling.

Classification: classify each actual requirement and assign a size to it (i.e. apply a scaling factor) that represents the COSMIC functional size for that requirement.

The general approach of classification is that each part of the actual requirements to be sized approximately is allocated to a pre-defined class (or reference piece) of requirements whose size has been calibrated in CFP, i.e. each class has its own scaling factor. A size is thus assigned to each part of the actual requirements, based on its classification.

It is highly desirable that an approximation technique that uses classification provides objective rules or criteria, or typical examples to assist the correct classification.

Chapter 3 describes a 'Fixed Size' classification technique and in Chapter 4, the classes are 'Equal Size Bands'. The techniques described in Chapters 7, 8, 9 and 10 use various techniques to requirements classification.

1.2.5 Accuracy of approximate sizing.

Any technique to approximate sizing is the result of a trade-off between ease and speed of measurement versus loss of accuracy. Therefore, the accuracy of each technique should be established and reported. See Chapter 13 for guidance on establishing the accuracy.

1.3 Levels of Documentation of Actual Requirements.

In the circumstances in which only an approximate COSMIC functional size may be possible, measurers should be aware of the level of documentation of the software artifacts that are used to approximate the functional size.

A problem for all approximation techniques is that there is no way of unambiguously defining standard levels of documentation higher than the functional process level. A set of higher levels of documentation might be named as, for example, the 'Use Case level', the 'Component' level, the 'Sub-system' level. But these levels can only be properly defined locally, usually with the aid of examples.

Furthermore, research has shown that measurers, especially if inexperienced, often do not realize that actual requirements are expressed at different levels of documentation and/or fail to distinguish the levels. This is one of the commonest problems faced when intending to measure functional sizes, whether accurately or approximately.

Chapter 12 is devoted to the aspects that must be considered when measurements are made at different levels of documentation of actual requirements. Measurers are encouraged to take note of the aspects discussed in that chapter, before applying any of the techniques described in this Guide.

1.4 Quality of Actual Requirements.

Classifying the quality of the various parts of actual requirements by using the scheme of the 'Guideline for assuring the accuracy of measurements' [33] can help determine the accuracy of an approximate size measurement. This quality classification scheme defines five levels for the quality of actual requirements of the functionality, to which we have added a sixth level for this Guide: *Not mentioned (An 'unknown unknown')*.

Table 1.2 shows the six levels and their definitions and which approximation techniques can be applied.

Functional Process Quality Level	Quality of the functional process definition	Approximate sizing techniques that can be used
Completely defined	Functional process and its data movements are completely defined	Use standard COSMIC FSM method
Documented	Functional process is documented but not in sufficient detail to identify the data movements	See Chapters 2 – 7
Identified	Functional process is listed but no details are given of its data movements	See Chapters 2, 5 - 9
Counted	A count of the functional processes is given, but there are no more details**Error! Bookmark not defined.**	See Chapters 2, 5, - 9

Implied (A 'known unknown')	The functional process is implied in the actual requirements but is not explicitly mentioned	See Chapters 2 – 9
Not mentioned (An 'unknown unknown')	Existence of the functional processes is completely unknown at present	Add a contingency for 'scope creep' on the basis of past experience (see 14.2)

Table 1.2 – Functional Process Quality levels related to the approximation techniques.

Given this guidance, we **strongly recommend** that measurers do not use any of the approximation techniques described in this Guide as simple 'recipe books'. Always:

1. examine the actual requirements to be measured closely so that you understand the level(s) of documentation, the completeness and quality of the actual requirements before starting to use an approximation technique;
2. try to obtain more detail than is given in the actual requirements from an expert in the software so that you can at least list and name the functional processes;
3. verify an approximation technique (by comparing accurate and approximate sizing) using local requirements and measurements to ensure it produces reasonably accurate sizes in your local environment and, if necessary, calibrate the technique locally before using it for your own real measurements. See Chapter 13 for more on localization.

1.5 Applicability of approximation techniques described in this Guide.

Most of the practical experience described in this Guide has been obtained from applying the approximation techniques described in Chapters 2, 3 and 4 to measuring the size of the actual requirements for new business application software.

For real-time embedded software: an example in Chapter 4 describes the result of applying an approximation technique successfully to some very complex real-time embedded avionics software. A case in section 12.1 describes the approximate sizing of the functionality of a complex telecoms software architecture at various levels of documentation.

The approximation techniques described here are applicable to the actual requirements:

- for new software and for enhancements to existing software that require whole new additions of functionality
- for software from any domain, e.g. business, real-time embedded or infrastructure.

We are not aware of any reported experience of applying these approximation techniques to size the actual requirements for enhancements that involve many changes (adds, modifies and deletes) to existing software. However, these techniques may be applied to size such enhancements, provided great care is taken for the calibration process. See Chapter 123for more.

For practical use we have divided the approximation techniques into three parts:

Part I Techniques that can be used in the requirements stage, when the requirements no longer contain any 'unknown unknowns'.

Part II Techniques that can be used in the feasibility stage, when the completeness of the requirements is an aspect that should be taken into account.

Part III Techniques that are still in the research phase. These techniques have no publicly reported practical track record as far as we are aware at the time of writing.

In **Part IV** a number of general aspects worked out in more detail for reference.

Part I

Techniques for the Requirements Stage.

2

2. AVERAGE SIZE OF FUNCTIONAL PROCESSES.

Origin and approximation mechanism.

The average functional process approximation was first introduced in version 2.2 of the COSMIC method [9]. This is the simplest process for obtaining an approximate size of a piece of software. It may be used when the actual requirements of a piece of software are known only to the level of functional processes but not to the level of data movements.

A Determine the scaling factor.

1. Identify a sample of actual requirements whose functional processes and data movements have been defined in detail, with characteristics similar to the actual requirements of the software to be measured.
2. Identify the functional processes of these sample requirements.
3. Measure the sizes of the functional processes of these sample requirements accurately using the standard COSMIC method.
4. Determine the average size, in CFP, of the functional processes of these sampled requirements (e.g. average size = 8 CFP). '8' is then the scaling factor for this technique.
5. Identify the standard deviation.

B Approximation using the scaling factor.

1. Identify and count all the functional processes of the actual requirements of the software to be measured (e.g. = 40 functional processes).
2. *For a set of requirements:* The approximate functional size of the set of actual requirements of the software to be sized is approximated to be (number of functional processes x scaling factor) = 40 x 8 CFP = 320 CFP [1].
3. *For a specific requirement:* determine the approximate size range by using the average size +/- 1 standard deviation.

From above, if the average is 8 CFP, and the standard deviation is 2 CFP:

- the range of a specific functional process is: [6 to10 CFP];
- the range for the set of 40 functional processes is: [240 to 400 CFP]

Applicability and reported use.

In organizations that have established a COSMIC measurement practice this technique is used to produce a first ball-park approximation of the size.

In 2005 Vogelezang reported [10] that in different industry sectors different sizes were measured for an average functional process, i.e. for the scaling factor. This supports our recommendation that the use of this approximation technique should always be calibrated locally.

Strengths and weaknesses.

Strength: Easy to use.

Weaknesses:

- The average functional process size is (assumed to be) domain dependent.
- It requires sampling and calculation of an average functional process, based on detailed measurements from within the same localization (see chapter 13). This data may not (yet) be available.

Recommended area of application.

This approximation is valid as long as there is sufficient reason to assume that the sample used to calculate the size of the average functional process is representative for the software of which the functional size is approximated. See also chapter 13.

To this end, it is good practice to remove from the dataset the applications that are considered dissimilar from the one being estimated [13]. Also note that this technique works best with a symmetrical dataset and a standard deviation (σ-value) that is significantly smaller than the average functional process size.

Research developments.

In 2009 Van Heeringen *et al*., carried out measurements to compare the accuracy of the average functional process approximation with measurements using the standard COSMIC method. In [11] they compared approximations of 24 pieces of software from different organizations against the accurate measurements.

In 2013 De Marco *et al*., reported good results with this technique to estimate the development effort for web application development [12].

In 2014 Del Bianco *et al*., did an experimental evaluation of this technique and did not find a good predictive power of this approximation [13]. The predictive power could be improved by calculating the ordinary least square formula of the sample and using that for estimation. They also proposed a similar method, based on the number of involved data groups, rather than the amount of data movements. See section 10.2 for details on that method.

In 2019 Lavazza and Morasca concluded that this technique generally provides approximations that are reasonable for early and quick sizing, but in some cases its estimation errors are too large to be acceptable [14].

Practical use in enhancement projects.

Often projects must not only create new functional processes, but must also modify existing functional processes. In practice the following technique has been observed:

1. While measuring projects, each functional process is marked as either 'New' or 'Modified'.
2. The average size for a new & a modified functional process were established separately. The average values obtained varied depending on the domain. Typically, the average size of a modified functional process was about half the average size of a new one.

When approximating the size of a project at an early stage, the new and modified functional processes must be identified, counted and multiplied by their respective average sizes.

3

3. FIXED SIZE CLASSIFICATION.

Origin and approximation mechanism.

The fixed size classification approximation was first introduced in the 'Advanced and Related Topics' document in version 3.0 of the COSMIC method [15].

The technique depends on defining a typical size classification of the functional processes in the piece of software to be measured. A corresponding size, or scaling factor, is then assigned to each class, for all of its functional processes.

A statement of actual requirements must be analysed to identify the functional processes and to classify each of them according to their size in one of three or more size classes called, for instance, Small, Medium and Large. Table 3.1 shows an example of a set of size classes that is in actual use in a specific business organization. The rows show the three possible size classes for this organization and the total number of CFP that must be assigned to a functional process in each group (for instance, if one small functional process is identified, it is assigned a scaling factor of 5, so that its size is 5 CFP). To force the measurer to make a deliberate choice of size, the step size between the classes is taken to be fairly wide, at 5 CFP.

The four columns #E, #X, #R and #W explain why the functional process of a given size is assigned the number of CFP. For instance, a Small functional process is assumed to consist of 1 Entry, 1 Read, 1 Write and 1 Exit data movements. For a Medium or Large functional processes more data movements of each type are assumed. The fifth column 'Error messages' adds in one Exit for error/confirmation messages.

Classification	Size (CFP)	#E	#X	#R	#W	Error messages
Small	5	1	1	1	1	1
Medium	10	2	2	3	2	1
Large	15	3	3	4	4	1
…						

Table 3.1 – Fixed size classification from [8].

If the functional size of some actual requirements must be approximated early in the development process, each actual requirement is assigned one or more functional processes, together with their appropriate size classification and corresponding size approximation. Use of a table such as 3.1 helps the measurer to make a quicker decision on the assignment of the size class for each functional process. If necessary, the table may be extended to accommodate one or more additional sizes, such as 'very large' of 20 CFP. When well calibrated, this technique should give a more accurate functional size than the average size technique of Chapter 2.

Applicability and reported use.

This technique has been used extensively by a large business organization in the Netherlands. The technique was successful within that organization. There is no public information on the use and accuracy of this approximation other than in this organization.

Strengths and weaknesses.

Strengths:

- Easy to use.
- Can be implemented in a simple way.
- The scale factors are documented, i.e. verifiable.
- As the approximate sizes are based on an expected number of objects of interest to be accessed (hence the data movements), knowledge of this factor helps the measurer to decide which classification to assign to a functional process.

Weaknesses:

- The definition of the size classification is (assumed to be) domain dependent.
- Assigning functional processes to a size class is a subjective element of this approximation technique, which reduces the strength of the approximation. See also Chapter 1.

Recommended area of application.

This approximation is valid as long as there is sufficient reason to assume that the assigned size classification is representative for the software of which the approximate functional size is to be measured. See also chapter 13. It is highly desirable that objective local rules are determined to assist measurers in assigning the correct classification.

Research developments.

In 2000 Santillo investigated the classification of Functional Processes in order of increasing magnitude, as Functional Process, General Process, or Macro Process. In this technique also a subdivision in Small, Medium and Large was used, but on unknown intervals, instead of known intervals as in this technique. This has led to the Early & Quick technique see - chapter 8 [31].

In 2019 Lavazza and Morasca tested two alternatives for the Fixed Size Classification technique [14]: the Equal Number Bands technique (see section 11.5) and the Equal Range Bands technique (see section 11.6).

In the Equal Number Bands technique the functional processes from a reference set are ordered and divided into a number of bands with an equal number of functional processes. The average functional size of the functional processes that are assigned to that band is used as the estimation value for that band (see section 11.5 for details).

In the Equal Range Band technique the range between the smallest and the largest functional process in the reference set is divided into a number of ranges that are defined so that all ranges have equal width. The average functional size of the upper and lower functional size of each band is used as the estimation value for that band (see section 11.6 for details).

Lavazza and Morasca determine that these techniques could lead to a better prediction of the actual size than the Fixed Size Classification technique. In general, they concluded, bands-based methods provide much more accurate estimates than the Average Functional Process technique, when sufficiently skilled classifiers are employed, and the proper number of bands is used.

4. EQUAL SIZE BANDS.

Origin and approximation mechanism.

The equal size bands approximation was first introduced in version 2.2 of the COSMIC method [9].

In the 'Equal Size Bands' technique, the functional processes are classified into a small number of size bands. The boundaries of the bands are chosen in the calibration process so that the total size of all the functional processes in each band is the same for each band.

A Determine the scaling factors.

1. Identify a sample of actual requirements whose functional processes and data movements have been defined in detail, with characteristics similar to the actual requirements of the software to be measured.
2. Identify the functional processes of these sample requirements.
3. Measure the sizes of the functional processes of these sample requirements accurately using the standard COSMIC method.
4. Sort the functional processes in ascending order and present them graphically in ascending order together with their cumulative size.
5. Using the information from the cumulative distribution, split the sample into a number of bands that all have the same total size (and thus contain a different number of functional processes). So if, for example, the choice is to have three bands, then the total size of all the functional processes in each band will contribute 33% to the total size of the software being measured.
6. Determine the average size, in CFP, of the functional processes in each band. These are the scaling factors for each band. These scaling factors are usually not integer numbers.

B Approximation using the scaling factors.

1. Identify for each of the functional processes to be approximated the size band in which it belongs.
2. Assign the scaling factor for that size band to the functional process.
3. Sum all the approximated functional processes to get the functional size approximation of the whole piece of software.

Applicability and reported use.

Vogelezang and Prins reported on a calibration using measurements on 37 business application developments, each of total size greater than 100 CFP [16]. They used four size bands to make a distinction between relatively small processes, medium-sized processes, large and very large ones. The average sizes of each band of the 2,427 functional processes of the 37 applications were distributed over the four bands were:

Band	Average size of a Functional Process	% of total Functional Size	% of total number of Functional Processes
Small	4.8	25%	40%
Medium	7.7	25%	26%
Large	10.7	25%	19%
Very Large	16.4	25%	15%

Table 4.1 – Equal size bands from 37 business applications [16].

This same approach was used to calibrate one component of a major real-time avionics system (of total size 10,875 CFP), with the following results:

Band	Average size of a Functional Process	% of total Functional Size	% of total number of Functional Processes
Small	5.5	25%	49%
Medium	10.8	25%	26%
Large	18.1	25%	16%
Very Large	38.8	25%	7%

Table 4.2 – Equal size bands from a major component of an avionics system.

Note the similarity between the *numbers* of functional processes in each of the four bands despite the totally different types of software. However, the average size of the functional processes in each band is quite different, especially for the large and very large bands. This emphasizes the need for local size calibration.

To size a new piece of software the functional processes of the new piece are identified, they are classified as 'Small', 'Medium', 'Large' or 'Very Large'. In the next step, the average sizes of each band (such as listed above but preferably calibrated locally) are then used to multiply the number of functional processes of the new piece of software, in each band respectively to get the total estimated approximate size.

The advantage of this technique is that at the end of a new approximated sizing for some new software, the measurer can check if the contribution to the total size of the functional processes of the new software in each size band is close to the 25% assumed by this 'Equal Size Band' technique. If so, the calibration will have been suitable for this new measurement. If not, the measurer should consider whether the calibration has been accurate enough.

The accuracy of any calibration of classification sizes for this technique is important for accurate sizing since functional processes typically exhibit a skewed size distribution, as illustrated by both sets of data given above. In other words, software systems typically have many functional processes of small size and few of larger sizes. More attention must therefore be paid to accurate sizing of the few 'large' and the even fewer 'very large' functional processes to get an accurate total size.

Strengths and weaknesses.

Strengths:

- Easy to use.
- The technique has been shown to be applicable for software in both the business application and real-time embedded domains.
- The use of more bands potentially leads to a more accurate approximation.

Weaknesses:

- Choose the number of bands carefully that the bands are significantly far enough apart to be different bands of functional processes.

- More bands lead to a higher probability that a given functional process is classified incorrectly. Therefore, only as many bands should be used that can be correctly identified by a measurer.
- A considerable number of accurately measured functional processes must be available before the bands can be safely determined.
- When carrying out an approximate size measurement, assigning functional processes to a size class is a subjective element.
- When there are a few functional processes in the Very Large band, the average size of this band must be used with great care, because the actual functional size of an approximated functional process can differ significantly from the scaling factor for this band.

Recommended area of application.

This technique is recommended for approximate sizing of software that has a significantly skewed distribution of the size of functional processes.

This approximation is valid as long as there is sufficient reason to assume that the assigned size classification is representative for the software of which the approximate functional size is to be measured. See also chapter 13.

It is highly desirable that objective local rules are determined to assist measurers in assigning the correct classification.

The conclusion we can draw from these results is that the greater the skew, the greater the advantage of this technique for accuracy over the techniques in chapters 2 and 3.

Research developments.

In 2009 Van Heeringen *et al*., compared approximations of 24 pieces of software from different organizations with the accurate measurements [11]. The results were that on average the difference between the approximated size and the size that was determined with the standard FSM procedure was only 1.26%. This means that for 90% of all results in the study, the Equal Size Bands technique gave a result within -15% to +25% of the corresponding result found with the standard COSMIC method.

In 2012 the results of the study done by Vogelezang and Prins in 2005 [16] were tested using a fuzzy logic model to approximate the functional size of the C-registration system Case Study by Valdes Souto and Abran [20]. This test showed that the equal size bands approximation was a better approximation technique than the EPCU fuzzy logic model used in the experiment when the FP are known and fully documented as in the C-registration Case Study. See section 8.2 for a description of the EPCU model.

In 2013, in his PhD thesis on the development of a scaling factors framework to improve the approximation of software functional size, Almakadmeh asserted that a solid approximation framework could be designed by combining the equal size bands approximation with the quality rating mechanism from the COSMIC Guideline for Assuring the Accuracy of Measurement [8].

In 2016, the problem with the distribution of functional processes' sizes in the historical dataset and in the new application to be sized was analyzed by Luigi Lavazza and Sandro Morasca. They demonstrated that if the distribution of functional processes' sizes is approximately the same in the historical dataset used to calibrate the model and in the new application to be sized then the accuracy of the Equal Size Bands method is similar to the Average Functional Process technique. This result shows that there is a risk that after applying the Equal Size

Bands technique (which involves the cost of classifying each functional process of the new application) one may discover that the classification work was useless, since the Average Functional Process method would have given the same result faster and with less effort. On the contrary, when the distributions are different, the Equal Size Bands technique provides definitely more accurate estimates.

In 2019 Lavazza and Morasca published an empirical evaluation of early and quick size estimating techniques [14]. In general, they concluded, bands-based methods provide much more accurate estimates than the Average Functional Process technique, when sufficiently skilled classifiers are employed, and the proper number of bands is used.

5

5. AVERAGE OF USE CASES.

Origin and approximation mechanism.

The average Use Case approximation was first introduced in the 'Advanced and Related Topics' document in version 3.0 of the COSMIC method [15].

The principle of the approximation is similar to the average functional process approximation from chapter 2, but on a higher level of documentation, namely the Use Case.

Local calibration might determine that a (locally-defined) Use Case comprises, on average, 3.5 functional processes, each of average size 8 CFP (as in the example in Chapter 2). Hence the average size of a Use Case according to this local definition, is 3.5 x 8 = 28 CFP per Use Case.

For a new project with 12 Use Cases, the software size would be 12 x 28 = 236 CFP.

Thus, with this calibration, identifying the number of Use Cases early in a development project's life will provide a basis for making a preliminary estimate of software size in units of CFP. The uncertainty on this approximate size will be greater than that with the techniques discussed in e.g. Chapter 2. This is because the scale factor 28 is the product of two other scale factors (8 and 3.5) which are themselves estimated. (The result might therefore be better expressed as, for example, 240 plus or minus x%, where the 'x%' has been obtained by appropriate analysis).

Applicability and reported use.

In literature there is no reported use of this approximation technique.

Strengths and weaknesses.

Strengths:

- Easy to use if there is a local standard on what is a Use Case, more specifically describing the expected level of documentation of a Use Case.

Weaknesses:

- The problem with this technique is that the concept of Use Case is interpreted in different ways by different organizations and people, so that the amount of functionality that is associated to a Use Case can vary widely [13]. There is evidence that the technique would not work unless the organization producing Use Cases adopts some sort of standard to ensure consistency in their size.

- The scaling factor is the product of two other scaling factors which are themselves estimated. This increases the uncertainty of the approximation result.

- Sufficient historical data are required for localization of the size of an average Use Case. Since Use Cases are not standardized, it is of vital importance to verify the homogeneity of the historical data as well.

Recommended area of application.

This approximation is valid as long as there is sufficient reason to assume that the assigned size classification of an average Use Case is representative for the software of which the functional size is approximated.

Also note that this technique works best with a symmetrical dataset and a standard deviation (σ-value) that is significantly smaller than the average use case size.

Research developments.

In 2013 a study by Gencel and Symons [23] of practices in a very large software house showed that different parts of the software house had quite different ideas on what a Use Case is. In one part of the software house there was a fairly consistent ratio of functional processes per Use Case. In another part, this ratio varied widely. This finding should be taken into account in the local calibration when this approximation technique is used.

In 2018 Ecar *et al*., proposed the COSMIC User Story Standard to help overcome the issue of differing ideas on what a User Story is. They introduced and tested a User Story template that facilitates measurement of the User Stories and describing them on a similar level of abstraction [24].

6. FUNCTIONAL SIZE MEASUREMENT PATTERNS.

Origin and approximation mechanism.

Functional Size Measurement (FSM) patterns [25] were proposed in 2016 as a means to help inexperienced measurers learn faster how to apply the COSMIC method by establishing the relationship between the method rules and the measurement results.

COSMIC experts have observed that some patterns of measurement results recur repeatedly. A formal definition presented in [25] is:

DEFINITION – FSM Pattern
A predefined generic software model solving a recurring measurement problem in a specific context.

PRINCIPLE – FSM Pattern
Functional User Requirements in a given software domain often follow patterns. Hence their related measure of functional size also follows a pattern.

Four types of patterns have been defined: “Micro FSM pattern”, “Basic FSM pattern”, “Composite FSM pattern”, and “Multi-composite FSM pattern” respectively. See figure 6.1.

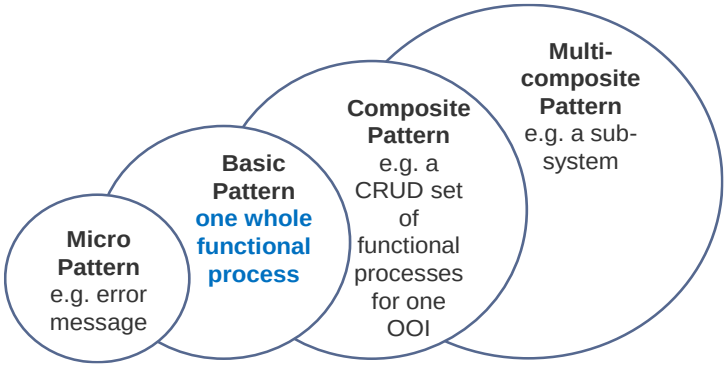

Figure 6.1 – The four types of measurement patterns.

Micro FSM patterns: a fragment of a functional process, involving one or several data groups. Example: displaying an error message.

Note that fragments have no independent occurrence in COSMIC detailed measurements, except during measuring changes.

Basic FSM patterns: a complete single COSMIC functional process.

Composite FSM pattern: a set of basic FSM patterns having a high level functional meaning together. A composite FSM pattern combines several functional processes. The CRUDL

(Create, Retrieve, Update, Delete, List) set of functional processes to maintain data describing one or more related objects of interest is an example of a composite FSM pattern.

Multi-composite FSM pattern: a set of composite and basic patterns having functional relationships among them. A multi-composite FSM pattern combines multiple functional processes handling data describing several objects of interest within the software being measured. In business application software, a multi-composite FSM pattern could represent a whole module, or component of a distributed application or even a whole application. In real-time embedded systems, it could be the set of back-end subsystem functionalities for a family of devices.

Applicability and reported use.

Each pattern type is described with three characteristics:

- **Problem:** Describe succinctly the problem the pattern is about to solve;
- **Context:** Describe the circumstance when that problem occurs;
- **Solution:** Describe how to solve the problem in this specific context.

It is also necessary to add a pattern name.

A) Example with Micro FSM patterns:

Pattern Name: Display simple error messages.

Problem: How to approximate (or measure) a FUR that displays one or several occurrences of error messages?

Context: For one or several validations within a functional process that outputs to a human functional user.

Solution: For Display simple error messages.

Functional Process	Data Group	Data Movements	Functional Size (in CFP)
<Functional process>	Error message	X	1
		Total:	**1**

a. Legend: E=Entry; X=eXit; R=Read; W=Write;

B) Example with Basic FSM Pattern:

Pattern Name: Basic Create Functional Process.

Problem: How to measure a FUR to handle the response to a single event that involves the persistence of a data group with a simple data existence validation?

Context: It is common in Information Systems that data needs to be saved for later use. The FUR may generically state that "the [functional user] enters data about the <object of interest> (1 Entry) and saves the occurrence; the [software] ensures that this occurrence does not already exist (1 Read) and makes the occurrence persistent (1 Write); If the occurrence can't be persisted or there are other validation failures, an error message is displayed (1 eXit)".

Solution: For Basic Create

Functional Process	Object of Interest	Data Movements	Functional Size (in CFP)	Remark
Create <object of interest>	<object of interest>	**E**RW	3	Creates a new occurrence
	Error message	X	1	
		Total:	**4**	

a. Legend: E=Entry; X=eXit; R=Read; W=Write;

b. The "**E**" in bold represents the triggering Entry of its functional process.

C) Example with FSM Pattern:

Pattern Name: Composite CRUDL-3OOI

Problem: How to measure a group of FUR (e.g. CRUDL) related to a given Object of Interest having the same functional processes as another similar set of FUR, but handling also two other Objects of Interest?

Context: It is common in Information Systems that FUR are repeated many times with variations of Objects of Interest while the expected software behaviour is the same. The FUR, at a higher level of decomposition than preceding examples, may generically state that "the [systems] is required to Create, Retrieve, Update, Delete, and List occurrences of <First OOI>. A <First OOI> must be linked with an existing <Second OOI> [for attribute X] and to an existing <Third OOI> [for attribute Y], chosen from a list on screen.

Solution:

Functional Process	Object of Interest	Data Movements	Functional Size (in CFP)	Remark
Create <First OOI>	<First OOI>	**E**RW	3	Create new occurrence
	<Second OOI>	RX	2	Read and display list
	<Third OOI>	RX	2	Read and display list
	Error message	X	1	**Subtotal: 8 CFP**
Retrieve <First OOI>	<First OOI>	**E**RX	3	Select, read and display existing occurrence
	<Second OOI>	RX	2	Must read its ID to display its name
	<Third OOI>	RX	2	Same as above
	Error message	X	1	**Subtotal: 8 CFP**
Update <First OOI>	<First OOI>	**E**RW	3	Update existing occurrence
	<Second OOI>	RX	2	Read and display list
	<Third OOI>	RX	2	Read and display list
	Error message	X	1	**Subtotal: 8 CFP**
Delete a <First OOI>	<First OOI>	**E**RW	3	Delete an occurrence, Read it first, no other OOI required
	Message	X	1	**Subtotal: 4 CFP**
List <First OOI>	<First OOI>	RX	2	Read and display list
	Filter	**E**	1	Search filter applicable to all OOIs
	<Second OOI>	RX	2	Read/display list (filter)
	<Third OOI>	RX	2	Same as above
	Error message	X	1	**Subtotal: 8 CFP**
		Total:	**36**	For this FSM pattern

[c.] Legend: E=Entry; X=eXit; R=Read; W=Write; The "**E**" in bold represents the triggering Entry of its functional process.

The equivalent of a CRUDL pattern in real-time embedded software is to establish a pattern for the set of changes in the state of the external functional users/objects of interest of the software; each such change results in an event that triggers a distinct functional process in the software.

D) Example with Multi-Composite FSM Pattern.

Pattern Name: Multi-composite module-3OOI.

Problem: How to measure a whole module related to two primary Objects of Interest having the same functional processes as another similar module, but handling also one secondary master Object of Interest, three reference Objects of Interest, and four transactional Objects of Interest?

Context: It is common in Information Systems that popular modules are repeated many times with variations of Objects of Interest while the expected software behaviour is basically the same. The FUR, at a module level of decomposition, may generically state that "the [systems] is required to "Manage" occurrences of <OOI1> and <OOI2> ("Manage" being equivalent to CRUDL in this case), and also "Manage" 2nd, 3rd, and 4th reference Objects of Interest, and so on until all expected functionalities for that module have been described.

Solution[2]:

FSM Pattern	Category	Functional Size (in CFP)	Example
CRUDL-3OOI	Composite	36	Manage "Customer"
CRUDL-1OOI	Composite	20	Manage "Sales Representative"
CRUDL-1OOI	Composite	20	Manage "Customer category"
CRUD-2OOI	Composite	22	Manage "Account aging parameters"
CRUD-3OOI	Composite	26	Manage "Invoicing parameters"
CRUD-3OOI	Composite	26	Manage "Cash receipt (C/R) parameters"
Transaction-7OOI	Basic	12	Enter manual invoices
Transaction-6OOI	Basic	10	Enter a manual cash receipt
Transaction-8OOI	Basic	14	Enter adjustment on Invoice or C/R
Report-3OOI	Basic	7	Report on customer sales
Report-4OOI	Basic	9	Customer aging report
Report-5OOI	Basic	11	Customer statement of account
Milestone-2OOI	Basic	10	End of month A/R processing
	Total:	**223**	**For this FSM pattern**

Strengths and weaknesses.

Strengths:

- Contributes to reduce measurement effort.
- Could be applied by relatively inexperienced users of the COSMIC method.
- Provides a more accurate size measurement by helping to avoid common measurement mistakes.
- This set of patterns, for any 'CRUDL' member is more detailed by giving the sizes of 'simple', 'average' and 'complex' functional processes Further, 'Reports' have been added to the CRUDL set. An early version of such a set of patterns is given in [27]. This experience suggests that the early estimation needs within any one organization can be largely satisfied by developing standard patterns only at the Basic and Composite Levels. If there is a need to estimate for a Multi-Composite set of FUR, a size can be built up from the lower levels by local knowledge and expert judgement.
- Using well-defined patterns adapted to an organization's local software requirements should enable improved repeatability of early size estimation.

Weaknesses:

- FSM patterns and their usage have not yet been quantitatively evaluated against the solution objectives for COSMIC FSM. More case studies and research are needed.
- FSM Patterns need to be described very well in order to be used by inexperienced measurers.

[2] Please note that is only a summary view for such module type. The full description is available in [25].

Recommend area of application.

This technique is suited for the following types of software: business and real time software within well-defined domains.

Research Developments.

In 2017 experiments with this technique have started at the Polish Agency for Restructuring and Modernisation of Agriculture.

Research and development is continuing on COSMIC support tools and pattern definitions in various contexts. The FSM Patterns need to be described very well in order to be used by inexperienced measurers.

Part II

Techniques for the Feasibility Stage.

7

7. SOFTWARE ICEBERG ANALOGY.

Origins and approximation mechanism.

This approximation technique described in Abran and Vedadi [42] is based on:

A. The iceberg analogy
B. ISO-IEEE standard 29148 on Requirements Engineering.

In the iceberg-software analogy illustrated in Figure 7.1, the left above-water line visible part of the iceberg corresponds to the very early-on description of the software requirements, which requirements are progressively described with more details, up to the final fully described details of these software requirements (the progressive visibility of the under-waterline of the iceberg).

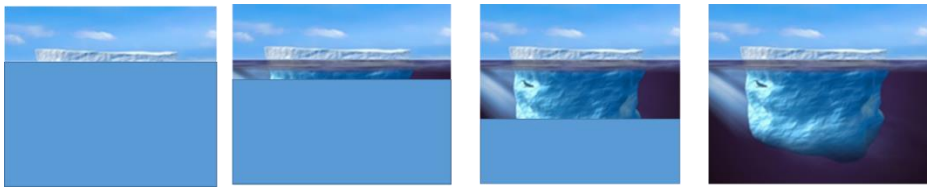

Fig. 7.1 Software-Iceberg analogy: from initially visible functions (left) to full view (right).

In physics there is a well-known ratio for the mass above to the mass under water for a floating iceberg – of course such a ratio was calculated based on a number of empirical measurements and scientific observations, and it is a constant.

In software development there is no known constant ratio, but using empirical observations and measurements from COSMIC case studies, some approximation procedures can be worked out to come up with ratios for local contexts.

Across all software projects, functional visibility will vary across the development lifecycle, and hence software functional documentation across lifecycle phases will vary.

The ISO-IEEE standard 29148 on requirements engineering presents a number of concepts related to the sources, types and levels of detail of the requirements throughout the system and software life cycle.

The initial set of requirements originates from two sets of sources, the business stakeholders and other stakeholders, which leads to the 'systems' requirements. From the system functional requirements, some will be allocated to software requirements (as well as to hardware requirements and at times to manual operational procedures). These sources provide the system contextual requirements, including the system purpose, system scope and system overview. From this contextual information, the following are then identified:

- system functional requirements (some of which will be allocated to software),
- system non-functional quality (some of which will be allocated to software).

ISO-IEEE 29148 also notes that in addition to software functions explicitly identified, there may be interfaces identified, but not yet specified, as well as quality requirements, still at a high level.

Applicability and reported use.

In Abran and Vedadi [42] these concepts were applied to two COSMIC case studies:

1. Course Registration System CRS
2. RestoSys

Within these COSMIC case studies:

A) Three levels of documentation were identified:

- Level 1: (Business) functions (list of 'system' functions).
- Level 2: (Business) functions allocated to software functional processes (list of 'software functions').
- Level 3: Detailed functionality allocated to each software functional process (functional details allocated to software).

B) Functional classifications.

Each of the functional details within each functional process of the case study was classified into the following five categories (with corresponding level of documentation within the case studies):

a. Functionality from business requirements– allocated to software functions - level 2,
b. Functionality with more details from business requirements - level 3,
c. Operational functionality for implementing in practice the business requirements functionality - level 3,
d. Functionality derived from system requirements & allocated to software - level 3,
e. Functionality related to an interface to other software applications - level 1 or 2.

Figure 7.2 presents the distribution of COSMIC-sized functionality in the CRS case study.

Fig. 7.2 CRS case study - Scaling factors based of the size of functional processes [42].

The iceberg-like ratios can then be used as scaling factors at various phases of the lifecycle and levels of documentation.

For instance, if COSMIC measurement is done very early on in the life cycle where only systems functions are identified and measurable with COSMIC, the 20% ratio of functionality can be used as a scaling factor in the following way:

If the systems functions to be allocated to software are measured for a size of 30 CFP at that point in time and with the functional documentation available at that specific point it time, then

it could correspond to only 20% of the expected final functional size: this then mean that this description of functionality is at a 1:5 scale:

- e.g. where 1CFP measured at the beginning of the life cycle could represent 5 CFP at the end of the life cycle.

Therefore a 30CFP at a scale of 1:5 would then be reasonably be expected to grow to 150 CFP once the software fully developed.

Recommended area of application.

In the very earliest stages of new software development, such as at the system level in ISO-IEEEE 29148 before a project is formally defined, the requirements will probably be known only in the broadest outline. At this stage it may be possible to determine an approximate size using the iceberg analogy with the known sizes of other existing software (such as the CRS or RestoSys case studies), but it will be too early to apply any of the approximate sizing techniques described in this Guide.

In [42] the presented scaling factors are specific to the case studies used, but the iceberg approximation technique can be used in most organizations, provided that data can collected on past projects and identify classifications of functionalities and levels of documentation that are relevant to the context for approximation.

8

8. EARLY & QUICK COSMIC APPROXIMATION.

Origin and approximation mechanism.

The Early & Quick COSMIC approximation technique is an adaptation of the Early & Quick Function Points technique [28]. Later, the Early & Quick technique has extended its applicability domain to the COSMIC measurement method [31].This was established by taking advantage of enhancement opportunities derived from local or global measurement data sets, like the ISBSG benchmarking data base and others [10][16].

The Early & Quick COSMIC approximation technique combines scaling and classification techniques [30]. It permits the use of different levels of documentation for different branches of the system on different levels of decomposition. The overall size approximation (which is a 3-point estimate of a minimum, most likely, and maximum size) is the sum of the individual components' size approximations.

The Early & Quick COSMIC approximation technique is based on the capability of the measurer to classify a part of the actual requirements as belonging to a particular functional category. An appropriate reference table then allows the measurer to assign a CFP average value for that item (this is applied for software in each identified layer of the software architecture separately, as per the standard COSMIC method). Each function can be categorized, in order of increasing magnitude and decreasing number of composing elements, as a Functional Process, Typical Process, General Process, or Macro-Process.

a) In the Early & Quick technique a Functional Process[3] (FP) is the smallest process. A Functional Process can be Small, Medium, Large or Extra Large, depending on its estimated number of data movements. This categorization is similar to the 'Fixed Size Classification approximation' technique discussed in chapter 3.

b) A Typical Process (TP) is a set of the four basic user operations: Create, Retrieve, Update and Delete (CRUD) on data describing a particular object of interest. These Typical Processes are frequently found in business application software.

c) A General Process (GP) is a set of medium Functional Processes and may be thought as an operational sub-system of the application. A GP can be Small, Medium or Large, based on the estimated number of Functional Processes that it contains

d) A Macro-Process (MP) is a set of medium General Processes and may be thought as a relevant sub-system of the overall Information System of the user's organisation. A MP can be Small, Medium or Large, based on the estimated number of General Processes that it contains.

Each level is built up on the basis of the lower one. An appropriate reference table then allows the measurer to assign a CFP average value for that item.

[3] The definition of a functional process in the E&Q approach differs from the COSMIC method standard definition. In a future version the COSMIC standard definition will be adopted. The functionality identified by the two definitions is intended to be exactly the same.

In order to make an estimate the measurer (after having gone through the preliminary steps of the standard method - defining boundaries of applications, layers and scope of measurement)[4] has to classify each part of the actual requirements as belonging to one level of the proposed categories. An assignment table will give the related size measure of that requirement. In this way not only leaves of the functionality tree may be directly quantified but also intermediate branches.

Applicability and reported use.

The Early & Quick COSMIC approximation technique is based on the capability of the measurer to classify a part of the actual requirements as belonging to a particular functional category. Each part of the actual requirements is to be classified, in order of increasing magnitude and number of composing elements at one of four levels, as a Functional Process, Typical Process, General Process, or Macro-Process. The reference table 8.1 then allows the measurer to assign a CFP value for that part of the actual requirements (this is applied for each identified level separately).

The most recently published values are depicted in table 8.1 [31].

Type	Level	Ranges / COSMIC Equivalent	min CFP	most likely	max CFP
Functional Process	Small	1 - 5 Data movements	2.0	3.9	5.0
	Medium	5 - 8 Data movements	5.0	6.9	8.0
	Large	8 - 14 Data movements	8.0	10.5	14.0
	Very large	14+ Data movements	14.0	23.7	30.0
Typical process	Small	CRUD (Small/Medium processes) CRUD + List (Small processes)	15.6	20.4	27.6
	Medium	CRUD (Medium/Large processes) CRUD + List (Medium processes) CRUD + List + Report (Small processes)	27.6	32.3	42.0
	Large	CRUD (Large processes) CRUD + List (Medium/Large processes) CRUD + List + Report (Medium processes)	42.0	48.5	63.0
General process	Small	6 -10 Generic FP's	20.0	60.0	110.0
	Medium	10 - 15 Generic FP's	40.0	95.0	160.0
	Large	15 - 20 Generic FP's	60.0	130.0	220.0
Macro process	Small	2 - 4 Generic GP's	120.0	285.0	520.0
	Medium	4 - 6 Generic GP's	240.0	475.0	780.0
	Large	6 - 10 Generic GP's	360.0	760.0	1,300

Table 8.1 – Estimation values for the functional categories of the Early & Quick technique.

Assigning each part of the actual requirements to a specific category higher than the Functional Process level is quite subjective. The detailed description of the technique gives guidance on assigning the proper category [30]. The accuracy of the technique is thus strongly dependent on the training and capability of the measurers who use it to understand the categories at the higher levels of documentation.

Strengths and weaknesses.

Strengths:

- Applicable when a significant part of the actual requirements is not yet known to a level of detail to allow functional processes to be identified.
- It can handle different levels of documentation and decomposition within the actual requirements.

Weaknesses:

- Assigning functional processes to a size class is a subjective element.

[4] N.B. This approach's use of 'preliminary steps' corresponds to the COSMIC method's 'Measurement Strategy' phase, but does not appear to be as rigorous as the process defined by COSMIC.

- The uncertainty of the method is only dependent by the ability of the estimator in identifying the correct entry in the table for the requirements statement which is not influenced by calibration. The definitions of a General Process ('an operational sub-system of the application') and of a Macro Process ('a relevant sub-system of the overall Information System of the user's organization') can lead to different interpretations by different measurers.
- Not designed for approximating enhancements.

Recommended area of application.

This technique is most suited when (a part of) the actual requirements is not detailed enough to identify functional processes. This technique should be used with caution and only after proper training in the correct use of the technique.

Research developments.

The proprietor periodically researches the technique in order to calibrate the weights of the categories' elements and for reporting the results of application.

An experiment reported by Almakadmeh [8] to evaluate the reproducibility and accuracy of this technique, concluded poor reproducibility when the same approximate measurement was carried out by different measurers within the experimental context reported.

Other experimental results have been reported in other contexts [32]. This technique may be used at very different levels of decomposition and documentation with greater uncertainty at low decomposition and documentation and very small uncertainty at high decomposition and documentation.

9

9. APPROXIMATION USING FUZZY LOGIC – THE EPCU MODEL.

Origin and approximation mechanism.

In 2012, Valdés *et al*. proposed a solution using a fuzzy logic model, referred to as the Estimation of Projects in a Context of Uncertainty – the EPCU model, to create an approximate sizing technique without the need to use local historical data [20].

The EPCU model takes into account:

- the experience-based linguistic variables used by estimation experts in the domain of estimation (approximation in this case) and
- the way experts combine these linguistic variables to approximate the functional size.

In practical experiments, Valdés reported that the EPCU estimation process for most of the projects was significantly better than the use of “expert judgment” estimation technique [34]. In addition, the EPCU model enables a systematic replication: whatever the skill-level of the people that assign values for the input variables, the EPCU model generates estimates with less dispersion than the “expert judgment” technique that is especially useful when there is inherent subjectivity in assigning a size class.

Applying the EPCU model has six steps:

1. Identification of the input variables
2. Specification of the output variable, i.e. the estimated functional size
3. Generation of the inference rules
4. Fuzzification
5. Inference rule evaluation
6. Defuzzification

The first three steps are related to the configuration of the estimation process and generate the “EPCU context”. An EPCU context is "a set of variables (inputs and output) and the relations that affect a specific project or a set of similar projects" [20].

This technique was developed for the early phases of a software development project, where most of the actual requirements are written in natural language, and more often the estimates are developed in an environment of uncertainty, because the current requirements are not fully known.

Two input variables were considered in the EPCU context for approximate sizing:

1. Variable 1: the functional process size, and
2. Variable 2: the number of objects of interest about which data is moved by the functional processes.

In 2014, and 2015 the solution was tested with a case study for an industry project where the actual requirements were made available for the measurer only as a list of use cases, in which is typical for the early phases of the software life cycle, i.e. the actual requirements were not detailed, for case study with a real industrial project, the EPCU size approximation technique

yielded better results than the equal size bands technique, while both techniques led to lower sizes. [21]

Research on the EPCU size approximation technique has focused on two documentation levels of the FUR description: Functional Process, and Use Case.

Applicability and reported use.

From the EPCU model definition, in the first steps, a set of variables (inputs and output) and the relations between them are defined (EPCU context), the EPCU application to the approximation of functional size, the EPCU context defined uses two input variables that impact in the functional size of a functional process or use case defined as output variable.

The output variable was defined as a continuous range of possible values with an upper boundary, or cut-off, at 16.4 CFP for one context based in the equal size bands technique defined by Vogelezang *et al*. [16]. In 2015, Valdés *et al*. [17] proposed another version of their fuzzy logic size approximation technique, defining an additional context with an upper boundary or cut-off at 44.

In 2017, Valdés [18] investigated and compared using a non-parametric test, which of the EPCU contexts appeared to better represent the distribution of the REAL sizes, when the documentation level was Functional Process. In this study, it was statistically demonstrated that:

- distribution for approximation values using cut-off, at 16.4 CFP was similar to REAL value distribution employing the standard COSMIC method with 180 Functional Process, and
- in [19] distribution for approximation values using upper boundary, at 44 CFP was similar to REAL value distribution employing the standard COSMIC method using a large sample of 293 Use Cases from real projects.

Considering the findings in the research, it is possible to define when the documentation level of the FUR description was Use Cases, the EPCU context with cut-off at 44 CFP is recommended. On the other hand, when the documentation level of the functional user requirements description was Functional Process, the EPCU context with upper boundary at 16.44 CFP is recommended.

Since 2015, the EPCU approximation technique has an extensive use in Mexico, is a fundamental element for the Mexican database that relate functional size with effort and cost.

Strengths and weaknesses.

Strengths:

- Applicable when a significant part of the actual requirements is not yet known to a level of detail to allow functional processes to be identified.
- It can handle different levels of documentation and decomposition within the actual requirements.
- Does not need local historical data to provide a size estimate, especially when most currently available approximation techniques for sizing the functional size of software requiring a calibration process employing historical data for better results in local contexts, however, collecting such data may be both expensive and time-consuming and approximation techniques based on historical data are of little use without such data.

- Exhibits good behavior, even when individuals are not acquainted with the COSMIC method.
- It has an intensive use in Mexico.

Weaknesses.

- Applying this technique involves a number of steps that require trained input, which makes it challenging for use in industrial software engineering projects.

- The easy way to use the EPCU technique, is to use the commercial version, built by the method proprietor.

Recommended area of application.

This technique is most suited when (a part of) the actual requirements is not detailed enough to fully describe the functional processes or use cases, when only the identification of functional process/use cases is made, that means in early phases of software development cycle.

Also this technique could be used when there is no historical database to calibrate specific approximation technique.

Research developments.

In 2012, Valdes Souto and Abran [20] reported on a case study using a fuzzy logic model to approximate the functional size of a software, when the FUR's are known and fully documented as in the C-registration Case Study. This research shows that the equal size bands approximation was a better approximation technique than the EPCU fuzzy logic model with the similar conditions.

In 2014 [21] further research with a case study aiming to simulate a real early approximation using the EPCU model for an industry project for which only the names of the Use Cases were made available to participants. This case study confirmed that the EPCU size approximation technique does not require local calibration and is useful when there are no historical data available. For a case study with a REAL industrial project, not a reference software, the EPCU size approximation technique yielded better results than the ESB technique, while both techniques led to lower sizes.

In 2015, Valdés *et al*. [17] proposed another version of their fuzzy logic size approximation technique. It defined a continuous range of possible values for the output variable with an upper Q4 (4th Quartile) cut-off of 44 CFP for a Functional Process using the dataset of Vogelezang *et al*. [16]. For the study of an industry project that considered Use Case documentation level, the EPCU cut-off at 44 CFP [17] yielded better results on comparison with the ESB technique and EPCU cut-off at 16.4 CFP [21], that underestimated the functional size. On the other hand, more realistic results were obtained using the EPCU cut-off at 44 CFP.

Research on the EPCU size approximation technique has focused on two documentation levels of the FUR description [20]: Functional Process, and Use Case, using two EPCU context definitions; however, it was not clear when to utilize each EPCU context (EPCU16.4, EPCU44), in order to analyze which of the two has a better performance for each documentation level of functional requirements. In 2017, Valdés [18] investigated and compared using a non-parametric test, which of the EPCU contexts appeared to better represent the distribution of the REAL sizes, when the documentation level was Functional Process.

There is no standard definition for Use Case, and it has been observed that frequently that Use Cases involve more than one Functional Process, sounds logical that the EPCU approximation technique with a cut-off of 44 CFP might be more useful if functional requirements are at the documentation level of Use Cases, a situation that is occurs very frequently in the industry, in 2017, a similar research oriented to investigated and compared using a non-parametric test, if the EPCU cut-off of 44 CFP betters represent the distribution of the REAL sizes, when the documentation level was Use Cases [19].

Based on the findings of [18] the valid conclusion is that the EPCU with a cut-off of 16.4 CFP is useful when the documentation level was Functional Process and the EPCU with a cut-off of 44 CFP is recommended when the documentation level is at the Use Case level.

10

10. EASY FUNCTION POINT APPROXIMATION.

Origin and approximation mechanism.

The EASY (EArly & SpeedY) approximation technique was first introduced in 2012, based on a more generic Software Measurement Approximation Rapid Technique (SMART) to size fuzzy actual requirements [41]. In the 'SMART' technique, on any function the measurer is free to assume one or more value 'possibilities' based on an understanding of the actual requirements describing the functionality.

> Example: "This report is 'most probably' 5-data-movements (60%), but it 'might have' 2 additional data movements (30%), or even 4 additional data movements (to be confirmed) (10%)." The approximate value for the function is the weighted sum of all possible values (where the weights are the corresponding probabilities. In the example, this would mean an approximate size of 5x0.6 + 7x0.3 + 9x0.10 = 6.0 CFP). (All probabilities of options for one function must sum to 100%.)

Note that the most probable value is not necessarily always the 'middle' one. It is up to the measurer to assign the probabilities on the possible values. This is different from any 'average' technique depicted in previous chapters, where average or middle values are taken as being 'always' the most likely values.

However, the 'SMART' technique might be time-consuming, for the measurer to assign more than one possible value to each function being sized, and a corresponding probability to each value per function.

The EASY approximate technique provides most typical probability distributions for the measurer to pick from, and allows for approximate sizes and accurate sizes to be mixed. (Accurate sizes, that is sizes measured according to the standard measurement method, correspond to sizes where a value is assigned with a close-to-100% probability).

Table 10.1 shows typical probability distributions of approximate values for most common cases in the Business domain (FP stands for 'Functional Process') [41].

Classification of the FP	Specification level	CFP (min)	CFP	CFP (max)	Approximate CFP	Probability
Small FP	Little unknown	2 (10%)	3 (75%)	5 (15%)	3.2	>80%
Small FP	Unknown (No FUR)	2 (15%)	4 (50%)	8 (35%)	5.1	<50%
Medium FP	Little unknown	5 (10%)	7 (75%)	10 (15%)	7.25	>80%
Medium FP	Unknown (No FUR)	5 (15%)	8 (50%)	12 (35%)	8.95	<50%
Large FP	Little unknown	8 (10%)	10 (75%)	12 (15%)	10.1	>80%
Large FP	Unknown (No FUR)	8 (15%)	10 (50%)	15 (35%)	11.45	<50%
Complex FP	Little unknown	10 (10%)	15 (75%)	20 (15%)	15.25	>80%
Complex FP	Unknown (No FUR)	10 (15%)	18 (50%)	30 (35%)	21	<50%

Table 10.1 – Probability distributions of approximate values in the business domain.

Different choices of probability distributions, as well as minimum and maximum CFP values for the several cases of Functional Process above, lead to different instantiation of the EASY approximation technique. A similar case can be made for the Real-time domain.

A similar technique, with different sizes and probabilities, can be used for approximating the sizes of 'small' to 'large' functional changes (for enhancement projects).

Applicability and reported use.

In the literature there is no reported use of this approximation.

Strengths and weaknesses.

Strengths:

- It can be mixed with standard measures.
- It can be scaled to different levels of documentation.
- It works for enhancement projects (size of changes, instead of sizes of functions).

Weaknesses:

- It might be time-consuming to establish the calibration and apply it to real requirements.
- It relies on the choice of the 'typical' cases to map the fuzzy actual requirements onto.

Recommended area of application.

This approximation is valid throughout the evolution of the actual requirements, as their description evolves in time.

Research developments.

Usage data is being collected for validation and improvement purposes.

Part III

Techniques in the Research Stage

.

11

11. EMERGING APPROXIMATION TECHNIQUES.

The approximation techniques in this chapter are still in an early stage of development. We believe that these techniques have the potential to evolve into approximation techniques or tools that can be used in the near future.

11.1 Approximation from informally written textual requirements.

The approximation technique from informally written textual requirements [33] builds on work on automating COSMIC functional size measurement from formal requirement specifications:

- First a number of informal textual requirements must be selected to describe a single functional process and then be manually measured.
- The textual requirements and the corresponding sizes are stored per functional process in a database to act as reference.
- Then the measured functional processes are divided into four fuzzy size classes based on the quartile boundaries of the total dataset.
- Then with text mining, linguistic features are extracted from the dataset to train a text classification algorithm that can automatically classify a new set of textual requirements belonging to one of the four fuzzy size classes.

Strength: after the preparation stage it can be fed with textual requirements that can automatically provide a size estimate. In the experiment, textual requirements from various sources were used to test the technique.

A weakness could lie in different linguistic characteristics in different environments, and the easy replication for distinct languages, meaning that local calibration would be required for each environment.

11.2 Approximation based on the average number of data groups.

In 2014, Del Bianco et al. reported that in their dataset the average number of data groups per functional process was a better predictor of functional size than the average number of data movements [13]. They reported that the number of data movements per data group involved in a functional process is quasi constant and that within their dataset the following formula gave a good estimation of COSMIC functional size:

$$\overline{CFP} = AvDGperFP * 1.8 * \#FPr$$

The prerequisite for using this approximation technique is that the organization performs full-fledged COSMIC measurement to be able to collect the necessary historical data needed to compute these estimation formulas.

11.3 Approximation based on Use Case names.

In 2016, Ochodek proposed an approximation technique on Use Case names in a particular environment in search for candidate techniques for automated functional size measurement

[35]. State-of-the-art guidelines for writing Use Cases advise to document them with a name that accurately expresses the goal of an actor using a simple clause with an implied subject. These names can be processed and assigned into one of thirteen categories:

- Check Object 4.25 CFP
- Asynchronous Retrieve 5.10 CFP
- Delete 5.87 CFP
- Create 7.01 CFP
- Dynamic Retrieve 7.56 CFP
- Retrieve 7.84 CFP
- Change State 8.23 CFP
- Complex Internal Activity 9.00 CFP
- Delete Link 9.00 CFP
- Transfer 9.19 CFP
- Link 13.70 CFP
- CRUD 17.60 CFP

It is important to mention that these types characterize Use Cases rather than single data movements. Some of these types are self-explanatory, e.g., Create or Delete. Other ones make more subtle distinctions between the use-case goals.

> For instance, Complex Internal Activity characterizes Use Cases that aim at running complex algorithms/processing. Usually, the scenarios of such Use Cases look trivial but when analyzed from the perspectives of the COSMIC method, it quickly becomes apparent that they involve single or multiple entry data movements (most often processing parameters and input data) as well as read and write movements.

The detailed description of this and other types of Use Cases can be found in [35].

Based on historical data on these categories a functional size can be determined. The author did a validation that the proposed categories are both complete and have a sufficient degree of discriminating efficiency to effectively classify all different Use Cases across different domains. Although most of the Use Cases were following the guidelines for writing Use Cases, for some of them anomalies have been found related to their proper naming. It has been observed that:

- 4% of Use Cases had misleading names that did not correspond to the semantics of their scenarios.
- Another 2% were so-called CRUD (Create, Retrieve, Update, Delete) or partial-CRUD Use Cases whose names suggested only one of the CRUD operations.

11.4 Approximation based on actions in UML Use Case diagrams.

In 2017, Haoues *et al.* proposed an approximation based on actions that can be easily retrieved from UML Use Case diagrams [36]. Although the measurement formulas are proposed for and tested with web and mobile applications, the concepts are generic for any type of software documented with UML Use Case diagrams.

The technique is based on the observation that the functional size of a functional process is bounded by a minimum and a maximum value, according to the following rule:

$$2 \leq \mathrm{FS(FP)} \leq \mathrm{FS(A)} + \mathrm{FS(S)} + \mathrm{FS(E/C)}$$

where 2 is the minimum size for any given functional process and the maximum can be determined by the sum of the functional size of actions from the Actor - $\mathrm{FS(A)}$, actions of the System - $\mathrm{FS(S)}$ and actions for Error handling and Confirmation messages - $\mathrm{FS(E/C)}$.

In the investigated dataset the measured functional size was always between 77-100% of the maximum functional size of the software. When calibrated with historical data, this method offers the potential of automated approximation from UML diagrams.

This technique can also be applied to approximate the size of changes that occur throughout the software life cycle. Changes in system requirements can be classified as either functional or technical. Functional changes affect FUR, while technical changes affect NFR or PRC.

Changes are most often expressed in natural language by change requesters (*e.g.,* customers, users, development teams, etc.). The technique proposed in [36] can be used with Machine Learning algorithms to approximate the size of a change. The benefits of applying this technique with Machine Learning will allow decision-makers to monitor rapidly change requests at different levels of details.

11.5 Approximation based on Equal Number Bands.

In 2019 Lavazza and Morasca tested two alternatives for the Fixed Size Classification technique [14]. They reported that these techniques could lead to a better prediction of the actual size than the Fixed Size Classification technique.

The approximation based on Equal Number Bands divides the set of functional processes into ordered adjacent bands that contain the same number of functional processes, i.e., such that

$$\forall i,j\ |Band_i| = |Band_j| \wedge i < j \Rightarrow \forall l,m\ \ FP_l \in Band_i\ \wedge\ FP_m \in Band_j \Rightarrow\ size(FP_l) < size(FP_m).$$

The band reference size to be used for estimation purposes is set to the average size of the functional processes belonging to the band.

In practice, however, the actual bands can only approximate the definition, because the number of functional processes is not always a multiple of the number of bands. Suppose we have a total of 98 functional processes, which we want to divide into 4 bands: the best we can do is to have:

- two bands with 25 functional processes and
- two bands with 24 functional processes.

11.6 Approximation based on Equal Range Bands.

In 2019 Lavazza and Morasca tested two alternatives for the Fixed Size Classification technique [14]. They reported that these techniques could lead to a better prediction of the actual size than the Fixed Size Classification technique.

The bands are defined so that all ranges have equal width.

> So, if we have a dataset where the minimum functional process size is 3 CFP and the maximum functional process size is 44, and we want to define 4 bands, we shall have bands with range width $\frac{44-3}{4}$ = 10.25 CFP. Therefore, the borders between the bands are at 13.25 CFP, 23.5 CFP, and 33.75 CFP.

Since the size of functional processes must be an integer, the Small band includes functional processes in the [3,13] range; similarly, the other bands include functional processes in the [14,23], [24,33], and [34,44] CFP range, respectively.

The band reference size to be used for estimation is set to the midpoint of the band bounds. For instance, a band including functional processes in the [24,33] CFP range is assigned a reference size of $\frac{24+33}{2}$ = 28.5 CFP.

Part IV

General Concepts.

12

12. DIFFERENT LEVELS OF DOCUMENTATION AND DECOMPOSITION

This chapter discusses some aspects of functional size measurement that must be considered to ensure that measurements made on different sets of actual requirements are comparable with respect to the levels of documentation and decomposition. The ideas described here can arise when functional size measurements must be made in the early stages of a large software project and in general when it is necessary to ensure the comparability of size measurements across the various parts of the actual requirements.

These aspects of measuring functional sizes need to be considered in the Measurement Strategy phase of the process described in the Measurement Manual. The ideas are not specific to the COSMIC method, but in principle are relevant to any functional size measurement method.

12.1 The evolution of requirements in the early stage of a large software project

In the early stage of a large software development project, when the actual requirements are first being established, one of the two following techniques may be followed.

- First, the actual requirements are being defined in ever-more detail.
- Second, the actual requirements may be split into smaller, well-demarcated, more manageable, 'chunks', so that separate teams can work on them in parallel. These separate 'chunks' may later be developed as separate pieces of software, e.g. as separate 'sub-systems'.

The result may be (and this has been observed on several occasions in practice) that when a first measurement of size is required, the actual requirements to be measured exist at various 'levels of documentation' (also called 'level of granularity' in the Measurement Manual v5.0) and at various 'levels of decomposition'. It is easy to confuse these two concepts. Therefore both concepts are described below.

DEFINITION – Level of decomposition
Any level resulting from dividing a piece of software into components (named 'Level 1', for example), then from dividing components into sub-components ('Level 2'), then from dividing sub-components into sub-sub components ('Level 3'), etc. NOTE 1: Not to be confused with 'level of documentation'. NOTE 2: Size measurements of the components of a piece of software may only be directly comparable for components at the same level of decomposition.

When faced with actual requirements documents that may or may not be at the same level of documentation and/or at the same level of decomposition, the measurer must clearly examine these levels before establishing any scaling factors, as described in chapter 1.

DEFINITION – Level of documentation

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

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

Accurate COSMIC functional size measurements require that the actual requirements to be measured exist at a level of documentation at which functional processes and their data movements can be identified (i.e. the level at which we refer to them as 'Functional User Requirements' (FUR). The 'functional process level of documentation' is defined as follows.

DEFINITION - Functional process level of documentation

A level of documentation of the description of a piece of software at which

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

NOTE 1: In practice, software documentation containing functional requirements often describes functionality at varying levels of documentation, especially when the documentation is still evolving.

NOTE 2: 'Groups of these' (functional users) might be, for example, a 'department' whose members handle many types of functional processes; or a 'control panel' that has many types of instruments; or 'central systems'.

NOTE 3: 'Groups of events' might, for example, be indicated in a statement of FUR at a high level of documentation by an input stream to an accounting software system labeled 'sales transactions'; or by an input stream to an avionics software system labeled 'pilot commands'

RULES - Functional process level of documentation

a) Accurate functional size measurement of a piece of software requires that its FUR are known at a level of documentation at which its functional processes and data movement sub-processes may be identified.

b) If some requirements must be measured before they have been defined in sufficient detail for an accurate measurement, the requirements can be measured using an approximate technique. These techniques define how requirements can be measured at higher level(s) of documentation. Scaling factors are then applied to the measurements at the higher level(s) of documentation to produce an approximate size at the level of documentation of the functional processes and their data movement sub processes. See the 'Guideline for Early or Rapid Functional Size Measurement'.

This Guide describes several techniques to implement rule b).

As described in section 1.4, a problem for all approximation techniques is that there is no way of unambiguously defining standard levels of documentation higher than the functional process level. Furthermore measurers, especially if inexperienced, often do not realize that actual requirements are expressed at different levels of documentation and/or fail to distinguish the

levels. This is one of the commonest problems faced when intending to measure functional sizes, whether accurately or approximately.

To assist measurers in assessing the measurability of requirements COSMIC has compiled a Guideline for Assuring the Accuracy of Measurements [37]. With this guideline the requirements can be scored in the following categories:

A. Completely defined
B. Partially Documented
C. Identified
D. Counted
E. Implied (a 'known unknown'), not mentioned or missing (an 'unknown unknown')

The following are examples of statements of requirements at different levels of documentation and how they should be analyzed before applying an approximate technique to functional size measurement.

Statement of Requirements	The Level of Documentation and how to analyze the Requirements	Cat.
"The software system shall control all processes of the washing machine, including washing cycles, heating, filling, emptying, user control panel interface, etc."	A very high level of documentation. Very difficult for someone without experience to measure even approximately without further detail. However, an experienced measurer with knowledge of the washing machine's hardware, should be able to at least list the number of functional processes and then use an approximate sizing technique.	**E**
The "software shall enable personnel officers to maintain data about all permanent employees."	The word 'maintain' usually implies at least functional processes to create, read, update and delete data (remember the acronym 'CRUD'). It is reasonable to measure using an approximation technique, but the measurer must check if the list of functional processes is complete. (Several types of enquiry and update functional processes could be needed.)	**C**
"I want to be able to enquire on the order backlog which must be up-to-date at any time." (Example of a possible 'User Story' for an Agile development)	This actual requirement appears to define a single enquiry functional process. It may even be that enough is known from the context that the functional process can be measured accurately. But the actual requirement is not clear. The Story may imply other functional processes. The measurer must ask, e.g. a) what does 'order backlog' mean? How detailed is the enquiry – by order, by customer, by product, by time since ordered, etc? b) what functionality is needed to maintain the order backlog up-to-date at any time?	**B**
"Access of customers to the system over the web shall be subject to industry-standard security, requiring e-mail address as the ID and a password."[5]	Most measurers should be able to list the functional processes for this actual requirement, e.g. to allow new and existing customers to access the system, to handle forgotten passwords, a facility to change a password, etc. So it should be possible to measure at least an approximate functional size.	**D**

[5] This security requirement might be considered as a 'Non-Functional Requirement' (NFR). But like many other NFR's, the requirement is satisfied by software and this software can be measured by the COSMIC method.

Statement of Requirements	The Level of Documentation and how to analyze the Requirements	Cat.
"Monthly reports shall be produced for the Sales Managers at Branch, Region and National levels."	This actual requirement is unclear. It might specify three functional processes, but because no detail of the data movements is given, it also might only indicate a different sorting. Without further details, the approximate sizes could have a wide range of uncertainty.	**E**
"Every 10 seconds the software shall read and display the external temperature and update the temperature history log."	A fully specified functional process with its data movements. Can be measured accurately	**A**

To illustrate the type of difficulties faced by measurers, we provide two cases.

1. In the first case we briefly re-consider the example of a well-known system for ordering goods over the Internet, which is referred to as the 'Everest' system in the Measurement Manual where it is discussed in detail. This case illustrates the difficulties of measuring actual requirements at different levels of documentation.

2. In the second case we consider an example from a telecoms software architecture. This example illustrates the difficulties of measuring actual requirements that are being defined to lower and lower levels of documentation and are being decomposed into smaller 'chunks' at the same time, in parallel.

Case #1 Measuring at varying levels of documentation - the 'Everest' system

The case of the Everest system is described in version 4.0.2 of the Measurement Manual, section 2.4.3. The description of the part of the Everest system that is given and analyzed is highly simplified and "covers only the functionality available to Everest's customer users. It thus excludes functionality that must be present so that the system can complete the supply of goods to a customer, such as functionality available to Everest staff, product suppliers, advertisers, payment service suppliers, etc."

If we were to describe the total Everest application at its highest levels of documentation, we might show it as a set of functional areas, of which 'customer ordering' would be only one area. The other areas might be: internal processes (e.g. accounting); product supply; management; advertising; payment services; system maintenance; etc. We could 'decompose' the total application at this level, and then consider each functional area independently.

The task for someone who must measure the actual requirements of any one area would therefore be to understand the actual requirements as they evolve at lower and lower levels of documentation. The measurement scope would be defined as the 'customer ordering functional area'. The measurer would not have to think about 'decomposition' within this scope.

Recalling some observations on this case study, as we 'zoom-in' to lower levels of documentation of the actual requirements of the customer ordering functional area:

- the scope of the area to be measured does not change,
- the functional users (individual customers who place orders) do not change. A customer can 'see' the whole functionality of the area at all the levels of documentation of the analysis.

A further, and most important observation, was that "in practice when some functionality is analyzed *in a top-down approach*, it cannot be assumed that the functionality shown at a particular 'level' on a diagram will always correspond to the same 'level of documentation' as this concept is defined in the COSMIC method. (This definition requires that at any one level of documentation the functionality is 'at a comparable level of detail'.)"

As the diagrams in the Measurement Manual showing a possible analysis of the Everest ordering system illustrate, in practice functional processes can occur at various levels in such diagrams. The measurer must therefore examine each main branch, minor branch, or leaf of the system 'tree' to develop a scaling factor appropriate to that part. As in practice at any given moment all parts of a functional model will not have evolved to the same level of documentation, the same one scaling factor cannot be applied to each part.

Case #2 Measuring at varying levels of documentation & decomposition in a software architecture

The example in this section illustrates a technique to sizing the actual requirements of software as they are defined at lower and lower levels of documentation that differs from the technique described for the 'Everest' system above. This example is also from a different software domain, namely from a complex, real-time telecoms software architecture. The example was provided by a major manufacturer of telecommunications equipment, as an illustration of their current practice.

The description below uses the manufacturer's terminology [8].

The purpose is to measure the functional size as the actual requirements evolve, as input to a project estimating method.

Description and analysis of the software architecture

Figure 12.1 a) shows a 'Logical Network Element' (or LNE) within the software architecture, and the analysis of its functionality into two lower levels of documentation, namely the 'System Component' (or SC) level in Figure 12.1 b) and the 'Sub-system' (or SS) level in Figure 12.1 c).

Models such as shown in Figure 12.1 are produced in the telecoms company in three stages (at each level of documentation) and the goal is to be able to estimate the project effort to develop the whole LNE at each stage. The analysis described here is therefore a form of 'early approximate sizing'. But this case and its analysis also helps illustrate other issues.

A first key difference between the way this telecoms architecture is described and analysed compared with that of the 'Everest' example in section above is that at each level of documentation the measurement scope is sub-divided so that each 'component' revealed at each level is measured separately. (Remember, in the 'Everest' example, the measurement scope was unchanged as the analysis zoomed-in on the lower levels of documentation.) This technique therefore involves 'decomposition' of the functionality as it is analysed, in addition to the 'zooming-in'.

An inevitable consequence of decomposing a piece of software (and hence of decomposing its actual requirements and its measurement scope) is that new functional users appear with each decomposition. Example: if a piece of software is decomposed into two inter-related components, then the two components must become functional users of each other and will exchange data movements. Hence, if the total size of several components is needed, the measurer will have to consider the rules on aggregating size measurements (see the Measurement Manual)

The different measurement scopes of the LNE are shown in Fig. 12.1 by the solid line at the LNE level, the dashed lines at the SC level and the dotted lines at the SS level.

Logically, at each level of documentation, the components appear to communicate with each other directly. (In practice, of course, the components communicate via an operating system; this becomes obvious in practice at the lowest SS level of documentation, which is the level at which physical components are developed.) For sizing purposes, therefore, the components of the architecture at each level of documentation may be considered as functional users of each other.

Figure 12.1 a) shows, at the highest level of documentation, the 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 documentation, 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 persistent storage by LNE1.

At one lower level of documentation, Figure 12.1 b) shows that LNE1 is decomposed into four System Components, namely SC1 to SC4. 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 Figure does not illustrate this latter aspect).

Figure 12.1 b) shows that the single functional process at the LNE 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 LNE level.)

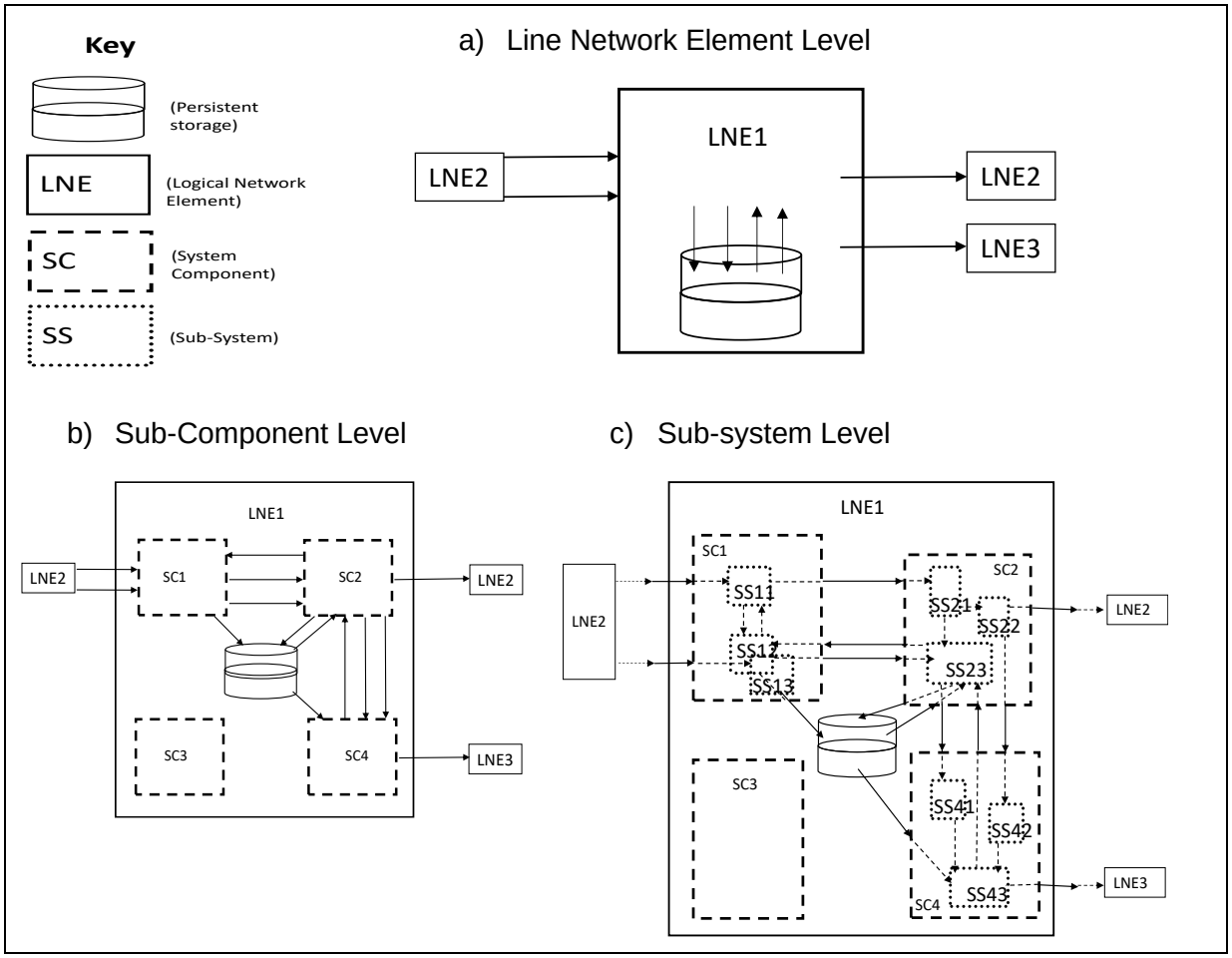

Figure 12.1 - A Line Network Element and its analysis into two lower levels of documentation.

At one lower level of documentation, Figure 12.1 b) shows that LNE1 is decomposed into four System Components, namely SC1 to SC4. 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 Figure does not illustrate this latter aspect).

Figure 12.1 b) shows that the single functional process at the LNE 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 LNE level.)

At the lowest level of documentation, Figure 12.1 c) shows that each System Component is decomposed into a number of Sub-systems (SS's). At this level, the functional users of any one Sub-system are either other Sub-systems within LNE1 or are Sub-systems in other LNE's (the latter is not illustrated in the Figure). The single functional process at the LNE level has now been decomposed into nine functional processes at this lowest level of documentation, one in each Sub-system.

At each level of documentation, some data is moved to persistent storage and some is retrieved from persistent storage. Figure 12.1 shows which components of LNE1 are involved in this functionality as we decompose to lower levels of documentation. (For diagramming convenience, persistent storage is shown as a 'common resource', irrespective of the levels of documentation and decomposition. Strictly speaking, as this Figure shows that at each level of documentation the functionality is also decomposed, resulting in new measurement scopes, persistent storage should be shown within each scope where it is used.)

For information, the physical sequence of Data Movements (DM) at the Sub-system level of documentation in LNE1 is as follows:

a) The triggering Entry to SS11 (of SC1 of LNE1) comes from LNE2 (sent by one of its SS's within one of its SC's)
b) After exchanges of DMs between SS's (which may be part of the same or different SC's), LNE1 sends an Exit (by SS22 of SC2) to a SS within LNE2 (e.g. for requesting more information from the original initiator of the functional process)
c) An SS within LNE2 then responds with another Entry (different from the triggering Entry) to LNE1 (actually to SS13 inside SC1)
d) Again after some internal DMs, LNE1 sends (by SS43 inside SC4) a final Exit to a SS within LNE3
e) In addition Reads and Writes take place during the process.

As stated above, it is only at the Sub-system level that project teams actually start to develop software; Sub-systems are autonomous applications. This is important because it is at this level of documentation that the telecoms software company that provided this example wishes to carry out individual project estimating.

Sizing the Logical Network Element.

With this analysis approach, the size of the functionality shown in Figure 12.1 apparently increases as more components and functional processes are revealed at the lower levels of documentation.

This 'growth' is analogous to what we see in 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, even though the unit of measure for both maps (e.g. the kilometre) is the same for each scale.

The sizes of the functionality on Figure 9.1 at each level of documentation are as follows.

- At the LNE level of documentation (one functional process) = 8 CFP
- At the SC level of documentation (three functional processes) = 20 CFP
- (At the SS level of documentation (eight functional processes) = 32 CFP

Note that as a check on the measurements in this example, the size at any one level of documentation can be obtained from the sizes at the immediately lower level by eliminating all the inter-component Entries and Exits for the components at the lower level.

Discussion of the analysis of the LNE example.

The first main observation from the analysis of this example is that when described in the terminology of the telecoms equipment manufacturer, it suggests a weakness in the definition of the 'functional process level of documentation' given in chapter 1, which states:

"A level of documentation of the description of a piece of software at which:

- the functional users are individual humans or engineered devices or pieces of software (and not any groups of these) AND (etc.)"

The difficulty in interpreting this definition in this context is that *all* the functional users in the LNE case are 'pieces of software', and it is impossible to define what is a piece of software in any way that is generally-applicable for our purpose.

The analysis approach and terminology of the telecoms equipment company illustrated here results in the measurement scope being re-defined at each level of documentation, and functional processes being defined at three levels of documentation, rather than the one standard level assumed by the definition.

This problem can be avoided by changing the telecom company's terminology so that the term 'functional process' would be used only at the Sub-system level. At the higher levels, terms such as 'super-process' and 'super-super-process' could be used.

But there is a more fundamental issue that whatever names are used, the definition of the 'functional process level of documentation' will be specific to this company. For the company this definition is relatively easy to interpret because it is the level at which they set project teams to work to develop 'individual …. pieces of software (and not any groups of these)', as per the definition. However, this approach does not guarantee that what this company means by 'a piece of software' will be comparable to that of another company. Software can be aggregated or decomposed to multiple levels of documentation, and the types of the software's components can vary with the technology used. One company's 'piece of software' might be a complete Logical Network Element. Another company's piece might be a single re-usable object (perhaps a saleable product), which would clearly be a different level of documentation.

The problem with the definition of the 'functional process level of documentation' should not arise when the context includes functional users that are 'individual human and/or engineered devices', which should always be accurately identifiable. In such a context, if there are also functional users that are 'pieces of software', then the level of documentation of those software users and the data exchanges with the software being measured should be the same as that of the human or engineered device functional users.

Computing 'scaling factors' for the LNE example.

Supposing that in the LNE example we are at the stage of having completed the specification partly at the highest level of documentation of the LNE's and partly at the SC level, and wish to determine the size of the eventual software at the lowest level of documentation of the Sub-systems for input to a project estimation method. For this, we need scaling factors to multiply the measured sizes of an LNE or a SC to obtain the size measured at the lowest SS level.

If we were using the size measurements given in previous sections on this LNE1 to calibrate an approximate technique to sizing other LNE's and their SC's at the SS level, we would conclude that the scaling factors to be used would be as follows.

- To scale a size measured at the LNE level to a size measured at the SS level, multiply the LNE size by 4.0 (32 / 8)
- To scale a size measured at the SC level to a size measured at the SS level, multiply the SC size by 1.6 (32 / 20).

In practice, it is likely that at the moment when a project estimate is required necessitating a functional size measurement as input, the actual requirements will have been developed at varying levels of documentation. In such circumstances the measurer will have to exercise judgment when estimating the size at the required level of documentation by using a mixture of actual and scaled measurements.

12.2 Functional size measurements and standard levels of decomposition.

The issue to be briefly dealt with in this section has already been referred to above, namely that to ensure comparability of measurements there is a need to standardize levels of decomposition as well as to use the standard functional process level of documentation. This can be difficult when requirements are evolving in different groups working in parallel on a large distributed software system.

Establishing the relationships between the level of documentation of the requirements (driven by the customer) and the level of decomposition of the software (driven by the system architects) is complex and is best carried out in four steps, as follows:

Starting point: We wish to measure a size of a large evolving software system for the purposes of effort estimation when the software architecture is broadly established but the requirements are still evolving.

Step 1. Identify the *level of decomposition* of each piece of software in the architecture which it is of interest to measure separately and define its measurement scope.

Examples of levels of decomposition of software, hence of possible measurement scopes:

- The telecoms system, the subject of section 12.2, could be measured at any of the LNE, system component or sub-system levels (as described in Fig. 12.1) or even at lower levels of decomposition such as that of sub-system major components, re-usable components, etc.

- In the business application domain, software can be measured at the level of a whole application, or of a major application component (e.g. of a 'n-tier' architecture) or an object class, or re-usable component, etc. (But note that in industries such as banking, where software systems have grown and evolved over decades, it can be difficult to define and distinguish an 'application'.)

Step 2. Identify the *level of documentation* of the actual requirements of each piece of software to be measured whose scope was defined in step 1.

Step 3. Identify the functional users of the piece(s) of software defined in Step 1 that *cannot be decomposed* (individual humans or pieces of hardware). Individual humans or pieces of hardware interact with pieces of software at only a limited number of software levels of decomposition of practical interest for measurement purposes. (Example, it may be useful to measure the size of business application software as seen by a human user at only two levels of decomposition, namely that of a whole application, or of the user interface component of a multi-tier application.

We can therefore define precisely the few combinations of human or hardware functional users and levels of decomposition of the software with which they interact. It follows that we can precisely identify the functional process level of documentation for these combinations, because we can identify the event-types that the functional users must respond to or that they generate. Hence we can identify the functional processes precisely and our functional size measurements can be reliably compared from different sources for these combinations. It also follows that we can apply the approximation techniques described in this Guide to any requirements that are at higher levels of documentation than the functional process level.

Step 4. Identify any functional users of the pieces of software identified in Step 1 that are other pieces of software. These *can exist at multiple levels of decomposition.* There are no *standard* levels of decomposition of software. In addition, the actual requirements of both the software being measured and of its software functional users can be expressed at multiple levels of documentation. It is therefore intrinsically difficult in practice to define a universal standard for a functional process level of documentation when the software being measured and all its functional users are pieces of software. It follows that it is equally difficult to identify and measure functional processes in a way that ensures measurements from all sources are comparable for this 'software/software' combination. Similarly, it is difficult to apply the approximation techniques described in this Guide for this 'software/software' combination.

These difficulties can be overcome within an organization or between collaborating organizations that can define local standards for software levels of decomposition, and for levels of documentation if needed for approximate sizing.

To enable greater size measurement comparability, COSMIC recommends that suppliers of services or tools that use functional size measurements specify standard levels of decomposition of software for which their service or tool can accept the size measurements.

COSMIC suggests the following as example candidates for standardization of levels of decomposition in the domain of business application software.

- A 'whole application'
- A major component of a whole application
- A re-usable object-class

The permitted functional users of software at all these levels of decomposition can be humans or other pieces of software at any of these levels of decomposition, Example: a whole application can be a functional user of a SOA component, and vice versa. But the case of a human user of a re-usable object-class is not likely to be of interest.

These standard 'levels of decomposition' are also typical examples of measurement scopes, as given in the Measurement Manual.

13

13. LOCALIZATION (CALIBRATION) GUIDELINES.

General guidance for establishing a locally-defined technique to approximate COSMIC sizing:

a) The local organization should define one or more (types of) artifact at a higher level of documentation than the level at which functional processes and their data movements are known, that describe the software functionality in a way that can be measured (i.e. at least identified and counted)

b) Artifacts selected in step a) might be, in *ascending* order of level of documentation, documents describing actual requirements at the functional process level, the 'Use Case' level, the sub-system level, etc. (Note that there is no standard terminology for levels of documentation above the level of a functional process[6].) Great care is therefore needed when defining standard artifacts suitable for approximate measurement above the functional process level of documentation.

c) The high-level artifacts selected for the measurements to calibrate the scaling factor must be representative of the software that needs to be approximated in the future by the locally-defined technique.

d) After applying a locally-defined technique to approximate some high-level actual requirements, it is important to learn from the experience by establishing the accuracy of the approximate measurements when the detailed actual requirements of the same software become known. This is done by:

- first measuring the accurate sizes for at least a sample of the detailed actual requirements.
- Then compare the approximate sizes with the accurate measurements to check that the scaling factor(s) used were reasonable.

  For instance, a result in a particular project could be that the actual total size when the final actual requirements are accurately measured turns out to be significantly greater than the measurements obtained by the approximate technique. The inaccuracy might be due to using inappropriate scaling factors and/or 'scope creep' on the project concerned. This result could be used to adapt the approximate sizing technique to take account of such factors in the future. (See section 14.2 for more on taking into account 'scope creep'.)

e) Given the uncertainty in approximate sizing, a range or some indication of the accuracy should be given when reporting an approximate size measurement, based on the established accuracy by comparing with the detailed size measurement as described under e).

f) The procedures to establish the accuracy of an approximate size measurement should also be established locally. The accuracy of any particular measurement will depend on the following two factors.

[6] Terms like 'Use Case' are of course defined, but in spite of such definitions, practice shows there is no guarantee that for a given actual requirement, two analysts will analyse the same number of Use Cases. Each organization must therefore establish its own understanding of what constitutes one Use Case.

- The level of detail and uncertainty of the requirements, which obviously varies with the state of progress of the project (See section 1.5 on the quality of requirements, the examples of section 12.1, and the guidance on 'scope creep' in section 14.2);
- The particular approximate sizing technique used for the measurement. (Example: the more detailed technique of Chapter 4 should, if well calibrated, give a more accurate size measurement than the technique of Chapter 2, for the same quality of the requirements.)

g) Best practice is then to produce a 'three point' estimate of the size, where the three points are the minimum size estimate, the most likely size estimate and the maximum size estimate. Presenting these three figures to show a range of uncertainty on the approximate size measurement is much more valuable to decision-makers and to anyone who must estimate project effort than just reporting the most-likely estimate.

h) The Early and Quick technique described in Chapter 8 includes a three-point size estimation element.

i) For more on the subject of three-point estimation, see for example Wikipedia.

14

14. APPROXIMATE SIZING OF CHANGES OF FUNCTIONALITY AND SCOPE CREEP.

14.1 Approximate sizing of changes to functionality.

To approximate the size of an actual requirement to change some existing software, the general guidance follows the same approach as for new software. If there is an existing catalogue of functional processes and their sizes (or size classification), then one of the two following techniques may be chosen.

a) If possible, based on the Functional User Requirements for the changes, judge which data movements of the relevant functional processes will be impacted and count these data movements;

b) Otherwise, estimate for each functional process the number or proportion of data movements that must be changed. For example, if a functional process to be changed is sized as 12 CFP and it is estimated that the change affects 25% of the data movements, then the size of the change is 3 CFP.

Use these numbers or proportions instead of the total sizes of the functional processes to be changed to complement one of the approximation techniques described above.

If there is no catalogue of existing functional processes, the first task would be to identify the functional processes affected by the actual change requirements, and then follow one of the approximate techniques above.

14.2 Approximate sizing and scope creep.

Experience shows that early in the life of a software development project, the functional size of the software tends to increase as the project progresses from outline actual requirements to detailed actual requirements, to functional specification, etc. This phenomenon, often referred to as 'scope creep', can arise because

- the scope expands beyond that originally planned to include additional areas of functionality
- and/or, as the detail becomes clearer, the required functionality turns out to be more extensive (e.g. to require more data movements per functional process) than was originally envisaged
- and/or Non-Functional Requirements may turn out to be (partly) implemented in software [38]

(It can also happen, of course, that the scope is reduced from the originally planned scope, e.g. due to budget cuts.)

The approximate sizing techniques described in this Guide do not explicitly take into account scope creep. When using these approximation techniques for early sizing therefore, potential scope creep should be considered as an additional factor. If potential scope creep is ignored, there is a risk of under-estimating the final software size and hence the project effort.

Estimating the potential for scope creep on a particular project goes beyond the scope of this Guide. However, it may be helpful to address the following questions:

- Are the actual requirements on this project particularly uncertain at the outset? If so, what correction (or 'contingency') to the approximate size should be made for possible scope creep?
- If scope creep is endemic within the organization, then we can use past measurements to help quantify this phenomenon. For instance, in a given organization and using a given development process for which many measurements exist, it may be possible to find a recurrent pattern such as 'by the end of phase 3, sizes are typically 30% greater than at the end of phase 1'.

15

15. CONCLUSIONS ON TECHNIQUES TO APPROXIMATE SIZING.

Techniques to approximate sizing can be made to work and are valuable for use early in a new software project's life and/or can save time and effort compared with sizing accurately using the standard COSMIC measurement method. Approximate sizing may also be necessary when the actual requirements are unclear. But approximate sizing techniques need to be used with care.

- Whatever the purpose of using an technique for approximate sizing, whenever further information becomes available enabling a more accurate and/or accurate sizing, the measurer should refine and update the measurement. This is especially required when using the measurement results as an input to estimation (such as effort prediction) – due to the phenomenon of error propagation [40].
- For obvious and similar reasons, no approximately measured size should be accepted as the 'actual' size in contractual situations, or analogous cases, where accurate figures are required – any preliminary approximate sizing should be replaced by standard measurements in the final stages of projects subject to such constraints.

When there is a need for approximate sizing, the organisation should:

- choose a technique which is optimal for the purpose of the measurement, given the availability of data for the calibration, the time available for the measurement and the accuracy required of the approximate size;
- calibrate the technique using accurately-measured local data on software that is comparable to that for which the approximate sizes must be measured;
- when actual requirements are unclear or incomplete, seek help to try to at least identify all the functional processes
- pay particular attention to identifying any large functional processes and to determining good scaling factors for them, as they can make a large contribution to the total size even though they are few in number;
- consider whether an allowance (or 'contingency') should be made for 'scope creep' and for the contribution that incorporating the Non-Functional Requirements may lead to when publishing an approximate size;
- estimate and report the plus or minus uncertainty on the approximate size, mentioning any contingency that has been made for scope creep; estimating the uncertainty on an approximate size is especially important in contractual situations.

References.

REFERENCES.

[1] C.R. Symons and A. Lesterhuis, “Introduction to the COSMIC method of measuring software”, version 1.1, January 2016.

[2] F.W. Vogelezang, C.R. Symons, A. Lesterhuis, R. Meli and M. Daneva, "Approximate COSMIC Functional Size: Guideline for approximate COSMIC Functional Size Measurement", 23rd International Workshop on Software Measurement and 8th International Conference on Software Process and Product Measurement – IWSM-MENSURA, Ankara, Turkey, 2013.

[3] C.R. Symons and A. Lesterhuis, "The COSMIC Functional Size Measurement Method Version 4.0.2 Measurement Manual, the COSMIC implementation guide for ISO/IEC 19761:2017", December 2017.

[4] H.C. Bauman, "Accuracy Considerations for Capital Cost Estimation", Industrial & Engineering Chemistry, April 1958.

[5] B. Boehm, “Software Engineering Economics”, Prentice-Hall, 1981.

[6] E. Berardi, C.R. Symons and S. Trudel, “Guideline for the use of COSMIC FSM to manage Agile projects”, September 2011.

[7] S. McConnell, "Software Estimation, Demystifying the Black Art", Microsoft Press, 2006.

[8] K. Almakadmeh, "Development of a scaling factors framework to improve the approximation of software functional size with COSMIC - ISO19761", PhD thesis, Ecole Technologie Supérieure - Université du Québec, Montréal, Canada,2013.

[9] A. Abran, J-M. Desharnais, S. Oligny, D. Saint-Pierre and C.R. Symons, "COSMIC-FFP Measurement Manual version 2.2, the COSMIC implementation guide for ISO/IEC 19761:2003".

[10] F.W. Vogelezang, "Early estimating using COSMIC-FFP", Proceedings of the 2nd Software Measurement European Forum, March 16-18, 2005, Rome.

[11] H.S. van Heeringen, E.W.M. van Gorp and T.G. Prins, "Functional size measurement – accuracy versus costs – is it really worth it?", Software Measurement European Forum - SMEF, Rome, Italy 2009.

[12] L. de Marco, F. Ferrucci and C. Gravino, "Approximate COSMIC size to early estimate Web application development effort", 39th Euromicro Conference on Software Engineering and Advanced Applications - SEAA, Santander, Spain, September 4-6, 2013.

[13] V. del Bianco, L. Lavazza, G. Liu, S. Morasca and A. Zaid Abualkishik, “Model-based early and rapid estimation of COSMIC functional size – An experimental evaluation”, Information and Software Technology, Volume 56, Issue 10, Oct. 2014, pp 1253-1267.

[14] L. Lavazza and S. Morasca, “Empirical evaluation and proposals for bands-based COSMIC early estimation methods”, Information and Software Technology, Vol.109, May 2019, pp 108-125.

[15] A. Lesterhuis and C.R. Symons, "The COSMIC Functional Size Measurement Method version 3.0 Advanced and Related topics", December 2007.

[16] F.W. Vogelezang and T.G. Prins, "Approximate size measurement with the COSMIC method: Factors of influence", Software Measurement European Forum - SMEF, Roma, Italia, 2007.

[17] F. Valdés-Souto and A. Abran, “Improving the COSMIC approximate sizing using the fuzzy logic EPCU model”, Joint International Workshop on Software Measurement and International Conference on Software Process and Product Measurement IWSM-MENSURA, Cracow, Poland, 2015.

[18] F. Valdés-Souto, Analyzing the performance of two COSMIC approximation sizing techniques at the functional process level, Sci. Comput. Program. 135 (2017) 105–121.

[19] F. Valdés- Souto, “Analyzing the Performance of Two COSMIC Sizing Approximation Techniques using FUR at the Use Case Level”, Joint International Workshop on Software Measurement and International Conference on Software Process and Product Measurement - IWSM-MENSURA, Beijing, China, September 2018.

[20] F. Valdes Souto and A. Abran, "Case Study: COSMIC Approximate Sizing Approach Without Using Historical Data", Joint 22nd International Workshop on Software Measurement and 7th International Conference on Software Process and Product Measurement – IWSM-MENSURA, Assisi, Italy, 2012.

[21] F. Valdés-Souto and A. Abran, “COSMIC Approximate Sizing Using a Fuzzy Logic Approach: A Quantitative Case Study with Industry Data”, Joint International Workshop on Software Measurement and International Conference on Software Process and Product Measurement - IWSM-MENSURA, Rotterdam, the Netherlands, 2014.

[22] L. Lavazza and S. Morasca, “An empirical evaluation of two COSMIC Early Estimation methods”, Joint 26th International Workshop on Software Measurement and the 11th International Conference on Software Process and Product Measurement – IWSM-MENSURA, Berlin, Germany, October 2016.

[23] C.R. Symons, and C. Gencel, ‘From requirements to project effort estimates: work in progress (still?)’, Requirements Engineering for Software Quality (REFSQ 2013) conference, Essen, Germany, April 2013.

[24] M. Ecar, F. Kepler and J.P.S. da Silva, “COSMIC User Story Standard”, International Conference on Agile Software Development (XP 2018), Agile Processes in Software Engineering and Extreme Programming, pp 3-18.

[25] S. Trudel, J-M. Desharnais, J. Cloutier, “Functional Size Measurement Patterns: A Proposed Approach”, 26th International Workshop on Software Measurement and the 11th International Conference on Software Process and Product Measurement – IWSM-MENSURA, Berlin (Germany), 5-7 October 2016.

[26] C.R. Symons, Guideline for ‘Measurement Strategy Patterns’, Ensuring that COSMIC size measurements may be compared, March 2013.

[27] C.R. Symons, ‘Software sizing and estimating: MkII FPA’, John Wiley & Sons, 1991.

[28] R. Meli, “Reference Manual release version 1.1 of IFPUG E&Q FP method release 3.1”, april 2012, dpo.it/wp-content/uploads/2018/01/EQFP-IFPUG-31-RM-11-EN-P.pdf

[29] R. Meli, "Early Function Points : a new estimation method for software projects", ESCOM 97, Berlin (Germany) 1997.

[30] T. Iorio, R. Meli and F. Perna, "Early & Quick Function Points® v3.0: Enhancements for a Publicly Available Method", Software Measurement European Forum (SMEF 2007), Roma, Italia, 2007.

[31] L. Santillo, "Early & Quick COSMIC-FFP Analysis Using the Analytic Hierarchy Process." 10th International Workshop on Software Measurement, *Lecture Notes in Computer Science 2006*, (pp. 147-160), Berlin (Germany) 2000.

[32] R. Meli, “Early & Quick Function Point Method - An Empirical Validation Experiment”, International Conference on Advances and Trends in Software Engineering (SOFTENG 2015), Barcelona, Spain, April 2015.

[33] I. Hussain, L. Kosseim and O.Ormandjieva, "Approximation of COSMIC functional size to support early effort estimation in Agile", Data & Knowledge Engineering 85 (2013) 2-14.

[34] F. Valdés, A. Abran, "Comparing the Estimation Performance of the EPCU Model with the Expert Judgment Estimation Approach Using Data from Industry", chapter 15, in 'Software Engineering Research, Management and Application 2010, published in 'Studies in Computational Intelligence', Volume 296, Springer-Verlag, Berlin, pages 227-240.

[35] M. Ochodek, “Functional size approximation based on use-case names”, Information and Software Technology, Volume 80, 2016, pp 73-88.

[36] M. Haoues, A. Sellami and H. Ben-Abdallah, “A Rapid Measurement Procedure for Sizing Web and Mobile Applications based on COSMIC FSM Method”, 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement – IWSM-MENSURA, Gothenburg, Sweden, October 2017, pp 130-138.

[37] A. Abran, A. Lesterhuis, and C.R. Symons, “Guideline for Assuring the Accuracy of Measurements”, version 1.1, July 2018.

[38] F. AbuTalib, D. Giannacopoulos and A. Abran, "Designing a Measurement Method for the Portability Non-Functional Requirement", 23rd International Workshop on Software Measurement & 8th International Conference on Software Process and Product Measurement – IWSM-MENSURA, October 2013, Ankara (Turkey).

[39] F.W. Vogelezang, C.R. Symons and A. Lesterhuis, "Web Advice Module – COSMIC Case Study", published January 2014 by NESMA (www.nesma.org).

[40] L. Santillo, "Error Propagation in Software Measurement and Estimation", 16th International Workshop on Software Measurement - IWSM, October 2006, Potsdam (Germany).

[41] L. Santillo, "EASY Function Points – ‘SMART’ Approximation Technique for the IFPUG and COSMIC Methods", 22nd International Workshop on Software Measurement & 7th International Conference on Software Process and Product Measurement – IWSM-MENSURA, November 2012, Assisi (Italy).

[42] A. Abran, S. Vedadi and O. Demirors, “Development of COSMIC Scaling Factors using Classification of Functional Requirements”, 29th International Workshop on Software Measurement & 14th International Conference on Software Process and Product Measurement - IWSM-MENSURA, October 2019, Haarlem (the Netherlands).

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

Glossary

GLOSSARY OF TERMS

The terms in this Glossary are specific to this Guideline. For other COSMIC terms, see the main Glossary in the Measurement Manual v4.0.2.

Accuracy. [43]
Closeness of agreement between a measured quantity value and a true quantity value of a measurand.
NOTE 1 The concept 'measurement accuracy' is not a quantity and is not given a numerical quantity value. A measurement is said to be more accurate when it offers a smaller measurement error.
NOTE 2 The term "measurement accuracy" should not be used for measurement trueness and the term "measurement precision" should not be used for 'measurement accuracy', which, however, is related to both these concepts.
NOTE 3 'Measurement accuracy' is sometimes understood as closeness of agreement between measured quantity values that are being attributed to the measurand.

Approximate Sizing.

1. Approximate measurement of a size.
2. Measurement of a size by an approximate technique.

Calibration

Determining the scaling factors or classification values to be used in the local environment in which the approximation technique is used instead of the scaling factors or classification values published in reference documents like this Guideline, aiming for the most accurate possible result of the application of the approximation technique.

Classification

Allocating a part of the actual requirements to a defined class (or reference piece) of requirements whose size has been calibrated in CFP.

Localization (Calibration)

Calibrating scaling factors or classification values to an environment that is representative of the environment the approximation technique is to be used in.

Precision.

The degree of exactness or discrimination with which a quantity is stated (ISO/IEC 24765:2010) *Example: a precision of 2 decimal places versus a precision of 5 decimal places.*

Scaling factor.

A constant that is used to convert a size measured under one set of conditions (e.g. one level of documentation of some actual requirements) to a size measured under another set of conditions (e.g. another level of documentation of the same requirements).