A Guide to Software Size Measurement

"As a software professional, I want to learn to use the COSMIC method, the most powerful standard way to size software, so as to help improve our software processes, the quality of our software products, and our estimating"

Charles Symons

Version 1.0, August 2020

Copyright 2020 Charles Symons. The author permits the free distribution and copying of this Guide for non-profit purposes. He acknowledges the permission of the Common Software Measurement International Consortium (COSMIC) to reproduce portions of text from the 'Measurement Manual', v4.0.2, December 2017.

Foreword

Knowing the size of your software is interesting, but *what you can do with measurements of software size is **valuable.***

If you know how to estimate or measure software sizes you have the basic means to:

- estimate the effort, time and then cost for a new development, early in its life;
- track **size** as the software is developed, to control 'scope creep' and manage risk;
- measure the productivity (= **size** / work-hours) and speed (= **size** / duration) with which the software was developed and is maintained;
- monitor the quality of the delivered software product (defect density = defects found in a given time-period / **size**)
- use measurements from across your software activities to learn how to improve organizational performance...... and more, limited only by your imagination!.

But first you have to learn how to measure software size, hence this Guide.

Intended readership and my aims for this Guide

In my experience of over 35 years of measuring and using software sizes, two observations stand out.

Firstly, organizations that routinely gather and exploit software metrics, and especially those who measure software sizes properly, often report huge economic benefits.

Secondly, few software managers and engineers now appreciate the benefits of software size measurement. This arises in part because size measurement is seen as a last-century activity which does not fit naturally with modern ways of developing software.

My main reason for writing this Guide, therefore, is to encourage software professionals to reconsider measurement as a way of enriching their mainstream activities by:

- describing the uses and value of measuring software size,
- showing how easy it is to measure software sizes using the COSMIC method - the most powerful and widely-applicable, ISO-standard[1] software-sizing method - and how measuring COSMIC sizes complements and enhances modern software development processes.

Specifically, as regards the COSMIC method, having read the Guide you should:

- be able to measure the size of requirements in units of 'COSMIC Function Points' (CFP) for business application, real-time and infrastructure software, at any level of decomposition from whole applications down to elementary requirements, for example for re-usable components or as in single User Stories;
- understand the COSMIC method sufficiently well that you can measure sizes from other types of software artefacts such as design models or for existing, installed software;
- understand how to approach the task of estimating approximate CFP sizes from less well-defined or outline requirements, for the purpose of early cost estimation;
- be able to pass the COSMIC 'Foundation Level' [1] certification examination.

[1] ISO/IEC 19761: 2011 'Software engineering – COSMIC: a functional size measurement method', International Organization for Standardization – ISO, Geneva, 2011.

(And by the way, when you are convinced by my story, try to persuade your manager to read at least the first Background chapter, and chapter 6 of this Guide.)

Achieving mastery of any software engineering method requires, of course, more than just understanding the basics; it requires practical experience. Although this Guide gives many simple examples of applying the COSMIC method, you will likely encounter more complex cases in practice that are challenging for the inexperienced measurer. Fortunately, the COSMIC web-site www.cosmic-sizing.org has many Guidelines, Case Studies and research papers that you can download for further advice if you need it.

The Guide is compatible with the standard description of the method, the Measurement Manual (the 'MM'), version 4.0.2, published in 2017 [2]. However, the Guide provides a shorter and simpler account of the method, more suited to first-time readers, with more advice on how to apply the method in practice. Some definitions and rules have been abbreviated, and a few rules in the MM concerning exceptional cases are only referred to from the Guide.

Structure of the Guide

The Guide opens with a 'Background' chapter which gives a brief introduction to the uses and value of measuring software sizes, the various ways that size can be measured, and why the COSMIC method is uniquely valuable.

The core of the Guide, Chapters 1 – 5, has the same structure as the MM v4.0.2, using the same section and figure numbering. If, therefore, at any point in this Guide you need more explanation or justification for a rule, or more examples to help understand a topic, just refer to the same section in the MM for more detail. Also refer to the MM for the complete Glossary of terms and their full definitions.

Chapter 6 discusses several topics that you will probably want to consider when preparing to implement COSMIC size measurement in practice, for example:

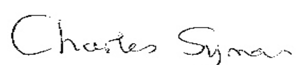
- How to measure an approximate COSMIC size from early, incomplete requirements.
- How to use the COSMIC method in an Agile development environment so as to gain the benefits of standard size measurements without disrupting existing Agile processes.
- How to use measurement results to establish local benchmarks, and to use them for effort estimation and for improving organizational performance.

This Chapter also presents evidence for the excellent correlations of COSMIC-measured sizes with development effort, and with memory space, which demonstrates that COSMIC sizes can be reliably used for all the various purposes that we claim.

Chapter 7 provides a large number of exercises and two mini case studies to test your understanding of what you have learned in the Guide and of whether you can now apply the COSMIC method in real-world scenarios.

About COSMIC (the Common Software Measurement International Consortium)

The COSMIC organization was founded in 1998 to develop a functional size measurement method based purely on fundamental software engineering principles. It is an 'open', not-for-profit, world-wide organization of software metrics experts whose publications are available for free download from www.cosmic-sizing.org. The basic principles of the COSMIC method have not changed since they were first published in the year 2000.



COSMIC co-Founder and past President

Acknowledgements*:* I am very grateful for their very helpful reviews of version 1.0 this Guide by:

- Allan Edwards (UK),
- Colin Hammond (ScopeMaster.com, UK);
- Lonnie Franks (a 'USA COSMIC Champion');
- Paul Piechota (The Dayton Improvement Group, USA);
- Sanath Rajagopal (QinetiQ, UK);
- Andrea Salvatori (Double Consulting, Italy)

Table of Contents

1 BACKGROUND: THE VALUE OF MEASURING SOFTWARE SIZE 8

Why measure software size? 8

Different ways of measuring software size 9

What uniquely distinguishes the COSMIC method? 9

2 INTRODUCTION 10

1.0 Chapter summary 10

1.1 Applicability of the COSMIC method 10

1.2 Functional User Requirements 11

1.2.1 Extracting the COSMIC concepts from the software artefacts 11

1.2.2 The process of deriving the COSMIC concepts from software artefacts 12

1.2.3 Non-Functional Requirements 12

1.3 The fundamental principles of the COSMIC method 13

1.3.1 The Software Context Model 13

1.3.2 The Generic Software Model 14

1.3.3 Types versus occurrences 16

1.4 The COSMIC measurement process 16

1.5 Perceived limitations on the applicability of the COSMIC method 16

1.6 Some simple examples to introduce COSMIC size measurement in practice 17

3 THE MEASUREMENT STRATEGY PHASE – WHAT SOFTWARE IS TO BE MEASURED AND WHY? 20

2.0 Chapter summary 20

2.1 Defining the purpose of a measurement 20

2.2 Defining the scope of a measurement 21

2.2.1 Deriving the measurement scope(s) from the measurement purpose 21

2.2.2 Layers 21

2.2.3 Levels of decomposition 22

2.3 Identifying the functional users and recognizing persistent storage 23

2.3.1 Functional size may vary with the choice of functional users 23

2.3.2 Persistent storage and the boundary 23

2.3.3 Context diagrams 24

2.4 Identifying the level of granularity of Functional User Requirements (FUR) 24

2.4.1 The need for a standard level of granularity 24

2.4.2 Clarification of 'level of granularity' 24

2.4.3 The standard functional process level of granularity 25

2.5 Measurement Strategy Patterns 25

2.6 Concluding remarks on the Measurement Strategy Phase 26

4 THE MAPPING PHASE 27

3.0 Chapter summary 27

3.1 Mapping from the software artefacts to the concepts of the Generic Software Model .. 27

3.2 Identifying functional processes 28

3.2.1 Definitions 28

3.2.2 The approach to identifying functional processes 29
3.2.3 Triggering events and functional processes of business applications 30
3.2.4 Triggering events and functional processes of real-time applications 30
3.2.5 More on separate functional processes 30
3.2.6 Measuring the components of a distributed software system 31
3.2.7 Independence of functional processes sharing some common functionality 31
3.2.8 Events that trigger a software system to start executing 31
3.3 Identifying objects of interest and data groups 31
3.3.1 Definitions 32
3.3.2 About the identification of objects of interest and data groups 32
3.3.3 Data or groups of data that are not candidates for data movements 34
3.3.4 The functional user as object of interest 34
3.4 Identifying data attributes (optional) 34
3.5 Definitions and Principles for data movements 34
3.5.1 Definitions of the data movement types 34
3.5.2 Identifying Entries (E) 35
3.5.3 Identifying Exits (X) 35
3.5.4 Identifying Reads (R) 35
3.5.5 Identifying Writes (W) 36
3.5.6 On the data manipulations associated with data movements 36
3.5.7 Data movement uniqueness and possible exceptions 37
3.5.8 When a functional process is required to move data to or from storage 37
3.5.9 When a functional process requires data from a functional user 37
3.5.10 Navigation and display 'control commands' for human users 38
3.5.11 Error/Confirmation Messages and other indications of error conditions 38
3.5.12 Identifying data movements that must be modified 39
3.6 Identifying COSMIC concepts in available software artefacts 40
5 THE MEASUREMENT PHASE 42
4.0 Chapter summary 42
4.1 The Measurement Phase 42
4.2 Applying the COSMIC unit of measurement 42
4.3 Aggregating measurement results 43
4.3.1 General rules of aggregation 43
4.3.2 More about functional size aggregation 44
4.4 More on measurement of the size of changes to software 44
4.4.1 Modifying functionality 44
4.4.2 Size of functionally-changed software 44
4.5 Extending the COSMIC measurement method 45
4.5.1 Introduction 45
4.5.2 Data manipulation-rich software 45
4.5.3 Limitations on the factors contributing to functional size 45
4.5.4 Limitations on measuring very small pieces of software 45
4.5.5 Local extension with complex algorithms 45
4.5.6 Local extension with sub-units of measurement 45
6 MEASUREMENT REPORTING 46
5.0 Chapter summary 46
5.1 Labeling 46
5.2 Archiving COSMIC measurement results 46
7 COSMIC SIZE MEASUREMENT IN PRACTICE 47

6.0 Chapter Summary 47
6.1 Estimating an approximate COSMIC size from incomplete FUR 47
6.2 Using COSMIC sizing in Agile software development... 48
6.3 CFP size/effort data and productivity benchmarks.. 49
6.4 Measurement of project effort.. 49
6.5 Use of COSMIC sizing as the foundation metric for estimating project effort 50
6.7 Using measurements to improve organizational performance 50
6.8 Where to get more information on use of COSMIC sizing in practice 51

8 EXERCISES.. 52

7.1 Questions.. 52
7.2 Mini Case Studies 58
7.2.1 The Branch Library System ('BLS').. 58
7.2.2 The Domestic Intruder (or Burglar) Alarm System. 59
7.3 Answers and discussions of the Questions of section 7.1... 60
7.4 Analysis and discussion of the Mini Case Studies of section 7.2 65
7.4.1 The Branch Library System 65
7.4.2 The Domestic Intruder (or Burglar) Alarm System. 68

REFERENCES.. 71

Background

BACKGROUND: THE VALUE OF MEASURING SOFTWARE SIZE

Why measure software size?

As in any other field of endeavour, size and scale matter. The larger the requirements for a new software system, the more expensive, risky, and difficult it will be to deliver. Similarly, the more an organization depends for its success on delivering software, the greater the challenges of managing project teams to deliver software on time, efficiently, and to acceptable quality.

So the ability to measure and understand the influence of size can be critical for the performance of a software-producing organization. Specifically, software size is a key parameter to measure and control the following tasks.

- **Improving organizational performance in software development**. Knowing the size of some delivered software and the effort to develop it, you can calculate the productivity and speed of the development process. (Productivity = size / effort; Speed = size / duration.) And a count of defects discovered in a given time-period divided by size (i.e. ‘defect density’) is a valuable indicator of product quality.

  The more performance data you gather, the more your organization can learn about the factors that influence performance favourably or unfavourably, which technologies are most productive, the possible trade-offs between effort, time and quality, etc., etc. That learning can be your foundation for improving performance.

- **Estimating the cost of new developments**. If you can estimate an approximate size of the software early in its life and you know the productivity typically achieved by previous developments of similar software (otherwise known as ‘benchmark’ productivity), you can make a first estimate of the effort for the new development:

  Estimated effort for new development = (Estimated size of new software) divided by (Productivity of previous developments).

  Estimated development effort then becomes the main input to estimated development cost, cost/benefit analysis, budgeting, development planning, resource allocation, etc.

- **Controlling software development**. If you can track the size of software under development as its requirements are worked out in more detail, then you have the means to control ‘scope creep’ and so help maintain the development cost within budget.

- **Managing investments in software**. Evaluating the cost/benefit of a new investment, or deciding if and when it is economic to replace an existing system, etc., all require a good knowledge of system costs, and therefore of software sizes, the main cost-driver of software (re-)development.

A further benefit of measuring a COSMIC size early in the life of a new system is that **the process leads to improved quality of the software requirements** by helping identify ambiguities and inconsistencies, missing requirements, and suchlike. Users report that the insights gained from the measurement process lead to fewer product defects and hence lower development costs.

Obviously the cost of gathering and using size and other data must be weighed against the potential value from pursuing the goals listed above. But for estimating and controlling medium/large software projects, there is no substitute for having 'hard' data for decision-making.

Remember the old adage '*you cannot manage what you cannot measure*'.

Different ways of measuring software size

The size of a piece of software can be measured in many ways. For example, you can:

- count the source lines of code (SLOC) of the software programs. However, SLOC counts are technology-dependent and are not much help for early cost estimation as you can only know the size accurately after the software exists;
- use methods such as Use Case Points, Object Points, etc., but these methods are not standardized and the resulting sizes depend on the software design;
- use Agile Story Points, but these are subjective. In practice, it's often not clear if they measure size, effort or duration, and their meaning varies with the individual agile team.

Above all, none of these methods can help yield all the possible benefits from measuring software sizes that we are aiming for.

If you do aim to gain these benefits, then the only choice is to measure a size of the required functionality of the software. A 'functional size' is based only on software requirements and so is totally independent of the technology, processes and the individuals or teams used to develop the software. 'Functional Size Measurement' (FSM) methods have been around for decades, but there is only one '2nd Generation' FSM method - **the COSMIC method, the most powerful, generally-applicable, ISO-standard FSM method**.

What uniquely distinguishes the COSMIC method?

The COSMIC method is the only functional size measurement method

- designed according to fundamental software engineering principles, and hence applicable to:
  - business, real-time and infrastructure software,
  - software in any layer of a software architecture, at any level of decomposition down to its smallest components,
  - in summary, any type of software where knowing its size is valuable;
- able to size requirements from single User Stories up to the requirements for whole releases or systems, with rules to ensure sizing consistency at all levels of aggregation;
- with a continuous, open-ended size scale that conforms with measurement theory.
- that is completely 'open' with all documentation available for free download.

A consequence of the method being based on fundamental software engineering principles is that its design is actually very simple.

Essentially, if you can identify the underlying functional processes of the software and analyse them into four types of data movements (Entries, Exits, Reads and Writes), all as defined in this Guide, then you can measure COSMIC sizes.

1

INTRODUCTION

1.0 Chapter summary

This chapter defines:

- the various types of software which the COSMIC method is designed to measure;
- the two types of software requirements: 'Functional User Requirements' and 'Non-Functional Requirements', how they relate, and explains why the COSMIC method can measure the functionality arising from both types of requirements;
- the basic principles of the method, expressed in two models;
- the 3-phase COSMIC measurement process.

Note: this Guide uses the term 'project' for any set of software activities with defined goals. The term does not imply any particular development process or scale of activities.

1.1 Applicability of the COSMIC method

COSMIC is the only FSM method *designed* to measure the size of the following types of software, at any time in its life-cycle:

- application software that is required to manage business transactions and data;
- real-time software that is required to monitor, control and communicate about events and data, subject to timing constraints;
- hybrids of the above, and infrastructure software that supports applications, such as operating system components;

All software activities involve some degree of creativity. However, some software activities involve *purely and truly* creative tasks. Examples would be creating artistic works or video games, or developing complex mathematical algorithms. And the COSMIC method was not designed to measure a size of the functionality of such products.

Hence for a software development that involves partly creative and partly measurable work-output, leave aside effort on the creative activities and the associated functionality. Instead, focus on the functionality that can be reliably accounted for by its COSMIC size and the associated effort.

One general note of caution: successful use of software size measurements for e.g. effort estimation depends on having reasonably repeatable development processes where size is the main driver of effort.

A consequence of this is that if the requirements are mainly implemented by COTS or by an application package, it may be best to estimate effort using an approach that is specific to the package. However, the actual productivity of the project that implemented the package can still be determined by measuring the functional size of the implemented software. Other performance metrics can then be derived as usual.

1.2 Functional User Requirements

The size of a piece of software in units of 'COSMIC Function Points' is a measure of the amount of functionality that the software provides to its users.

Ideally, this functionality would be defined in the software's 'Functional User Requirements' (or 'FUR'). Although a complete unambiguous statement of FUR rarely exists physically in the real-world, we can still define the concept of FUR for the purposes of measurement.

DEFINITION – Functional User Requirements
A sub-set of the user requirements. Requirements that describe what the software shall do, in terms of tasks and services.

The key to understanding this definition is to distinguish requirements for ***what*** the software must do, from requirements for ***how*** the software should do it***.*** (The ISO standard version of the definition of FUR gives requirements for data transfer, transformation, storage and retrieval as examples of FUR.)

EXAMPLE STATEMENTS OF FUR: 'As a customer I want to make on-line payments to my suppliers from my current account.' 'The software shall control the fuel supply to the engine.' 'The budgeting system shall use a table of actual foreign exchange rates at December 1st'.

EXAMPLE STATEMENTS OF REQUIREMENTS THAT ARE NOT FUR: 'The software shall be written in Java.' 'The response time shall be less than one second averaged over the peak hour.' 'The software shall be ready for roll-out by the end of the year.'

1.2.1 Extracting the COSMIC concepts from the software artefacts [2]

As noted above, a complete, unambiguous statement of FUR is rarely available for measurement. Usually therefore, the concepts needed for a COSMIC size measurement must be derived from whatever software artefacts happen to be available for the measurer.

These 'concepts' are *the concepts that are used to define the COSMIC method's principles*, as described in section 1.3.

The 'artefacts' can be any of the various manifestations of software that may exist at different times in its life-cycle, e.g.

- requirements artefacts in natural language or in formal languages such as EARS, entity/relationship diagrams, state-transition diagrams, User Stories, etc.
- design artefacts, e.g. Use Case diagrams, OO designs, database designs, MVD models, etc.
- deliverable or operational artefacts, e.g. screen and report layouts, user documentation, test cases, database definitions, program code, etc.

There are basically two starting points for identifying these concepts in practice:

- If the software is in a state where requirements are being defined, e.g. in a formal language, a convention such as User Stories, or even free text, it should be very easy to identify the COSMIC concepts as the work of defining the requirements proceeds.

---

[2] The titles of this section and of the next section 1.2.2 differ from those in the MM, as the latter are rather misleading. Happily, you do not literally have to *"extract the FUR"* from the available software artefacts.

- If the software has progressed beyond requirements definition, whether still in design or even implemented years ago, it is often still easy to extract the COSMIC concepts by 'reverse-engineering' from whatever artefacts are available.

In the very early stages of development of a new piece of software, the requirements may not have been worked out in sufficient detail to measure the size precisely. However, you can still estimate an approximate size from the concepts that can be identified in the available artefacts. See Section 6.1 on approximate size measurement.

At any stage of development you may have to make assumptions about requirements that have not yet been defined in detail or that are simply missing or unclear. Document these assumptions so as to make clear the uncertainty in the measurement.

1.2.2 The process of deriving the COSMIC concepts from software artefacts

The process by which you derive COSMIC concepts from the available software artefacts obviously depends on the type of artefact. Given the huge variety of types of artefacts, this topic is generally beyond the scope of this Guide (and for the Measurement Manual).

However, provided you fully understand the concepts defined in the COSMIC method principles and the relationships between them, it is easy for a software professional to identify these concepts in real-world software artefacts. Section 3.6 at the end of Chapter 3 gives several examples of how to identify COSMIC concepts in various types of software artefacts.

If further help is needed, there are several Guidelines and Case Studies available from www.cosmic-sizing.org that show how to map from various data-analysis and requirements-determination methods, used in different domains, to the concepts of the COSMIC method [3], [4], [5], [6].

1.2.3 Non-Functional Requirements

Statements of requirements for software often contain a mixture of functional and non-functional requirements.

DEFINITION – Non-Functional Requirements (of software)
Any requirement for the software part of a hardware/software system or for a software product, except a functional user requirement for software. Non-functional requirements concern: • software quality attributes (e.g. availability, maintainability, usability, portability, response time, etc.;) • the environment in which the software must be implemented and which it must serve (e.g. the physical environment, the number of users, etc.;) • the processes and technology to be used to develop and maintain the software, and to execute the software (e.g. the programming language, the operational platform, etc.)

The COSMIC method does not attempt to size 'Non-Functional Requirements' (NFR), partly because it is impractical to define a common measurement scale for the huge variety of possible NFR [7], and partly because requirements that are initially expressed as 'Non-Functional Requirements' (NFR), often evolve, as a development progresses, wholly or partly into additional FUR for software.

A significant advantage of the COSMIC method is that it can be used to measure functionality regardless of whether the functionality was originally specified in statements of

FUR, or evolved as a result of examining NFR in more detail as the development progressed.

Figure 1.3 shows the evolution of functional size, and how size can be estimated, as a project progresses. (For simplicity, Figure 1.3 shows this evolution against a waterfall model of software development, but it could apply to any type of project.)

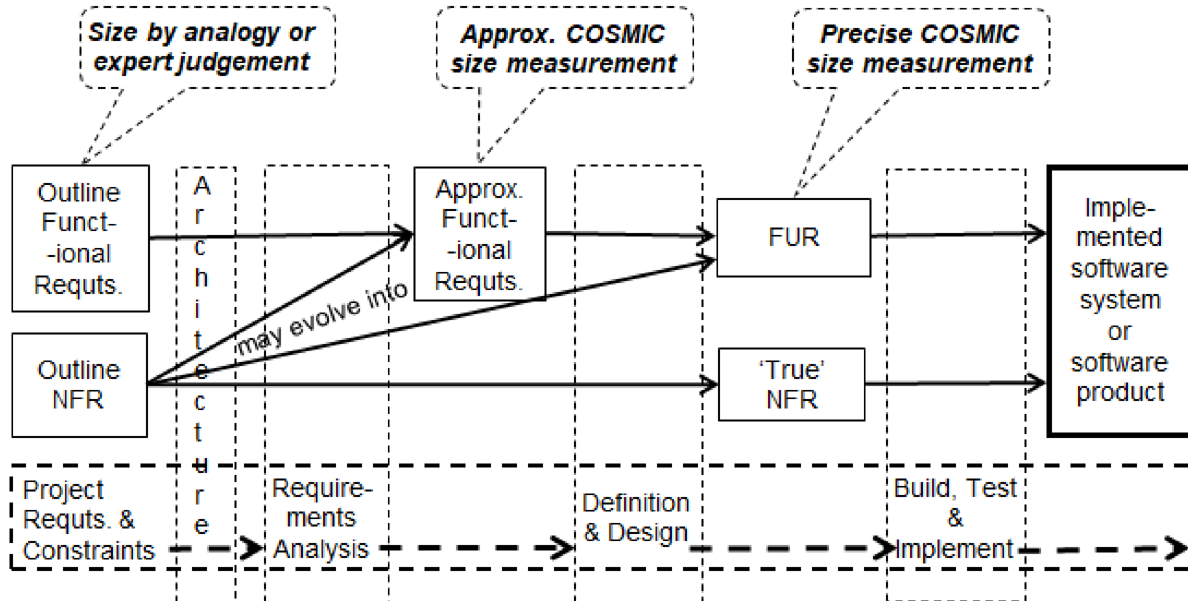


Figure 1.3 - Requirements expressed initially as NFR often evolve into FUR

EXAMPLE: System quality attributes such as for ease of use, maintainability and portability may evolve wholly into FUR for software. Other quality requirements, such as for a target system response time may evolve partly into FUR for specific software functions, and partly into 'true' NFR, for example for the technology to be used.

EXAMPLE: A quality NFR may state: 'The system shall be usable by members of the general public with a 99% successful completion rate'. This statement could evolve into FUR for the software to provide well-structured menus and comprehensive Help facilities. The target successful completion rate of 99% remains as a 'true' NFR for the system.

1.3 The fundamental principles of the COSMIC method

The fundamental software engineering principles on of the COSMIC method are expressed in two models.

The principles of the 'Software Context Model' (SCM) enable a measurer to define the software to be measured and the size to be measured.

The principles of the 'Generic Software Model' (GSM) define the concepts that must be identified in the artefacts of the software so that its functional size can be measured.

N.B. In the following, terms shown in bold when first used are key concepts of the COSMIC method. The references given with each principle are to the sections of this Guide (and of the MM) where the concepts are defined and discussed in detail.

1.3.1 The Software Context Model

PRINCIPLES – The Software Context Model (SCM)
• Software is typically structured into **layers** (2.2.2). • Any piece of software to be measured, shall be defined by its **scope**, which shall be confined wholly within a single layer (2.2). • The scope of a piece of software shall depend on the **purpose** of the measurement (2.1). • The **functional users** of a piece of software to be measured are the senders and/or intended recipients of data to/from the software respectively (2.3). • A piece of software interacts with its functional users across a **boundary**, and with **persistent storage** within this boundary (2.3). • A precise COSMIC size measurement of a piece of software requires that its Functional User Requirements are known at the **level of granularity** (2.4) at which the concepts of the Generic Software Model can be identified.

Figure 1.3.1 shows the relationships between the concepts of the Software Context Model.

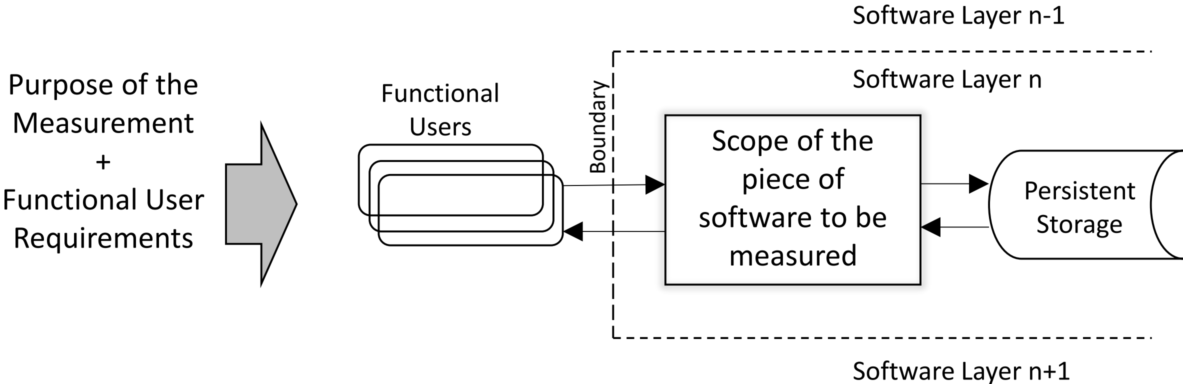


Figure 1.3.1. Relationships between the concepts of the Software Context Model

1.3.2 The Generic Software Model

PRINCIPLES – The Generic Software Model (GSM)
• Functional user requirements of a piece of software can be mapped into unique **functional processes.** (3.2) • Each functional process consists of sub-processes. (3.2) • A sub-process may be either a **data movement** or a data manipulation. (3.2) • As an approximation for measurement purposes, data manipulation sub-processes are not separately measured. The functionality of any data manipulation is assumed to be accounted for by the data movement with which it is associated. (3.5.6) • A data movement sub-process moves a single **data group.** (3.3)

- A data group consists of a unique set of **data attributes** that describe a single **object of interest.** (3.3)
- There are four data movement types, **Entry**, **Exit**, **Read** and **Write.** (3.5)
- The first data movement of a functional process is its **triggering Entry** which moves a data group generated by a functional user in response to a **triggering event.** (3.2)
- A functional process shall consist of the triggering Entry data movement and at least either a Write or an Exit data movement, i.e. it shall consist of a minimum of two data movements.(3.5)
- There is no upper limit to the number of data movements in a functional process. (3.5).
- The unit of measurement of the COSMIC method is one data movement, which has a size of one **COSMIC Function Point (or 'CFP')**. (4.2)
- The functional size of a functional process is equal to the total count of its data movements. (4.3)
- The functional size of a piece of software is equal to the sum of the sizes of its functional processes within the defined scope of the Functional Size Measurement. (4.3)

NOTE: The Generic Software Model is a logical model that describes units of Functional User Requirements from which the software's functional size can be measured. The model does not intend to describe the physical sequence of the steps in which software is executed, nor any technical implementation of the software.

Figure 1.3.2 shows the static relationships between some concepts defined by the principles of the Generic Software Model and the degree of their relationships. (Figure 3.2 shows the *dynamic* relationships between some of the concepts of the GSM.)

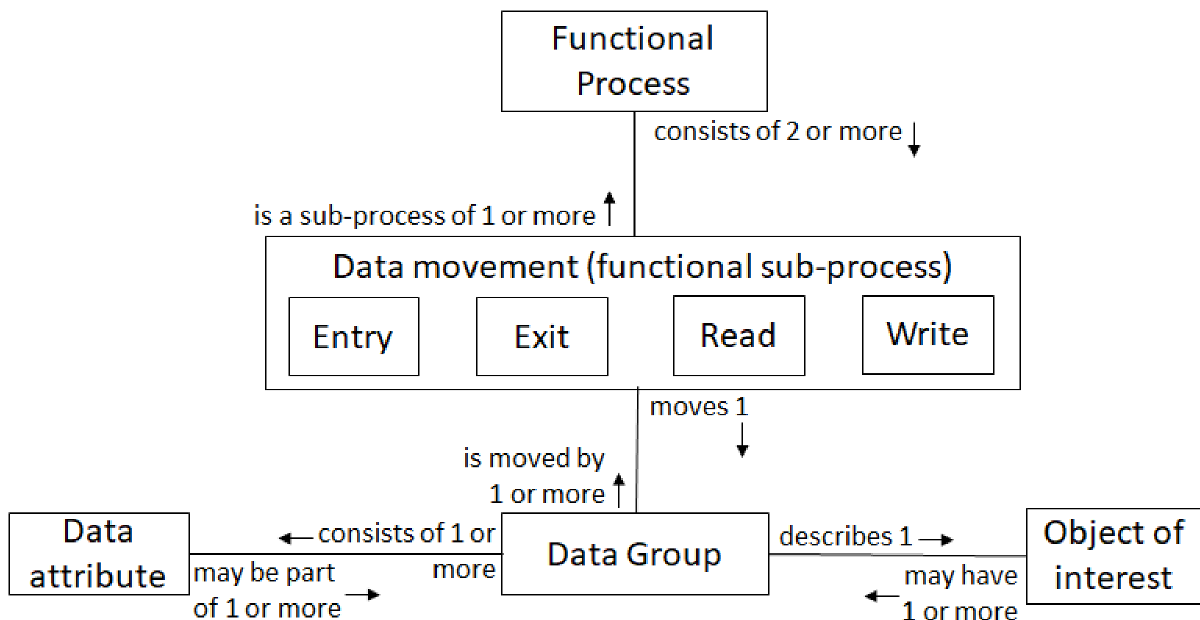


Figure 1.3.2. Static relationships between the concepts of the Generic Software Model

1.3.3 Types versus occurrences

All the concepts defined by the COSMIC method in its two models are types of things. Functional sizes are totally independent of the numbers of occurrences (or 'instances') of a concept that are required to be processed [3].

BUSINESS APPLICATION EXAMPLE: A company has 9,356 employees. A functional process (-type) to search the company's employee file for employees with a given characteristic needs only one Read (-type) for this purpose. When the process is executed, the Read will occur 9,356 times but that is of no interest for the functional size.

REAL-TIME SYSTEMS EXAMPLE: A paper-making machine uses 200 identical sensors to detect holes as the paper passes underneath. These sensor occurrences are functional users of the process control software, but they are all of one sensor-type.

1.4 The COSMIC measurement process

The COSMIC measurement process has three phases (see Figure 1.4):

- the Measurement Strategy phase, in which the Software Context Model is used to define the software to be measured, and the required measurement. (Chapter 2)
- the Mapping Phase in which the artefacts of the software to be measured are mapped to the Generic Software Model to identify the concepts that are needed for measurement. (Chapter 3)
- the Measurement Phase, in which sizes are measured. (Chapter 4)

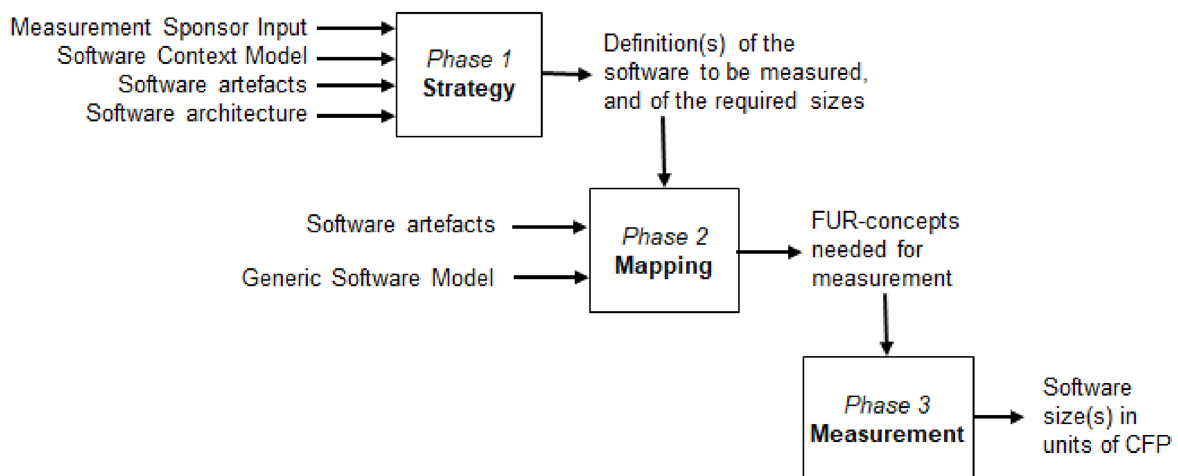


Figure 1.4 – The COSMIC method measurement process

1.5 Perceived limitations on the applicability of the COSMIC method

If you wish to account for some aspect of functional size in more detail than is accounted for by the standard COSMIC method, this can be done locally. (Example: you might want to account for the numbers of data attributes per data movement.)

See section 4.5 for how to do this without undermining the standard sizing method.

[3] Sometimes, the number of occurrences of a concept may be relevant to measuring a functional size. For example, section 3.3.2 defines rules for distinguishing different data movements of a given type (E, X, R or W) dependent on the degree of their relationship with other data movements of the same type (E, X, R or W), i.e. on their *relative* frequency of occurrence. But the *absolute number* of occurrences does not affect the measured size.

1.6 Some simple examples to introduce COSMIC size measurement in practice

The following examples illustrate the different types of requirements, first from the domain of business application software and the second from the real-time software. Concepts from the Software Context Model and from the Generic Software Model are shown in in **bold**.

EXAMPLE: PERSONNEL SYSTEM REQUIREMENTS:

Example Statements of Requirements	***Analysis using the concepts from the Software Context Model and the Generic Software Model***
The software shall maintain data about current and past permanent and temporary employees, their contact details, grades, ... (etc.).	*'Maintain' usually implies requirements for separate **functional processes**, arising from separate **triggering events** to create, update, read, and to delete data describing an **object of interest**, in this case 'Employee'. So this is a statement of FUR at a higher **level of granularity** than the concepts of the GSM.* *'Maintain' usually implies at least four functional processes.*
As a Personnel Officer, I wish to enter for each new employee their name, date of birth, home address, gender, marital status .... (etc.).	*This is a FUR for a single **functional process** ('Create Employee') arising from the **triggering event** of a new employee starting work. The Personnel Officer is the **functional user** who enters the **data group** ('Employee data') via one **Triggering Entry data movement**.*
Employee IDs shall be generated by the Personnel System	*This is a FUR for a part of a **functional process.** It could be implemented in various ways, e.g. generated by an algorithm (a data manipulation) or by a **Read data movement** of the next Employee ID from a list of available IDs, etc.*
Only Personnel Officers shall be able to access the Personnel System.	*This requirement is not clear. It could be one or more of:* • *a NFR (a quality requirement) to be implemented by a hardware security device;* • *FUR for additional functionality of the Personnel System;* • *FUR to use existing security software in another **layer** than the application layer.*

The following Sequence Diagram shows the minimum data movements for the functional process 'Create Employee' (the second requirement in the table above).

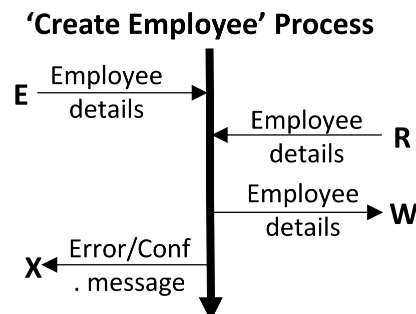


Figure 1.6.1. Sequence diagram for a simple 'Create Employee' functional process

This functional process requires an Entry to move the 'Employee details' data group entered by the Personnel Officer, a Read from persistent storage to check that the

employee data has not already been entered, a Write to make the entered data persistent, and an Error/Confirmation message as an Exit. The latter, by a COSMIC rule, accounts for all possible error messages in a functional process reported back to a human functional user, e.g. for validation failures, and for any confirmation message that the process has completed successfully.

The minimum total size of this functional process is therefore 4 COSMIC Function Points, i.e. ***4 CFP.***

EXAMPLE: REAL TIME SYSTEM REQUIREMENTS

Example Statements of Requirements	***Analysis using the concepts from the Software Context Model and the Generic Software Model***
The Home Control System ('HCS') shall control temperature and humidity in up to six zones of a house.	*This system requirement is at a very high* ***level of granularity****. The system's architecture must show the allocation of functionality between hardware and software. The number of separate* ***functional processes*** *of the software will then be determined by the number of* ***triggering events*** *that the software must respond to.* *The number 'six' is the number of occurrences of identical zone (-types), which is irrelevant for sizing the software.*
On receipt of a signal from the clock, the software shall compare the actual zone temperature against the pre-set target temperature. If the difference exceeds 1.0 degree C, it shall switch the heater on or off so as reduce the difference.	*This is the FUR for a single* ***functional process*** *to control the temperature of a zone, whose* ***triggering Entry*** *is a clock timer signal.* *The functional process has as its* ***functional users*** *the clock, the zone thermostat and the heater. The FUR do not specify whether the target temperature must be obtained by a* ***Read*** *of a pre-set temperature from* ***persistent storage*** *or by an* ***Entry*** *from a hardware functional user, e.g. the setting of a temperature dial.*
The clock shall issue its timing signal at one-minute intervals.	*If the clock is a hardware device, the one-minute interval is a NFR for sizing any HCS software. Or, this could be a FUR for part of a HCS* ***functional process*** *to issue the timing signal, taking input from its Operating System clock.*

The following Sequence Diagram shows the data movements for the functional process 'Control Temperature' (the second requirement in the table above).

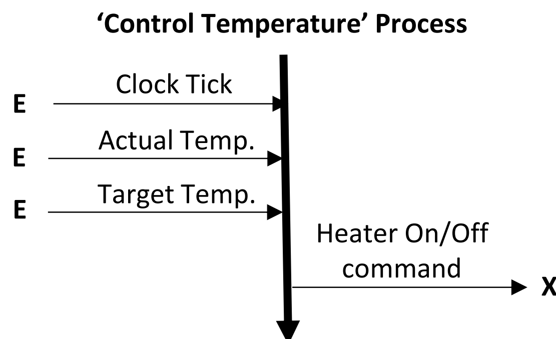


Figure 1.6.2. Sequence diagram for the 'Control Temperature' functional process

The analysis of this functional process assumes the target temperature is obtained as an Entry from a hardware functional user such as a temperature-setting dial. Note that the

*size of this functional process, **4 CFP**, is unchanged even if the target temperature were obtained via a Read from persistent storage.*

General comments on these two examples:

- Both of these examples are typical very simple processes in their respective domains. In practice, the sizes of functional processes can vary enormously across the many types of software. Single functional processes have been measured in banking applications at over 70 CFP and in avionics software at over 100 CFP. The COSMIC method design principles result in size measurements on a ratio scale, and there is no upper limit to the size of a functional process. (This contrasts with the design of '1st Generation' FSM methods and is very important for improving the accuracy of productivity measurement and of effort estimation.)
- The size of the two example functional processes is the same (4 CFP) even though many more data attributes must be moved in the Personnel System functional process than in the Control Temperature process. This indicates that comparisons of functional sizes, and therefore derived productivity measurements, across software from different domains must be treated with caution. This very rarely matters in practice. For example, it is unlikely that anyone wants to compare the productivity of, say, teams working on developing retail banking applications against the performance of teams developing the embedded software of programmable logic controllers.
- The examples illustrate that to measure a COSMIC functional size of some software, you must identify only its functional processes and their data movements (Entries, Exits, Reads and Writes).

  So although, you need to understand the other COSMIC concepts in order to be sure you have correctly identified the functional processes and their data movements, fundamentally the COSMIC method for measuring a functional size is very simple.

2

THE MEASUREMENT STRATEGY PHASE – WHAT SOFTWARE IS TO BE MEASURED AND WHY?

2.0 Chapter summary

The aim of the work in this phase is to agree certain parameters with the Sponsor of the measurement before actually starting to measure. These are principally:

- the **purpose** of the measurement,
- the **scope** of each piece of software to be measured, which means defining the **layer(s)** in which the software is located and its **level of decomposition**,
- the software's **functional users**.

The parameters should be recorded so that the size measurement can always be correctly interpreted in the future.

Figure 2.0 shows the steps of determining a Measurement Strategy.

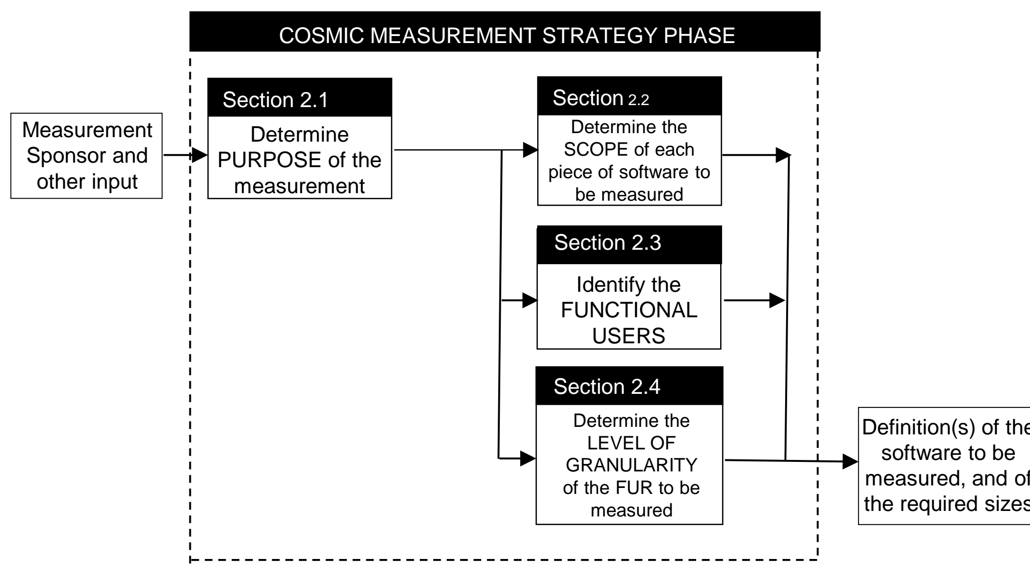


Figure 2.0 - The steps for determining a Measurement Strategy

2.1 Defining the purpose of a measurement

DEFINITION – Purpose of a measurement
A statement that defines why a measurement is required, and what the result will be used for.

The first step of the strategy is to determine what the Sponsor needs the measurement for, i.e. the purpose of the measurement. The purpose then determines the scope to be measured (see section 2.2), when it is required, and then the other strategy parameters.

For example, the purpose could vary from estimating an approximate size early in the life of some new software so as to estimate its development cost, through to measuring an accurate size of the delivered software in order to be able to pay the supplier.

2.2 Defining the scope of a measurement

DEFINITION – Scope of a measurement
The extent of the Functional User Requirements of the software to be accounted for when measuring a functional size.

2.2.1 Deriving the measurement scope(s) from the measurement purpose

Most software is built nowadays in an architecture of defined layers.

So if all the software to be measured resides in different layers of an architecture, first define a separate scope for each piece of software in each separate layer (due to the second principle of the Software Context Model),

You may then need to sub-divide the software in any one layer into separate pieces, each with their own measurement scope, for purposes such as performance measurement or effort estimation. This may be necessary for example because the different pieces:

- are developed using different technologies, e.g. hardware platform, programming language, etc.,
- execute in different modes, i.e. on-line versus batch modes,
- are developed as opposed to ‘delivered’ (the latter including package-implemented or other re-used software),
- are at different ‘levels of decomposition’ (as defined in 2.2.3), e.g. a whole application or a major component, or a minor component such as a re-usable object,
- include software that is used once for data conversion, and then discarded, so does not contribute to the size of the ‘main deliverable’.

2.2.2 Layers

DEFINITION – Layer
A functional partition of a software system architecture.

The layers of a software architecture may differ depending on the ‘view’ of the architecture. And it is the purpose of the measurement that determines which view should be used.

EXAMPLE: Consider an application A, as in Figure 2.2, which shows three possible layer structures a), b) and c) according to different ‘views’ of its architecture.

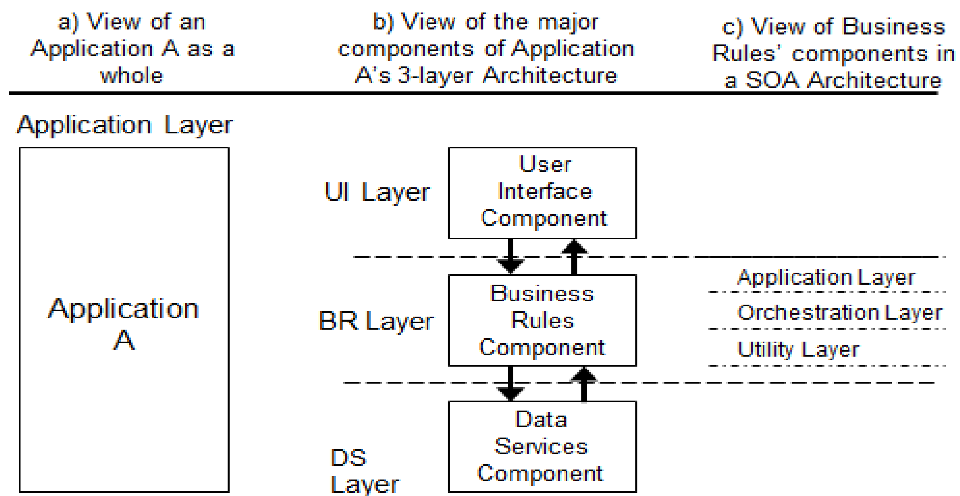


Figure 2.2 - Three views of the architecture of an application

View a) shows application A which exists entirely in its Application Layer. Purpose 1 is to measure the functional size of application A 'as a whole'. The measurement scope is the whole of application A.

View b) shows that application A has been built according to a 'three-layer' architecture comprising three major components: User Interface, Business Rules and Data Services, Purpose 2 is to measure each component separately. So each component has its own measurement scope.

View c) shows that the Business Rules component has been built from re-usable components in a Service-Oriented Architecture, which has its own layered structure. Purpose 3 is to measure the SOA components of the Business Rules component separately. Each SOA component has its own measurement scope. (Note that SOA terminology also uses 'application layer' within its own architecture.)

See the MM for other examples of typical layered architectures. In addition, for cases where the layers of the software to be measured are not clear, the MM provides guidance on how to distinguish layers for COSMIC size measurement purposes.

2.2.3 Levels of decomposition

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: Do not confuse the 'level of decomposition' of software with the 'level of granularity' (or the *level of detail*) of software requirements – see section 2.4.. NOTE 2: Size measurements of pieces of software are only directly comparable for pieces at the same level of decomposition.

Note 2 of this definition is important because sizes of pieces of software at different levels of decomposition cannot simply be added up without taking into account the aggregation rules as we shall see in section 4.3.1.

2.3 Identifying the functional users and recognizing persistent storage

DEFINITION – Functional user
A (type of) user that is identified in the Functional User Requirements of a piece of software being measured as a sender and/or an intended recipient of data processed by that software.

Functional user types usually depend on the software domain.

- In the domain of business application software, the functional users are normally humans, plus perhaps other applications or software components with which the application interfaces.
- For real-time software, the functional users would normally be engineered hardware devices that interact directly with the software, plus perhaps other interfacing software.

Remember that the COSMIC method measures sizes based on a *logical* model of the interactions of functional users with the software being measured.

In these logical models, we can show a human functional user sending and receiving data to/from the software being measured because we can ignore the intervening hardware keyboards and screens and the operating system software that *physically enable* the interactions. Similarly, a logical model can show two pieces of software interacting with each other, ignoring the physical reality that the two pieces may execute on different computers and exchange messages over a communications network.

Using the name 'Functional User' means we can now interpret the term 'FUR' as the *functional users' requirements for the functions they want the software to perform*.

Further, the 'Views' shown in the layers of Figure 2.2 can now be seen as defining the functionality available to the functional users of each piece of software of defined scope.

2.3.1 Functional size may vary with the choice of functional users

The choice of functional users depends on the purpose and scope of the measurement. Two examples illustrate the importance of the choice.

> *Example 1: The size of a mobile phone app may be measured in two ways depending on the Purpose of the measurement, namely either a) the size of the functions provided for its human functional users, or b) the size of all the functions that the app must provide to its immediate hardware/software functional users, i.e. the objects on the screen, other interfacing apps, the phone's operating system etc. The human user is not aware of all these technical functional users that the app developer must deal with*

> *Example 2. Application software interacts with many parts of an operating system (OS). But the FUR of an application never normally include interactions with the OS because these are common to all applications. The OS is therefore never normally considered as a functional user of an application. (In fact, an application is a functional user of the OS and its interactions with the OS are usually established by the compiler or interpreter.)*

2.3.2 Persistent storage and the boundary

DEFINITION – Persistent storage
Storage which enables a functional process to store a data group beyond the life of the functional process and/or from which a functional process can retrieve a data group.

(For the definition of a 'functional process', see section 3.2).

Persistent storage is a logical concept of the Software Context Model, not to be confused with any physical storage device. It does not need to be identified because it is available to software in any layer, and it exists within the 'boundary' of all software being measured.

You do not have to consider how data are logically or physically stored when measuring a CFP size.

DEFINITION – Boundary
A conceptual interface between the software being measured and its functional users. NOTE: It follows from the definition that there is a boundary between any two pieces of software in the same or different layers that exchange data.

Note: Do not confuse the *boundary* with any line that you might draw around some software to define the *scope* of the measurement.

2.3.3 Context diagrams

It can be very helpful when defining a measurement strategy to draw a 'context diagram' showing the scope of each piece of software to be measured within its context of functional users, plus the movements of data between them and persistent storage if relevant.

A context diagram is effectively an instance of a measurement pattern (see section 2.5). For examples see Figures 2.5.2 and in section 7.4.1.

2.4 Identifying the level of granularity of Functional User Requirements (FUR)

2.4.1 The need for a standard level of granularity

DEFINITION – Level of granularity
Any level of expansion of the description of any part of a piece of software (e.g. a statement of its requirements, or a description of the structure of the software) such that at each increased level of expansion, the description of the piece of software is at an increased and uniform level of detail. NOTE: Early in the life of a software project when requirements are evolving, at any moment different parts of the software FUR will typically have been defined at different levels of granularity.

If the task is to measure some software before it is implemented – anytime from the early stages of elicitatng requirements through to design – we may be faced with FUR or other artefacts at different levels of granularity (some at a high level, some in detail, some only guessable). We must therefore define a standard level of granularity to ensure we can measure consistent functional sizes across all parts of a software system.

The only level that can be unambiguously defined as a standard level is the 'Functional Process Level of Granularity' - see section 2.4.3.

2.4.2 Clarification of 'level of granularity'

As the FUR for a piece of software are worked out in more detail, their *description* moves from a 'higher' to a 'lower' level of granularity, *without changing the measurement scope*. This process of evolving the FUR should NOT be confused with any of the following.

- Examining some software in order to reveal its components, sub-components, etc. at different 'levels of decomposition' – see section 2.2.3.

- Evolving the description of software as it progresses through its development cycle, e.g. from requirements to logical design, to physical design, etc. We are only interested in measuring the software FUR, regardless of the actual stage in its development.

2.4.3 The standard functional process level of granularity

DEFINITION - Functional process level of granularity
The level of granularity of the FUR of a piece of software at which: • its 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 of granularity at which groups of events are defined).

The ‘functional process level of granularity’ is critically important because only at this level of granularity of FUR can we be certain to correctly identify the concepts that are needed to measure standard CFP sizes. For more on this see the next Mapping Phase, especially section 3.2.

It is also important to be able to recognize the functional process level of granularity when measuring an approximate size form outline FUR using a variant of the standard COSMIC method – see section 6.1.

2.5 Measurement Strategy Patterns

(This section of the Guide appears in section 2.0 of the Measurement Manual, v4.0.2.)

DEFINITION – Measurement (Strategy) Pattern
A standard template that may be applied when measuring software from a given software functional domain, that defines the types of functional user that may interact with the software, the level of decomposition of the software and the types of data movements that the software may handle.

Within an organization, the Measurement Strategy parameters are likely to be the same for many pieces of software, so it is not usually necessary to have to repeat all the Strategy steps for every measurement. If a standard Pattern can be applied for measuring several pieces of software, the same Strategy parameters can be re-used for all the measurements.

A ‘Guideline for Measurement Strategy Patterns' [9] describes, for several different types of software, a standard set of parameters for measuring software sizes.

EXAMPLE: Figure 2.5.1 shows a typical Measurement Strategy pattern for a real-time application, showing its various possible types of functional users. Figure 2.5.2 shows the context diagram for an intruder (or burglar) alarm - a specific instance of the pattern shown in Figure 2.5.1. (The specification for the intruder alarm is given in a mini case-study exercise in section 7.2.)

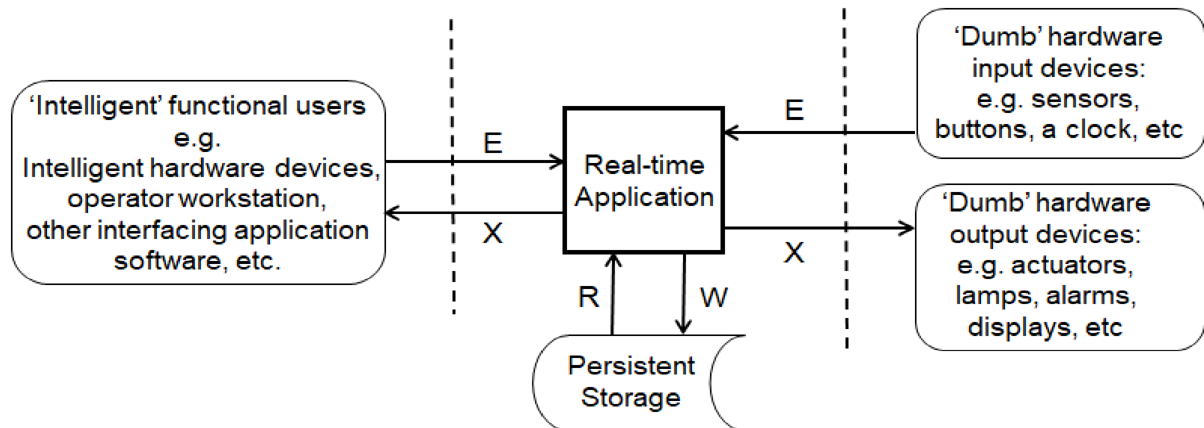


Figure 2.5.1. A general measurement pattern for a real-time application

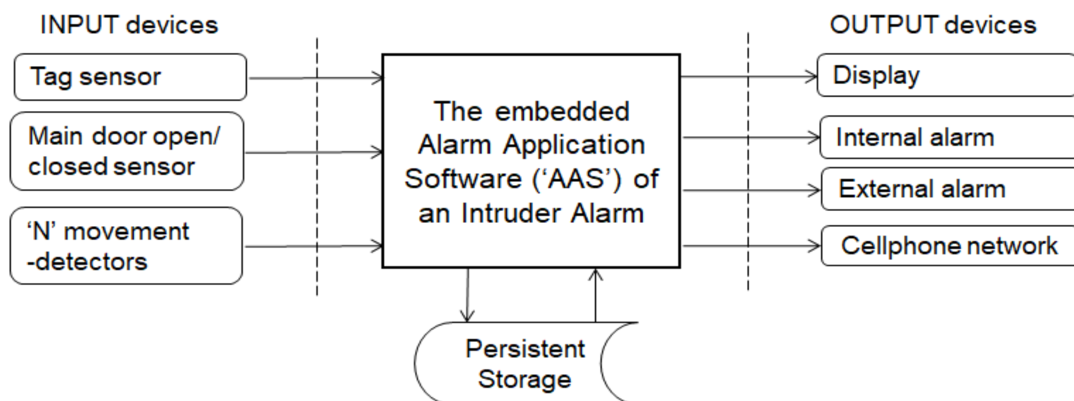


Figure 2.5.2. The context diagram for the application software of an intruder alarm

2.6 Concluding remarks on the Measurement Strategy Phase

As illustrated by the discussion in section 2.3.1, a piece of software can have more than one size in units of COSMIC Function Points. And its size may be measurable accurately or only approximately.

Defining the Measurement Strategy parameters may sound like a significant overhead before starting to measure, but it is the only way to ensure that the size to be measured will serve its intended use. And it is important to document the Measurement Strategy parameters so that the resulting size measurements can be correctly understood and used in the future.

In practice, the overhead is usually minimal, since in any one organization, the same Measurement Strategy patterns will re-occur for many measurements.

3

THE MAPPING PHASE

3.0 Chapter summary

In the Mapping phase of the measurement process, the aim is to identify the **functional processes** and the **data movements** from the available artefacts of each piece of software whose scope was defined in the previous Strategy phase.

The definitions, rules and examples of this chapter are necessary to help you identify these two concepts consistently and reliably from real-world software artefacts However, when reading these details, it is important not to lose sight of the 'big picture', namely the Generic Software Model.

The principles of the GSM are like the pieces of a jigsaw puzzle. They all fit perfectly together, to build the beautifully-simple COSMIC model of the software you are measuring. When you have really grasped this model, you will understand why we claim that COSMIC size measurement is so easy. We therefore suggest that you first re-read the GSM principles (section 1.3.2) and keep these in mind as you work through this chapter. The figures in sections 1.3.2, 3.2, and 3.5 are also very helpful by showing the relationships between the key concepts of the GSM.

3.1 Mapping from the software artefacts to the concepts of the Generic Software Model

The process to identify the functional processes and their data movements is shown in Figure 3.0 as a sequence of steps, though in practice you will probably identify them in parallel or iteratively, depending on the available artefacts.

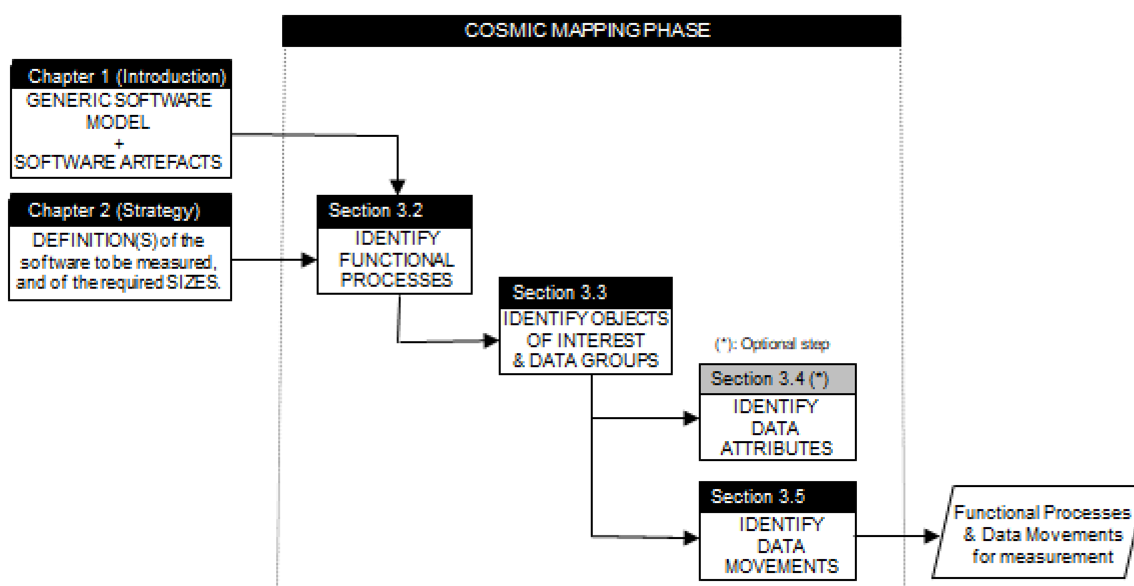


Figure 3.0 – The steps of the Mapping Phase

3.2 Identifying functional processes

The functional processes of software are designed to respond to the triggering events that occur in the world of its functional users. So to identify the functional processes of the software to be measured, it helps to mentally construct the sequence of steps linking a triggering event to the start of a functional process as shown in Figure 3.2.

This is one of the most important diagrams to understand if you aim to master the COSMIC method.

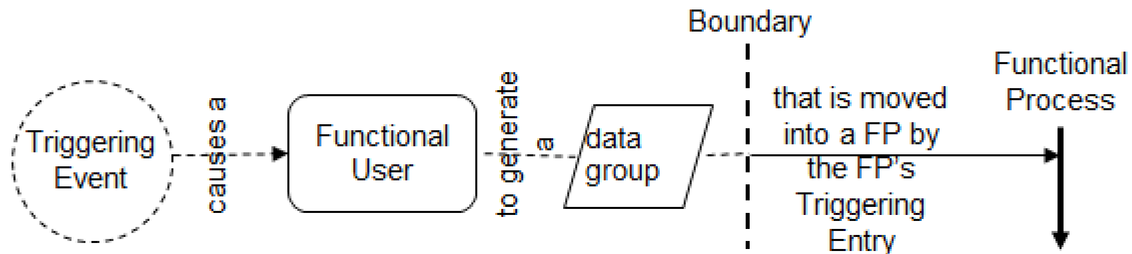


Figure 3.2 – The dynamic relationships between a triggering event, a functional user and a functional process

We have already defined a functional user (as an 'intended sender and/or recipient of data ...'). In this section, we define a triggering event, a triggering Entry, and a functional process. In section 3.3 we will define data groups and objects of interest, and in section 3.4 data movements.

3.2.1 Definitions

DEFINITION – Triggering event
An event (something that happens), recognized in the Functional User Requirements (FUR) of the software being measured, that causes a functional user of the software to generate (i.e. create) one or more data groups. NOTE 1. In a given statement of FUR, a triggering event cannot be sub-divided; it has either happened or not happened. NOTE 2: Clock and timing events can be triggering events.

DEFINITION – Triggering Entry
The first Entry data movement of a functional process. This Entry moves the data group generated by a functional user that is needed by the functional process to start processing.

DEFINITION – Functional process
A unique set of data movements of a piece of software to be measured that is needed to meet its FUR for all the possible responses to the data entered as a consequence of a triggering event. NOTE 1: The FUR for a functional process may require one or more other Entries in addition to the triggering Entry. NOTE 2: Two or more functional processes within the same FUR may be unique, even though they share some common functionality.

The table below gives some simple examples for Figure 3.2.

You will see that business application software may be required to capture data for triggering events that have already happened (example a)) or for the triggering event of a human functional user deciding 'I want to enquire upon ...' (example b)). Real-time software is required to respond to triggering events within strict time-constraints (examples c) and d)).

Triggering Event	***Functional User***	***Data Group Generated***	***Functional Process Triggered***
a) New employee starts work	*Personnel Officer*	*New employee details*	*'Create new employee'*
b) A PO thinks: 'I want to enquire on ...'	*Personnel Officer*	*Employee ID or name*	*'Enquire on employee details'*
c) Pre-set time interval	*Clock*	*A 'tick'*	*A control process cycle*
d) Missile approaching	*Aircraft radar*	*'Missile approaching' message*	*A process to start taking evasive action*

Remember that the Generic Software Model is a logical model. Physically, a functional process may start its processing before any data has been entered e.g. when a human user clicks on a menu to select the process and to display a screen for data entry. But logically a functional process only starts when its triggering Entry sub-process has received a data group. We are not interested in the physical implementation.

The degree of the relationships between the triggering event, the functional user and the data group in Figure 3.2 may be one-to-many, many-to-one, or many-to-many. **However, any one functional process has only one triggering Entry that moves one data group.** For a fuller discussion of the possible cardinalities along the chain of Figure 3.2 and for more examples, see Appendix C of the MM.

If you can apply the model of Figure 3.2 to the FUR or other artefacts that must be measured, then you can also be confident that you are correctly measuring at the 'Functional Process Level of Granularity' (see section 2.4.3).

3.2.2 The approach to identifying functional processes

The approach to identifying functional processes depends on the software domain and on the software artefacts that are available to the measurer, which in turn depend on the point in the software lifecycle when the measurement is required.

For real-time software, it usually helps to identify the functional processes by following the chain of Figure 3.2, i.e. first identify the triggering events in the FUR. For example, state transition diagrams may indicate the events that lead to the triggering of a functional process.

For business application software it is often easier to first identify the objects of interest (or 'entities' or 'data subjects', see section 3.3) for each of which a data group must be entered and stored, and to remember the 'CRUD' acronym. This is because each object of interest usually requires separate functional processes to 'Create' data about it, to 'Read' (or enquire, or report on) the data describing the object of interest in various ways, to 'Update' the data one or more times in response to various events, and to 'Delete' the data at some stage.

Use the following rules to validate candidate functional processes.

RULES – Functional process
a) A functional process exists entirely within the scope of one piece of

software, in one layer.

b) An executing functional process terminates when it has satisfied its FUR for all the possible responses to the data moved by its Entries.

[Note: This rule b) in the MM ends with the phrase ‘responses ... to its triggering Entry’. This is too restrictive. Data in other Entries may also require particular responses that involve more data movements than result from just the triggering Entry. These other data movements must also be measured in the size of the functional process.]

3.2.3 Triggering events and functional processes of business applications

The triggering events that business application software must respond to may be:

- single physical events, e.g. to record that an employee’s address has changed,
- or a single decision-event e.g. ‘I want to enquire on my order-status’, or ‘I want to open a bank account’,
- or a class of events, e.g. a general-purpose update process to handle a variety of events corresponding to real-world changes, or a general-purpose enquiry tool.

EXAMPLE: The FUR for a system to maintain basic employee data may specify many separate Update functional processes to respond to separate events for an employee such as a change of marital status, change of address, change of grade, to add a new educational qualification, etc. Alternatively, the FUR for a simple system may specify only one Update functional process for recording changes to an employee’s details arising from all possible change-events. (There is nothing absolute about the choice of events and hence of the functional processes in business application software. The choice depends on the FUR.)

EXAMPLE: A Personnel Officer wanting to update an employee’s details will probably first make an enquiry to display the employee details to check that the correct employee has been selected. The enquiry and the update are separate functional processes because they require separate decisions (triggering events) from the user.

See the MM and section 1.6 below for many more examples.

NOTE: There is no difference in principle to the analysis of a functional process whether it is required to be processed in real-time, on-line or in batch mode. A requirement for how some input data should be processed (e.g. subject to a timing constraint, or batch-processed) is a non-functional requirement (NFR). See the MM for more on this point.

3.2.4 Triggering events and functional processes of real-time applications

Real-time application software must normally respond to real-world physical events, so identifying the events that can occur is critically important.

EXAMPLE: *When a bar-code reader (a functional user) of a supermarket checkout system senses that a bar-code has appeared in its window (a triggering event), this starts a functional process of the checkout software. The process takes the scanned the bar-code image as the data group moved by its triggering Entry. The functional process checks the bar-code, sounds a ‘beep’ if the code is valid, obtains the product cost and adds the cost to the customer’s bill, logs the sale, etc.*

See the MM for more examples.

3.2.5 More on separate functional processes

According to the definition of a funcitonal process and rule b) in section 3.2.2, the set of data movements of a functional process must satisfy ‘its FUR for all the possible responses to the

data moved by its Entries'. This means that the same one functional process *type* must be able to deal with all possible *occurrences* of *values* of the data attributes of the data groups moved by its Entries, including both valid, invalid, and missing data values. These variations in the values of the entered data may result in different processing paths being followed within the functional process when it executes. But there is still only the one functional process *type*, and its size depends only the total number of its data movement *types*. The number of processing paths that may occur is irrelevant to the measurement.

See the MM for examples.

3.2.6 Measuring the components of a distributed software system

When the purpose is to measure the size of each component of a distributed software system, a separate measurement scope must be defined for each component. Each component is then a functional user of any other component with which it exchanges data. Each component has its own functional processes and these are identified following the normal sequence of steps as in Figure 3.2. A functional process cannot exist partly in one scope and partly in another scope.

> *EXAMPLE: Consider the three components of the distributed software system shown in 'View b)' from Figure 2.2. Suppose a functional process 'A' of the User Interface component must obtain a service from the Business Rules component. The UI component is then the functional user of the BR component.*
>
> *When the process 'A' sends its request to the BR component, this request is the triggering event for a functional process 'B' of the BR component that must service the request. The request message is the data group that is moved by the triggering Entry of process 'B'. This process 'B' will then return its reply to the process 'A', assuming synchronous communications, or to another process 'C' of the UI component if the communication is asynchronous.*

See the MM for more details.

3.2.7 Independence of functional processes sharing some common functionality

Two or more functional processes in the same software may require some functionality that is identical or very similar in each process. However, each functional process must be analysed and measured independently. Any functionality that is common to any two or more functional processes in the same software must be accounted for in the size of each of these processes.

See the MM for more details and examples.

3.2.8 Events that trigger a software system to start executing

When measuring the size of a piece of software, identify only the events and corresponding triggering Entries that trigger the functional processes that the software must respond to as defined in its FUR. Functionality needed to start-up (or 'launch') the software itself is not part of these functional processes and should be ignored (or measured separately, if required).

See the MM for more details and examples.

3.3 Identifying objects of interest and data groups

Having identified the functional processes of the software to be measured, our next goal is to identify its sub-processes, i.e. its data movements.

However, recall the fifth and sixth Principles of the Generic Software Model, namely:

- A data movement sub-process moves a single **data group.**
- A data group consists of a unique set of **data attributes** that describe a single **object of interest.**

We must therefore first define these other three concepts. (Note: as defined in the COSMIC method, a 'data group' moved by a data movement is not just any arbitrary grouping of attributes. Figure 1.3.2 also shows the relationships between these three concepts.)

3.3.1 Definitions

DEFINITION – Object of interest
Any 'thing' in the world of the functional user that is identifiable in the Functional User Requirements of software, that is the subject of one or more data groups moved by the software. It may be any physical or any conceptual thing. NOTE 1: A synonym for 'object of interest' is an 'entity-type' or a 'data subject'. The term does not imply 'an object' in the sense used in Object-Oriented methods. NOTE 2: When a functional user sends a data group about itself, e.g. its state or its identity, or when a functional user receives data concerning itself, e.g. an instruction to do something, then the functional user is also the object of interest of the data group moved.

DEFINITION – Data group
A distinct set of data attributes where each data attribute describes a complementary aspect of the same one object of interest. NOTE: The term 'data group' does not necessarily mean 'the set of *all* data attributes that describe an object of interest'. Different groups of data attributes, all describing the same object of interest, may need to be formed by different movements of a functional process and by different functional processes.

DEFINITION – Data attribute
The smallest parcel of information, within an identified data group, carrying a meaning from the perspective of the software's Functional User Requirements. NOTE: A synonym for 'Data Attribute' is 'Data Element'.

3.3.2 About the identification of objects of interest and data groups

When analyzing the data attributes input, output, stored and retrieved by a functional process, *it is critically important* to group the attributes so that that each group conveys data about a *single* object of interest. The one-to-one relationship between a single object of interest and a data movement therefore ultimately determines the number of data movements of the process.

See the MM for a fuller discussion.

The following rule helps the identification of data groups and hence objects of interest.

RULE - Identifying different data groups (and hence different objects of

interest) moved in the same one functional process
For all the data attributes appearing in the input of a functional process: a) sets of data attributes that have different frequencies of occurrence describe different objects of interest; b) sets of data attributes that have the same frequency of occurrence but different identifying key attribute(s) describe different objects of interest. This same rule applies for all the data attributes appearing in the output of a functional process, and for all that are moved by a functional process to or from persistent storage. See the MM for cases where the FUR may specify exceptions to this rule.

Analysis of data groups, in input, in storage, or in output

'Data models', built using e.g. entity-relationship analysis or relational data analysis, and often developed during the analysis and design of business application software, are valuable sources for identifying objects of interest for *persistently* stored data.

However, the same data modelling techniques can be used to identify the data groups and hence objects of interest that appear in the input (all the Entries) and the output (all the Exits) of a functional process.

Objects of interest and data groups in the business applications domain

EXAMPLE: In business application software, an object of interest could be 'employee' (physical), or 'order' (conceptual). In the case of 'order', the FUR may specify multi-line orders, indicating there are two objects of interest: 'order (-header)' and 'order-line'.

EXAMPLE: Suppose the FUR for an enquiry functional process against an employee file (i.e. persistent data) to list the names of employees older than a given age (which must be input) and the total number of such employees. The input and output data groups are all transient, i.e. they exist only in this process; they are analysed as shown below.

	Data group	***Object of interest of the data group***
Input	*Given age limit*	*The set of employees older than the given age limit*
Output	*Employee name*	*Employee older than the given age limit*
	Total number of employees	*The set of employees older than the given age limit*

Note that a 'set' (e.g. all employees) and a 'member of a set' (e.g. an individual employee) are always different 'things', have different frequencies of occurrence and so must be different objects of interest.

See the MM for a fuller discussion and for several examples.

Objects of interest and data groups in the real-time software domain

EXAMPLE: A data group sent by a functional user that is a physical device to a functional process may inform the process about the state of an object of interest, e.g. that a valve is open or closed. (In such a case, the functional user has sent data about itself, so the functional user is also the object of interest of the sent data group.)

Similarly a data group output to a functional user that is a physical device, such as a command to switch a warning lamp on or off, conveys data about the lamp object of interest.

See section 3.3.4 and the MM for a fuller discussion and for several examples.

3.3.3 Data or groups of data that are not candidates for data movements

Any data appearing on input or output screens or reports that are not related to an object of interest to a functional user should not be identified as indicating a data group, so should not be measured.

See further in section 3.5.10.

3.3.4 The functional user as object of interest

As per NOTE 2 of the definition of an object of interest, a functional user of the software being measured may also be the object or interest of a data group that it sends or receives concerning itself.

See the MM for a fuller discussion and for more examples.

3.4 Identifying data attributes (optional)

It is not mandatory to identify the data attributes in a data group. However, understanding the concept of a data attribute is necessary in order to understand the measurement of required changes to software, such as a change to a data attribute (see sections 4.3 and 4.4).

Also, it may be helpful to analyze and identify data attributes in the process of distinguishing data groups and objects of interest.

See the MM for a fuller discussion and for several examples.

3.5 Definitions and Principles for data movements

DEFINITION – Data movement
A functional sub-process which moves a single data group.

3.5.1 Definitions of the data movement types

There are four sub-types of a data movement namely: Entry, Exit, Read and Write.

DEFINITIONS – Entry (E), Exit (X), Read (R) and Write (W)
An Entry is a data movement that moves a data group from a functional user across the boundary into the functional process where it is required. An Exit is data movement that moves a data group from a functional process across the boundary to the functional user that requires it. A Read is a data movement that moves a data group from persistent storage into the functional process which requires it. A Write is a data movement that moves a data group from inside a functional process to persistent storage.

Figure 3.5 illustrates the relationships between the four sub-types of data movement, the functional users of the measured software, and persistent storage.

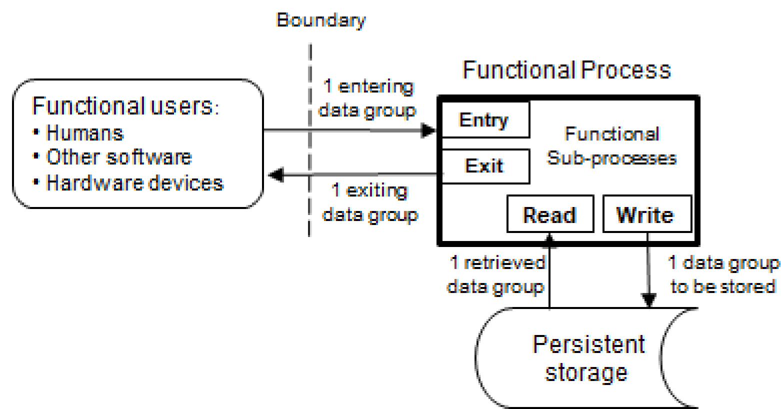


Figure 3.5 – The four sub-types of the data movements of a functional process. (A functional process can, of course have many E, X, R and W data movements.)

It follows from the principles of the Generic Software Model and the definitions of a functional process and of the four types of data movements, that **a functional process must have *at least* one Entry (the triggering Entry to start) and *either* one Exit *or* one Write, as an outcome**. **A functional process must therefore have at least two data movements.** (If there was no outcome resulting from a triggering Entry, a functional process would be a 'black hole' that just sucked in data.)

It also follows that there is no *maximum* number of data movements in a single functional process. As already noted, single functional processes have been measured with over 100 data movements.

3.5.2 Identifying Entries (E)

The input of a functional process may consist of multiple Entries.

See the MM for rules concerning various cases. The rule of section 3.3.2 and the rules of sections 3.5.9 and 3.5.10 in this Guide are also important for identifying Entries.

3.5.3 Identifying Exits (X)

The output of a functional process may consist of multiple Exits. For example, the output of a business application may:

- be a report showing totals of 'things' at various levels of aggregation; each 'thing' will be a different object of interest, each needing its own Exit;
- show the results of enquiries where the Exits can vary depending on the input;
- show data groups that are unrelated to each other, e.g. an invoice which includes the fixed text of an advertisement for an unrelated service, needing a separate Exit.

See the MM for rules for distinguishing Exits. The rule of section 3.3.2 and the rules of sections 3.5.11 in this Guide are also important for identifying Exits.

3.5.4 Identifying Reads (R)

PRINCIPLE – Read
A Read data movement always accounts for any 'request to Read' functionality. (Hence a separate data movement shall not be counted for any 'request to Read' functionality). See also section 3.5.9.

See the MM for a fuller account of the principles and rules for a Read data movement. The rules of sections 3.5.9 in this Guide are also important for identifying Entries.

3.5.5 Identifying Writes (W)

PRINCIPLE – Write
a) A Write data movement always accounts for any response resulting from storing persistent data. (Hence a Write data movement accounts for e.g. a ‘return code’ reporting its success or failure.) b) A Write data movement shall be counted for any requirement to delete a data group from persistent storage.

See the MM for a fuller account of the principles and rules for a Write data movement.

3.5.6 On the data manipulations associated with data movements

DEFINITION – Data manipulation
Anything that happens to data processed by a functional process other than its movement into or out of a functional process, or between a functional process and persistent storage. NOTE: Data manipulation can be e.g. computation, logical decision-making, etc.

The fourth principle of the Generic Software Model states that data manipulation sub-processes are not measured. All data manipulation is considered to be accounted for by the data movement with which it is associated.

Hence data manipulation can be ignored when identifying the concepts needed for measurement EXCEPT if there is a FUR that must be measured for a *change* to the manipulation of the attributes of a data group, but not to the group’s movement. For such a case, the following rules will be needed that define the data manipulation associated with each data movement sub-type.

RULES – Data manipulation associated with data movements
a) An Entry accounts for all data manipulation to enable a data group to be entered (e.g. formatting and presentation manipulations) and to be validated, except for any validation that requires other, additional data movements. b) An Exit accounts for all data manipulation to create a data group and to prepare it for output, (e.g. by formatting and presentation manipulations), and to be routed to the intended functional user. c) A Read accounts for all data manipulation needed in order to retrieve a data group from persistent storage. d) A Write accounts for all data manipulation needed in order to create or to update a data group to be moved to persistent storage, or to delete a data group from persistent storage.

See the MM for a fuller discussion of the data manipulation to be associated with each data movement sub-type, and for some examples.

3.5.7 Data movement uniqueness and possible exceptions

RULE – Data movement uniqueness and possible exceptions
Unless the Functional User Requirements specify otherwise, all data describing any one object of interest that is required to be entered into one functional process shall be identified as one data group moved by one Entry. NOTE: A functional process may, of course, have multiple Entries, each moving data describing a different object of interest. The same equivalent rule applies to any Read, Write or Exit data movement in any one functional process.

EXAMPLE OF AN EXCEPTION TO THIS RULE: Suppose the FUR specify a functional process to merge and validate data from two input streams, each stream comprising a different data group, but both groups describe the same object of interest. Two Entries should be measured for the two input streams to this functional process.

For more rules and for examples of the exceptional cases where FUR 'specify otherwise', see the MM.

3.5.8 When a functional process is required to move data to or from storage

When a functional process is required to retrieve some data from storage, or to store some data, this can happen in three ways depending on the context. The functional process can move data:

- to or from persistent storage within its own boundary (via Writes or Reads);
- across its boundary to or from another piece of software that is one of its functional users, that will handle the storage task (via Exits and Entries);
- directly to or from a functional user that is a physical hardware storage device, e.g. if the functional process is part of a software device-driver (via Exits and Entries).

See the MM for a full description with examples of the functional processes and data movements in these various ways of moving data to or from storage.

3.5.9 When a functional process requires data from a functional user

If a functional process must obtain data from a functional user there are two cases. If the functional process does not need to tell the functional user what data to send, a single Entry is sufficient (per object of interest).

Alternatively, if a functional process must tell the functional user what data to send, the process must request the data via an Exit and then receive the data via an Entry.

EXAMPLES of when a functional process does not need to tell the functional user what data to send:

- *when the process, having received a data group via a triggering Entry, waits, expecting a further data group from the functional user via another Entry. (This can occur when a human functional user is entering data to business application software);*
- *when the process, having started, inspects the state of a hardware functional user and retrieves the data it requires via one Entry.*

See the MM for the various cases and the applicable rules, and for examples.

3.5.10 Navigation and display ‘control commands’ for human users

DEFINITION – Control command
A command that enables human functional users to control their use of the software but which does not involve any movement of data about an object of interest of the FUR of the software.

RULE – Control commands in applications with a human interface
In an application with a human interface, ignore ’control commands’.

N.B. The term ‘control command’ is used by the COSMIC method only for interactions of human functional users with software that do not involve entering or receiving data about.an object of interest. Such interactions must not be counted in a functional size measurement.

EXAMPLE CONTROL COMMANDS:

- *Commands to ‘page up/down’ or between physical screens, to hit a Tab or Enter key, or to press a ‘Continue’ button.*
- *Clicking on an ‘OK’ button to confirm or cancel a previous action, or to acknowledge an error message or to confirm some entered data, etc.*
- *Menu commands that enable a user to navigate to one or more functional processes but which do not initiate a functional process. (It is the arrival of the data group moved by the triggering Entry that initiates a process, not the Menu command per se.)*

In other contexts, the normal meaning of ‘control command’ applies, e.g. a command sent by some real-time software to control a sensor should be measured as an Exit according to the normal rules.

See the MM for more examples of control commands.

3.5.11 Error/Confirmation Messages and other indications of error conditions

DEFINITION – Error/confirmation message
An Exit issued by a functional process to a human functional user that either confirms only that entered data has been accepted, or only that there is an error in the entered data.

RULES – Error/confirmation messages and other indications of error conditions
a) One Exit shall be identified to account for all types of error/confirmation messages issued by any one functional process to a human functional user from all possible causes according to its FUR.
b) If a message to a human functional user provides data in addition to confirming that entered data has been accepted or is in error, then this additional data should be identified as one or more data groups, each moved by an Exit in the normal way, in addition to the error/confirmation Exit.
c) All other data groups, issued or received by a functional process, to/from its hardware or software functional users should be measured as Exits or Entries respectively, regardless of whether or not the data values indicate an

error condition.
d) Reads and Writes account for any associated reporting of error conditions.

EXAMPLE illustrating rule a): In a human-computer dialogue, examples of error messages occurring during validation of data being entered could be 'format error', 'customer not found', 'error: please tick check box indicating you have read our terms and conditions', 'credit limit exceeded', etc. All such error messages should be considered as occurrences of one Exit in each functional process where such messages may occur.

See the MM for more examples.

3.5.12 Identifying data movements that must be modified

(The text in this section appears in section 4.4.1 in the Measurement Manual.)

When an existing software system must be modified, as in a maintenance or enhancement activity, and the size of the modification must be measured, the task is to identify the data movements that are affected by the FUR for the modifications.

DEFINITION – Modification (of the functionality of a data movement)
a) A data movement is considered to be functionally modified if at least one of the following applies: • the data group moved is modified, • the data manipulation associated with the data movement is modified. b) A data group is modified if at least one of the following applies: • one or more new attributes are added to the data group, • one or more existing attributes are removed from the data group, • one or more existing attributes are modified, e.g. in meaning or format (but not in their values) c) A data manipulation is modified if the manipulation is changed in any way.

EXAMPLE: A data manipulation is modified for instance by changing the calculation, the specific formatting, presentation, and/or validation of the data. 'Presentation' can mean, for example the font, background colour, field length, field heading, number of decimal places, etc.

RULES – Modifying a data movement
a) If a data movement must be modified due to a change of the data manipulation associated with the data movement and/or due to a change in the number or type of the attributes in the data group moved, one changed data movement shall be identified, regardless of the actual number of modifications in the one data movement. b) If a data group must be modified, data movements moving the modified data group whose functionality is not affected by the modification to the data group shall not be identified as changed data movements.

EXAMPLE: A change request for a functional process requires three changes to the data manipulation associated with its triggering Entry and two changes to the manipulation associated with an Exit, as well as two changes to the format of attributes of the data

group moved by this Exit. Identify one Entry and one Exit as changed. Do NOT count the number of data manipulations or data attributes to be changed.

EXAMPLE: A required modification to a functional process FP1 results in a change to a data group X moved to persistent storage. The Write data movement of FP1 must be identified as modified. Another functional process, FP2 (in the same or another system) must read the data group X, but the functionality of the Read data movement is unaffected by the fact that the data group it moves has been changed. Do NOT identify the Read of FP2 as modified.

See section 4.4.1 of the MM for more examples of measuring modified software.

3.6 Identifying COSMIC concepts in available software artefacts

Given the enormous variety of possible software artefacts, the various ways of identifying COSMIC concepts are best illustrated by a few cases.

User Stories: The convention for stating a User Story starts with: 'As a [functional user], I want to [functional process) .....' The nature of the 'I want to', i.e. the action, then helps us identify candidate objects of interest (underlined in the example below).

Example: 'As a traveler, having selected a hotel, I may want to book a room.' Domain or real-life experience tells us that this high-level requirement must be met by a series of functional processes. At each point in the series, the traveler must make a new decision (a triggering event) to continue or to stop making the booking. The way functionality is spread over various processes and data movements will depend on the exact wording of the Stories. The following Stories, each for one functional process, form one possible sequence:

- *'Given my number of guests and preferred dates, I want to enquire on the availability and price of each room-type'. {The data group moved by the Entry for this enquiry process is for a possible reservation for each room-type, in the first 'offered' stage of its life-cycle. The multiple occurrences of the data group displayed by the Exit show the price and outline description for each room-type that is actually available for the given dates; these are attributes of the offered reservations for the given input data.)*
- *'For a selected room-type, I want to see all the room-type details and booking conditions.'*
- *'For this hotel, I do not like any of the offers so want to stop my enquiries'.*
- *(OR) 'For my selected room type and preferred dates I want to make a reservation'*. *(At this point the traveler ID and contact details must be entered. One of the offered reservations then becomes a confirmed reservation.)*
- *'For my reservation I want to pay the deposit, if required by the booking conditions', etc.*

The first of these enquiries must have an Entry for the number of guests and preferred dates, a Read and an Exit for the offered reservations, and an Exit for any error/confirmation messages. There may also be an Exit to start a timer to 'lock' the offered reservations to prevent double-booking of the same reservation until it is accepted or released by the traveler, or timed-out by the system.

Data artefacts: As noted in section 3.3.2, an examination of data models for persistent data or of a (normalized) physical database or file definitions, helps us identify objects of interest. Knowing the objects of interest, we can then identify the functional processes and some of the data movements that must process data about them.

Each object of interest almost certainly implies that the software must have functional processes to Create data about the object of interest, to Update the data (often several, in response to different triggering events), to Read the data (again several for various

purposes) and finally to Delete or maybe archive the data. All these processes must have corresponding Reads and Writes.

APIs: The definition of an Application Programming Interface (API) of a software component should automatically tell us the various events that the component will respond to and therefore its corresponding functional processes. Furthermore, the API must define the Entries needed to call the component and the responses or Exits that will be returned. (The sum of the Entries and Exits of an API provide an interesting measure of the size of its interface functionality that is available to any calling software.) These data movements of the component must correspond to the Exits and Entries, respectively, of the processes of the calling software. As an example, a recent study [10] showed how COSMIC sizing was applied to estimate the effort and memory space to incorporate a geolocation service from its Open API Specification.

UML: Use Cases defined in the conventions of the Unified Modelling Language can help identify functional processes directly. However be careful because Use Case diagrams can be drawn at any level of granularity. A single Use Case may describe the interactions of a user with a cluster of related functional processes, or a single functional process, or even a part of a functional process. (The UML does not explicitly define the concept of an 'event'.)

Sequence Diagrams that show the communications between the methods of object classes are also useful because these communications are potential data movements. The methods themselves may be whole functional processes or sub-processes.

Physical input/output: Examining screens and reports provided to users of an installed system will help identify the functional processes that the software can support (e.g. from menus) and the Entries and Exits of the processes. With experience or system/domain knowledge, the Reads and Writes of the input and output data can be deduced.

Warning**:** be careful to distinguish and measure the logical functional processes of the FUR and not the physical transactions that may have been implemented for the system. See Sections 3.2.5 - 3.2.8 in this Guide and in the MM for more examples.)

Events: For real-time software, knowing the events that the software must respond to will tell us the functional processes of the software. The hardware devices (functional users) that provide data to the software tell us the Entries of these processes, whilst the devices that receive data tell us the Exits they receive.

In real-time software, when a functional user sends or receives data about itself, the functional user is also the object of interest of the data group moved. So there is often no need to separately identify objects of interest in addition to the functional users of the software (as is usually necessary for business application software). This makes the application of COSMIC sizing to real-time software particularly easy.

Furthermore real-time specification and design languages and tools often use a syntax that maps directly to COSMIC concepts. For more, see the 'Guideline for sizing real-time software' [3].

In summary, if you fully understand the contents of this chapter, then with domain knowledge and/or with the help of a system expert, identifying COSMIC concepts in the available software artefacts is often very easy.

Finally, it's worth remembering that the most common mistake by those performing functional size measurement is to miss certain functional processes or data movements, and so under-size the software. Be warned.

4

THE MEASUREMENT PHASE

4.0 Chapter summary

In this final phase, we calculate the required software sizes from the data movements and functional processes identified in the Mapping phase, drawing on the last three principles of the Generic Software Model. This chapter therefore describes the steps and rules for:

- the basic process of obtaining the size of a functional process by adding up the number of its data movements, and then the size of the software by adding the sizes of its functional processes;
- obtaining the size of some software from the sizes of its components, or for example, obtaining the size of a release from the sizes of its sprints, or obtaining the size of a sprint from the sizes of its user stories – the so-called 'aggregation rules';
- measuring the size of enhancement or maintenance requirements that may involve combinations of adding, changing and deleting functionality.

The chapter also describes how to extend the rules of the COSMIC method locally for sizing other aspects of the FUR. For the chapter structure, see Figure 4.1.

4.1 The Measurement Phase

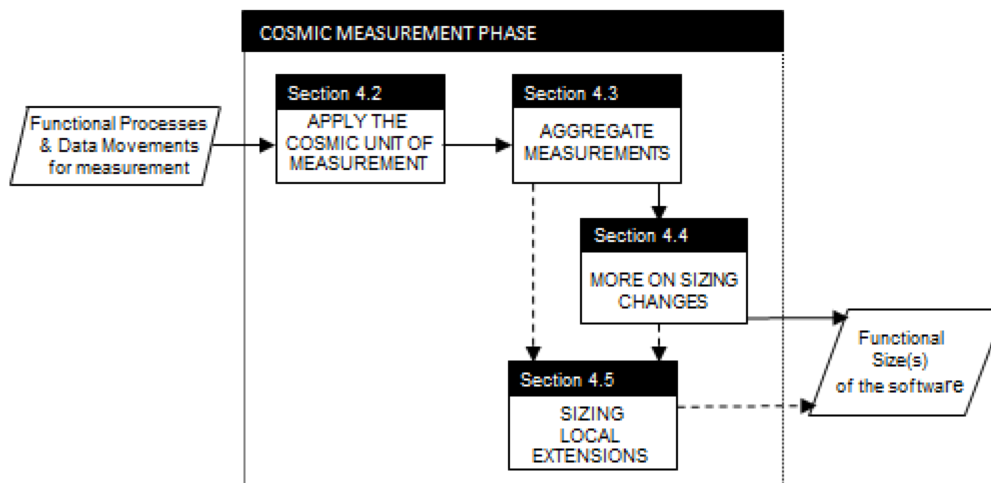


Figure 4.1 – The steps of the COSMIC Measurement Phase

4.2 Applying the COSMIC unit of measurement

DEFINITION – COSMIC unit of measurement
The COSMIC method unit of measurement is one data movement, denoted as one Cosmic Function Point (or one 'CFP').

Each data movement (Entry, Exit, Read or Write) that is required to be added, modified or deleted for the software being measured is therefore also measured as one CFP.

4.3 Aggregating measurement results

4.3.1 General rules of aggregation

RULES – Aggregating measurement results
a) The functional size of any functional process shall be measured in units of CFP by aggregating the sizes of its data movements. Size (functional process) = Σ size (Entries) + Σ size (Exits) + Σ size (Reads) + Σ size (Writes)
b) The functional size of changes to a functional process shall be measured in units of CFP by aggregating the sizes of the data movements that must be added, modified or deleted in the functional process. Size (Change(functional process)) = Σ size (added data movements) + Σ size (modified data movements) + Σ size (deleted data movements) NOTE: For the meaning of 'modified' data movements see section 3.5.12.
c) The size of the FUR for a piece of software shall be obtained by aggregating the sizes of the functional processes within its scope, subject to rules e) and f) below.
d) The size of the FUR for any changes to a piece of software shall be obtained by aggregating the sizes of all changes to all functional processes within its scope, subject to rules e) and f) below.
e) Sizes of pieces of software or of changes to pieces of software may be added together only if measured at the same functional process level of granularity of their FUR.
f) Sizes of pieces of software and/or changes in the sizes of pieces of software within any one layer or from different layers may be added together only if it makes sense to do so for the purpose of the measurement. NOTE: For more on aggregating functional sizes, see section 4.3.2. For measuring the change in the size of a piece of software that has been changed, see section 4.4.2.
g) The size of a piece of software may be obtained by adding up the sizes of its components (regardless of how the piece is decomposed) and then eliminating the size contributions of inter-component data movements.
h) If the COSMIC method is extended locally (see further in section 4.5), then the size measured via the local extension shall NOT be added to, and shall be reported separately from, any size measured in CFP.

EXAMPLE FOR RULE g): Figure 4.3 shows View b) of the distributed business application 'A' from Figure 2.2 (and as also discussed in section 3.2.6) in more detail.

Suppose we must calculate the total size of the Application 'A', i.e. as in View a) in Figure 2.2, from the sizes of its three components measured separately as in Figure 4.3.

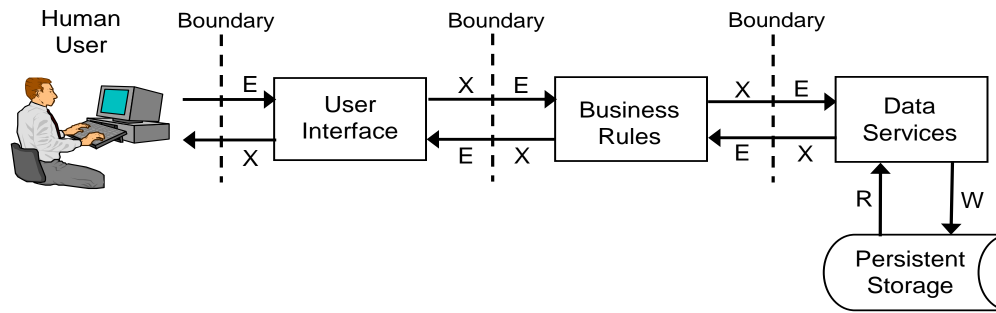


Figure 4.3. The three components of Application ‘A’ in Figure 2.2

Using the rule g) above:

Size of Application ‘A’ = Σ size (UI + BR + DS) <u>less</u> Σ size (all inter-component X/E pairs), where one X/E pair = 2 CFP.

See the MM for more examples of size aggregation.

4.3.2 More about functional size aggregation

Size aggregation rule f) is important, for example when developing effort estimation models. Suppose the three components of the distributed application ’A’ in Figure 4.1 were developed using different technologies which could have different associated productivity levels. It might then be preferable to develop three separate estimation models, one for each technology. The alternative of aggregating the sizes to develop one estimation model for future distributed applications such as ‘A’ would need to take account of the proportions of the size contributions of the three components. These proportions and the number of X/E pairs may vary from one application to another, making the single estimation model complex.

See the MM for a discussion and more examples of size aggregation when the software to be delivered and measured extends over different layers of a software architecture.

4.4 More on measurement of the size of changes to software

The need for a change to software may arise for various reasons, e.g.

- a new FUR (i.e. only additions to the existing functionality);
- from a change to the FUR, perhaps involving additions, modifications and deletions (referred to as a maintenance task or as an enhancement);
- from a maintenance task to correct a defect.

The rules for sizing any of these changes are the same but there may be other practical issues to consider when the purpose of measuring the sizes of changes is to support project performance measurement or estimation. See the MM for a discussion of these other issues.

4.4.1 Modifying functionality

(The text of this section has been moved to section 3.5.12 in this Guide.)

4.4.2 Size of functionally-changed software

RULES – Size of functionally-changed software
After functionally changing a piece of software: Size after change = Size before change *plus* Σ size (added data movements) *less* Σ size (deleted data movements)

Modified data movements have no influence on the size of the changed piece of software as they exist both before and after the modifications have been made.

EXAMPLE: A change to a piece of software requires adding one new functional process of size 6 CFP, and in another existing functional process adding one data movement, modifying three other data movements and deleting two data movements. The size of the required change is 6 + 1 + 3 + 2 = 12 CFP.

The total size of the piece of software will have increased by 6 CFP due to the addition of the one new functional process and will have decreased by 1 CFP due to net effect of the additions (+1 CFP) and deletions (-2 CFP) for the other functional process. After the change, the piece of software will therefore have increased in size by (+6 - 1) = 5 CFP.

4.5 Extending the COSMIC measurement method

4.5.1 Introduction

The COSMIC method allows the possibility of extending the method locally to measure some aspect of FUR for its software in more detail than the standard method. For example an organization may wish to account for its requirements for algorithms explicitly for the purposes of effort estimation.

Only rule h) of section 4.3.1 applies to measuring such local extensions. See the MM for more on extending the method.

4.5.2 Data manipulation-rich software

In spite of not explicitly measuring data manipulation, experience has shown that the COSMIC method can be successfully applied to measure some types of 'data manipulation-rich' software. See the MM for a fuller discussion.

4.5.3 Limitations on the factors contributing to functional size

Although a COSMIC-measured size does not account for all aspects of the 'complexity' of software (however defined), it does account in a simple way for processing complexity, and thus indirectly for the complexity of the data processed. See the MM for a fuller discussion.

4.5.4 Limitations on measuring very small pieces of software

Because the COSMIC method measures a simplified model of FUR, it might be expected for statistical reasons that CFP sizes of very small pieces of software would be of limited value for the principal purposes of measurement as stated in Chapter 0.

However, recent studies have shown that the COSMIC method can be successfully used to measure sizes and then to compare the productivity of small enhancement projects, and of sprints and even individual User Stories in an Agile environment – see Chapter 6 of this Guide for examples.

4.5.5 Local extension with complex algorithms

See the MM for a fuller discussion and for an example.

4.5.6 Local extension with sub-units of measurement

See the MM for a fuller discussion.

5

MEASUREMENT REPORTING

5.0 Chapter summary

This chapter lists the parameters that should be considered for recording of measurements.

5.1 Labeling

Appendix A of the MM gives a simple example of a spreadsheet for recording a COSMIC size measurement.

RULE – COSMIC measurement labeling
A COSMIC measurement result shall be noted as '***x*** CFP (v) ', where: • '**x**' is the numerical value of the functional size, • 'v' identifies the version of the standard COSMIC method used to obtain the functional size value 'x'.

RULE – COSMIC local extensions labeling
A COSMIC measurement result using local extensions shall be noted as: '**x** CFP (v.) + z Local FP', where: • '**x**' represents the numerical value obtained by aggregating all individual measurement results according to the standard COSMIC method, • 'v' identifies the version of the standard COSMIC method used to obtain the functional size value 'x'. • 'z' represents the numerical value obtained by aggregating all individual measurement results obtained from local extensions to the COSMIC method.

5.2 Archiving COSMIC measurement results

The MM gives a long list of data that might need to be kept when archiving COSMIC measured sizes, so as to ensure that their meaning is always clear for future users of the sizes.

6

COSMIC SIZE MEASUREMENT IN PRACTICE

6.0 Chapter Summary

This Chapter provides brief advice on some practical questions that inevitably arise when thinking about implementing a software metrics and estimating programme based on COSMIC functional size measurement.

6.1 Estimating an approximate COSMIC size from incomplete FUR

In the early stages of a new project, the Functional User Requirements (FUR) for the software are rarely available in sufficient detail that a precise CFP size measurement is possible. However, several variants of the standard sizing method exist for estimating an approximate CFP size from incomplete FUR, as described in a COSMIC Guideline [11].

As an illustration, the simplest variant for estimating an approximate CFP size of a piece of software has two steps:

a) Estimate the total number ‘N’ of the functional processes of the new software, either by identifying them specifically, by inferring the need for them using an approach such as described in section 3.2.2, or by expert judgement..

b) Estimate the average size ‘S_{AV}’ of a functional process of the new software. Do this by measuring the average size of the functional processes of other software that is functionally similar to the new piece of software. (This is the ‘calibration’ step.)

The estimated total size of the piece of software is then obtained by multiplying the estimated number ‘N’ of functional processes by the estimated average size S_{AV}.

The Guideline [11] describes other variants that aim to help estimate more accurate sizes than this simplest approach, and gives example calibration results. However, for the best accuracy, you should always calibrate your chosen variant locally, based on measurements of CFP sizes on your own software.

Variants that help estimate more accurate sizes are particularly necessary for approximate sizing of software whose functional processes have a skewed size distribution, e.g. typically ranging from many small processes to a few very large processes. Pay particular attention to estimating the size contribution of the few very large processes if you aim to estimate an accurate size for such a piece of software.

The biggest difficulty in early estimation of an approximate size, apart from the lack of detail, is often having to take account of requirements that have hardly been considered at all or are missing. A COSMIC Guideline [12] describes ways of documenting the quality of requirements to help indicate any limits on the accuracy of the estimated size.

In summary, for accurate sizing early in a project, calibrate your chosen approximation variant locally, pay particular attention to sizing any very large functional processes and consider how to allow for functionality that may have been missed, e.g. by allowing for a contingency.

6.2 Using COSMIC sizing in Agile software development

The Agile practice of using 'Story Points' (SPs) for planning and estimating the time and effort to develop, test and implement User Stories (i.e. statements of FUR) and sprints is of little value for achieving our goals of software size measurement set out in the Foreword to this Guide. Individual teams may be content with their use of SPs for their own planning purposes, but the meaning of a SP is local to each individual team. An SP is a very poor unit of measurement for long-term learning on how to improve performance, for managing large projects across multiple teams, for early budget estimation purposes, etc.

As an alternative to Story Points, the rules for measuring COSMIC sizes fit perfectly with Agile frameworks, such as the Scaled Agile Framework (SAFe) because the size aggregation rules enable consistent CFP size measurement at all levels from User Stories up to whole software systems. And using CFP sizes ensures valid, reliable comparisons across multiple teams.

Moreover, multiple studies show that CFP sizes correlate much better with effort than do SP 'sizes'. Figure 6.2 shows the results obtained in a Canadian company from a study of the size/effort relationship for completed sprints, where sizes were measured with both SP and CFP [13]. In this organization estimated SPs were converted directly to work-hours. Each dot represents one sprint.

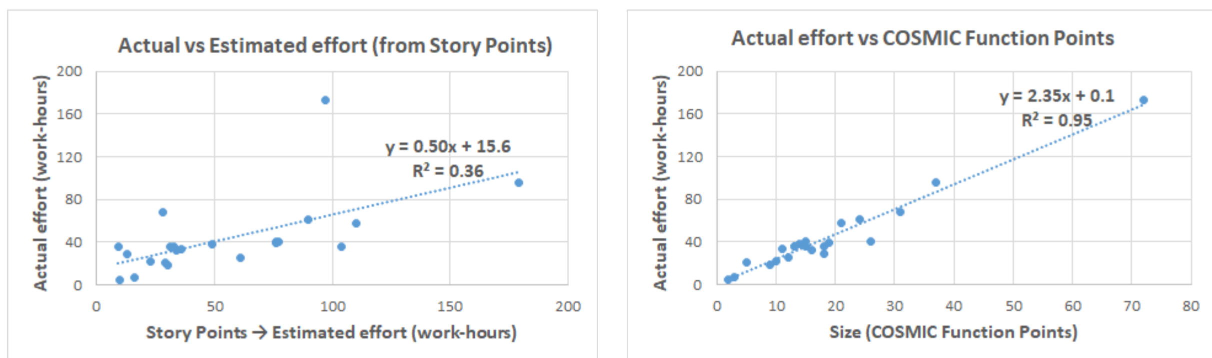


Figure 6.2: Actual effort vs SP, and actual effort vs CFP for 22 sprints of a Canadian supplier of security and surveillance systems

The report [13] presents the results of the size/effort relationships for agile sprints from a total of four organizations; all four sets of results revealed a much closer CFP/effort relationship, and a much closer extrapolation of the fitted line to the (Effort, CFP) intercept of (0,0) than for the SP/effort relationship. Note that CFP sizes were measured retrospectively.

Another study [14] in an Indian-headquartered global healthcare testing company similarly showed much better size/effort relationships when size was measured in CFP rather than with SP. For this study, however, the measurements were at the level of individual User Stories (rather than at the level of sprints as shown in Figure 6.2). Each User Story ranged in size up to 20 CFP or 13 SP.

Introducing COSMIC sizing into an agile environment

The practical issue is therefore not *whether* but *how best* to first introduce CFP measurement into an established Agile (or DevOps) environment so as to benefit from standard software size measurement, without disrupting existing practices. Experience suggests to start by measuring CFP sizes when completing sprints or releases, leaving individual teams to continue to use SPs for their own sprint planning purposes. Teams can begin to use CFPs to replace SPs as confidence in the use of CFP measurement grows. They will then find that

identifying the COSMIC concepts will support the analysis process, leading to improved quality of User Stories.

Agile teams may then recognize that ‘velocity’ (actual SP/estimated SP) really conflates two measures: ‘productivity’ (size/actual effort), and ‘estimation accuracy’ (actual effort/estimated effort).

An interesting endorsement for using COSMIC sizes in an agile environment came from Denis Krazinovic of Aon Australia. He blogged in 2014: “*We have found that adopting this approach provides us with excellent predictability and comparability across projects, teams, time and technologies. The reality of achieving predictable project performance has driven me to investigate many methods of prediction. COSMIC is the method that lets me sleep at night.*”

6.3 CFP size/effort data and productivity benchmarks

Adopting or establishing an average or ‘benchmark’ productivity for each class of your organization’s software projects (where a class implies a set of common characteristics such as software domain, technology, level of decomposition, etc.) is essential if you want to develop local effort-estimation methods (see section 6.5) and to undertake performance improvement actions (see section 6.7).

When starting a software measurement program, you will have limited CFP size/effort data from your own organization’s projects. One option is then to consider using publicly-available benchmark productivity data for COSMIC-measured projects such as from the International Software Benchmarking Standards Group [15]. These data, from diverse organizations, inevitably show a wide spread of productivity. However, the ISBSG productivity data may provide interesting comparators and plausibility checks on your own measurements.

Undoubtedly the best long-term approach [16] is to collect and record size and effort data from your own organization’s projects and to establish your own benchmark productivity levels for your own classes of projects.

As an example, the average productivity of the sprints shown in Figure 6.2 is 1 / 2.35 = 0.43 CFP per work-hour. This value could be taken as the first benchmark productivity for agile sprints for this class of software in this organization.

There are now many study results that report good CFP size/effort correlations for different classes of software, and that therefore give confidence that CFP sizes can be used to give meaningful productivity measurements and as input for accurate effort estimates.

In addition to the results for Agile sprints, good correlations have also been reported for activities as diverse as monthly releases of enhancements to a Chinese insurance company’s applications [17], and for developments of software embedded in the electronic control units (ECUs) of a European automotive manufacturer [18]. Another study [19] showed that CFP sizes of industrial web applications correlate better with effort than sizes of the same applications measured by a ‘First Generation’ FSM method.

6.4 Measurement of project effort

To measure project productivity and to use these measurements to establish benchmarks and build estimation models, you must obviously record project effort (and duration) data in a consistent way across all projects, as well as software size. Be aware that this task can need as much careful attention as software size measurement. ISBSG data collection forms show the variety of factors that need to be considered to measure project effort data consistently.

6.5 Use of COSMIC sizing as the foundation metric for estimating project effort

The formula given in Chapter 0 for estimating project effort (by dividing an estimated size in CFP by an expected, or 'benchmark' productivity figure) is of course only a simple starting point.

In practice and with experience, this first 'average expected effort' estimate can be refined by taking into account various factors specific to the project being estimated. Dozens of factors (or 'cost-drivers') can potentially impact the finally-estimated effort, including:

- risk factors, such as uncertainties in the software requirements, whether the available staff have the experience to tackle the new challenge (or are very experienced so should be more productive), etc.;
- constraints such as delivery deadlines, inter-dependencies with other related projects, Non-Functional Requirements, etc.;
- the methods and technology to be used for the software development and the platform for its execution.

Many of these factors are taken into account by open or proprietary estimating methods and tools. These usually require input of an initial estimate of software size in units such as SLOC or in the units of an older FSM method. One approach to using CFP sizes for estimation is therefore to continue using these methods and tools but to re-calibrate them by substituting CFP sizes for the existing input size units.

However, developing your own in-house estimation model using your own CFP measurements and benchmarks [16], should have the advantage that the number of factors that affect in-house productivity is far fewer than proprietary 'black-box' estimating tools need to take into account. An in-house estimating model should therefore be easier to understand and much simpler to use.

Developing an in-house model does mean that as well as collecting size and effort data for each completed project you will have to collect data on the factors that are found to have influenced the performance of those projects. These can be gathered from post-project reviews or agile retrospectives. Analysis of the data will then reveal the few important factors that influence performance locally. All these data add to organizational learning and confidence in the use of software metrics.

6.6 Use of COSMIC sizing as the foundation metric for estimating processor memory size

As well as being used in the context of project performance measurement and effort estimation, CFP sizes have also been shown to correlate well with the memory size (in bytes) of microprocessors needed for implementation of the requirements. This is the case for CFP sizes estimated at the design stage for software embedded in automotive system electronic control units [20], [21] and for smartphone apps [22].

6.7 Using measurements to improve organizational performance

The emphasis in this Guide has been on the measurement of productivity and effort estimation and also measurement of product quality in terms of defect density (Defects/CFP). But in many circumstances time can be more important than efficiency. So what has been written for productivity measurement is equally valid for measuring speed (CFP/duration). Note however, that whereas size and effort usually have a linear relationship, the size versus duration relationship is usually non-linear, e.g.

Speed (size/duration) = C x $(Size)^N$, where C and N are constants, and N is less than one.

Consequently, Duration = $(Size)^{1-N}$ / C.

Monitoring productivity, speed, and quality data (such as defect density) over time, combined with knowledge of the factors ('cost-drivers') that drive these performance parameters provides a powerful and valuable foundation for improving organizational performance.

The variety of performance measures that can be derived once you have measured a size of software is limited only by your imagination and the value of the measure. The following is a small sample of possible useful measures and analyses. Once you have gathered enough data on actual performance, you may then seek to build models to predict some of these measures for new projects.

- Defects found, and removed per CFP, for the whole development or for phases or releases of the software life-cycle.
- Maintenance and support productivity, e.g. CFP supported per support staff-member (which depends on how 'support' is defined locally.)
- Tests per CFP.
- Developers per CFP, usable for resource allocation.
- Artefacts (e.g. pages of documentation) per CFP
- Gaining insights into the effort/duration trade-off for project design.
- Gaining insights into how performance depends on the balance of effort (or time or resource) spent on the various project activities (e.g. analysis, design, programming, testing and project management).
- (If you can track performance data over enough time), gaining insights into the trade-off between effort and time spent on an original development versus the quality of the resulting software product and effort to maintain and support it over its life.
- Value-for-money evaluations (benefits realized versus estimated and/or versus development and maintenance costs).

A note of caution: The performance measures described here are great for helping identify areas of waste or inefficiency that need attention, training needs, etc.., and for monitoring the outcome of investments in performance improvement e.g. when adopting a new technology.

But be very careful if you want to use performance measures as *targets or incentives for individuals or groups to improve performance*. There are many aspects to performance which are tradeable (so focusing on one target may be easy to achieve at the cost of under-performing on another aspect of performance), and targets are easily gamed. I am not saying 'don't set targets for performance improvement'. I am saying 'proceed with caution'.

One final thought. Taken altogether, these capabilities:

- to predict effort and duration and/or memory space and then costs, early in development,
- to support the achievement of product quality,
- and to help achieve organizational performance improvement,

are of enormous economic importance.

6.8 Where to get more information on use of COSMIC sizing in practice

Go to www.cosmic-sizing.com for a wealth of guidelines, case studies and research and conference papers on the many uses of CFP sizing. All documents are available for free-download, and many are available in multiple languages.

7

EXERCISES

The exercises in this Chapter are designed to test your understanding of the COSMIC Functional Size Measurement method based on the content of this Guide and your general knowledge of software systems. You will not necessarily find the exact answer to each question in this Guide. You may have to derive some of the answers from the content of the Guide.

Some questions may have more than one answer. I have tried to choose questions that do not require specialist domain knowledge so that any software professional should be able to answer them. But if you do have more specialist domain knowledge than me, feel free to challenge my answers. And different answers may be valid depending on the assumptions you make. Welcome to the real world of functional size measurement!

Section 7.1 has questions mostly requiring 'True' or False' answers. Section 7.2 has two mini cases studies. Section 7.3 has my answers to all the exercises. Please inform me at cr.symons@btinternet.com if you think my answers are wrong or incomplete – of if you have any other ideas to improve the Guide.

[The type of questions in these exercises are different from the questions you should expect to find in a COSMIC certification examination. An examination is held on-line and assessed automatically. All examination questions therefore must have precise answers with no uncertainty.]

7.1 Questions

1. The following statements concern the **applicability** of the COSMIC method. They describe various types of software for which the COSMIC method **can measure a valid, meaningful, functional size**. Decide if each statement is TRUE, or FALSE, or PARTLY TRUE or FALSE.
   a) The national software system to enable citizens to declare their annual income for tax collection purposes and to compute their tax liability.
   b) A software system to maintain a common set of tables of codes and descriptions of 'entities' that must be referenced by all applications of a large organization to ensure consistent data coding. 'Entities' means e.g. countries, currencies, offices, factories, finished products, etc.
   c) The avionics software of a civil aircraft.
   d) The software systems to automatically collect meteorological data from unmanned weather stations.
   e) A weather forecasting system, updated periodically from data collected automatically from weather stations
   f) Apps to run on a smartphone or tablet.
   g) An enquiry system to enable epidemiologists to search a database of hospital records of infections and deaths from identified diseases to establish statistics and risk factors by gender, age, location, ethnicity, blood-group, etc.
   h) The application software to drive a video-game.
   i) A computer operating system.

j) The software of an internet router.

k) A component of a Business Rules sub-system to calculate the annual car insurance premium for a given combination of driver, driver-location and vehicle.

2. The following statements concern the **concepts** that are used to define the COSMIC method principles, and the topic of **types versus occurrences**. Decide whether each statement is TRUE or FALSE.

   a) The concepts needed for a CFP measurement can only be extracted from statements of FUR.

   b) Rules for extracting COSMIC concepts from any software artefact can only be established locally due to the enormous variety of artefacts.

   c) The functional size of a piece of software depends on the number of types of COSMIC concepts found in its FUR.

   d) The functional size of a piece of software is totally independent of the number of occurrences of any of its concepts that the software is required to execute.

   e) The number of occurrences of any concept is totally irrelevant to the process of measuring a functional size.

3. The following statements concern **FUR, NFR or project requirements**. Decide whether each statement is 'TRUE' or 'FALSE'.

   a) A statement of FUR that 'the application shall meet external audit standards' is a Non-Functional Requirement (NFR).

   b) FUR statements effectively constrain how software should be implemented.

   c) A requirement that a software application can handle a maximum of 1000 concurrent users is a Functional User Requirement (FUR).

   d) The statement: "The application shall be accessible only via the Company's standard login procedure" is a FUR for the application.

   e) The statement: "The Customers who are responsible for delivering the system benefits shall sign-off the system test results before release for public use" is a NFR.

   f) The statement: "The controller software shall continue to operate without interruption if mains power fails (when power is switched to the stand-by generator) or is restored" is a FUR for the software.

4. The following statements concern the **scope** of a measurement. Decide whether each statement is TRUE or FALSE

   a) The scope of a piece of software to be measured can be defined by drawing the boundary around the software.

   b) Before you can define the scope of a measurement you must determine if the software to be measured extends over more than one layer of the architecture in which it resides.

   c) Two pieces of software whose sizes must be measured reside in different layers of an architecture; they exchange data in a 'master/slave' hierarchical relationship. Each piece is a functional user of the other piece.

   d) The master piece of software in question c) writes data to persistent storage that is read by the slave piece of software. Each piece has its own persistent storage. Interactions take place by exchanges between the two persistent stores.

   e) The scope of some software whose total size must be measured may be assembled from components at different levels of decomposition.

   f) The FUR for an enhancement project that must be measured must always be limited to one piece of software to be enhanced.

5. The following statements concern identifying the **functional users** of a piece of software to be measured. Decide whether each statement is 'TRUE' or 'FALSE', maybe depending on certain assumptions.
   a) The one functional user of a company's Human Resource (HR) database is defined as 'HR User'. This means: 'all HR Officers, HR Managers, the HR Director, Security, and Payroll Administration staff'.
   b) The functional users of an automated cow-milking software system are the cows.
   c) The sizes of the different 'views' (i.e. sub-sets) of the total functionality of a piece of software available to each functional user must be added together to obtain the total size of the software.
   d) When a functional process 'A' of software component 'X' must obtain some data from another software component 'Y', component X is a functional user of component Y.
   e) When a functional process 'A' of software component 'X' must obtain some data from a hardware device 'Y', component X is a functional user of the hardware device Y.
   f) Refer to the 'Real-time systems' example of section 1.3.3. Each of the 200 sensors that detect holes is individually identified by an ID. The ID is transmitted with the 'hole/no-hole detected' status in the data group sent to the controller when the sensor is polled. The presence of the ID does not affect the functional size. There is still only one functional user 'sensor (-type)', which has 200 occurrences.
6. The following statements concern the **level of granularity** of the FUR of a piece of software to be measured. Decide whether each statement is 'TRUE' or 'FALSE', maybe depending on certain assumptions.
   a) Different levels of granularity of FUR specify the break-down of a 'whole' piece of software into its main components, and of each component into sub-components, etc.
   b) Different functional users may be revealed as requirements are analyzed into lower levels of granularity.
   c) Agile User Stories may safely be assumed to be all expressed at the functional process level of granularity.
   d) When extracting COSMIC concepts from the artefacts of an existing, operational software system it may be assumed that the concepts will all be at the functional process level of granularity.
7. The following are statements of FUR. Decide whether each statement is for a **group** of functional processes, or a **single** functional process, or a **part** of a functional process.
   a) The application must maintain data about stock levels for all our products
   b) Interest shall be applied daily to savings account balances at the relevant current rate, obtained from the 'retail interest-rate tables'.
   c) Foreign income shall be credited at annual-budget exchange rates.
   d) The system shall check all four tire pressures at one-second intervals. If any tire pressure drops below standard, a warning light shall be illuminated.
   e) Each worker on the conveyor belt shall have an emergency-stop button. When pressed and held for 2 seconds, the software shall stop the conveyor belt and sound the alarm.
   f) The electronic control unit shall control the vehicle front and rear lights.

8. The following statements concern the definition and rules for distinguishing and measuring **functional processes**. Decide whether each statement is 'TRUE' or 'FALSE'.

a) A functional process is defined as ‘A unique set of data movements that is needed to meet the Functional User Requirements for all the possible responses to the data entered by its triggering Entry.’
b) A single event can result in triggering several functional processes.
c) A real-world event (something that happens) must be the same for all observers (i.e. all potential functional users).
d) A functional process can have only one triggering Entry.
e) Two separate pieces of software can exchange data only via functional processes within the scope of each separate piece of software.
f) When a human functional user decides to make an enquiry on a database, the user’s decision is the triggering event for the enquiry process.
g) Two Entries may be necessary to trigger a functional process of software that will be executed in batch mode. The first is the triggering Entry of a process to start the batch processing. The second is the triggering Entry of a process that moves the first data group in the batch of data to be processed.
h) Two functional processes use three identical data movements, which can be implemented as a common module. To avoid double-counting in the size measurement, the three shared data movements shall be counted in only one of the functional processes.
i) To measure the size of a functional process in CFPs you must determine the number of possible paths through the process when it is executed.

9. The following statements concern the definition and rules for distinguishing **objects of interest**. Decide whether each statement is ‘TRUE’ or ‘FALSE’.
   a) ‘Object of interest’ is a synonym for ‘object-class’ in the terminology of object-oriented design.
   b) The FUR of a piece of software may specify many different data groups, all describing the same object of interest, to be moved in the same or different functional processes.
   c) When a human functional user identifies himself in a login process, the human user is also the object of interest of the login functional processes.
   d) The value of the total sales in units of $ of a given product in a given month is an attribute of a conceptual object of interest.
   e) If you have a complete definition of all data stored by a software system, then you can derive all the objects of interest of the data groups moved by the system.
   f) Objects of interest found in FUR at a high level of granularity always evolve as the FUR are worked out in more detail to different objects of interest at lower levels of granularity.
   g) Data for an applicant for a new passport entered into a Passport Application System (PAS) must include the applicant’s ‘country of birth’. The screen for on-line data entry requires the country to be selected from a drop-down list of standard country names. The list is obtained from a table of standard country names that is maintained by other functions of the PAS. Because ‘country of birth’ is an attribute of the data group describing the object of interest ‘applicant’, a country cannot be an object of interest in the PAS.
   h) The database of a Recruitment Agency’s software system holds data on:
      - its job-seeking clients, including their educational qualifications and employment history;
      - its employer clients, and the job-vacancies they wish to fill;
      - the job-seeker/employer interviews that the Agency arranges.

      The database holds data describing at least six objects of interest.

10. The following statements concern the definition and rules for identifying the four types of **data movements**. Decide whether each statement is 'TRUE' or 'FALSE'.
    a) A Read data movement accounts for the 'request to read' functionality and any data manipulation needed to prepare the request to read.
    b) An Entry always accounts for all the functionality needed to validate the data group that it moves.
    c) An Entry accounts for the 'empty' screen needed to enter a data group on-line.
    d) A Write data movement can be used to either make a data group persistent or to delete a data group from persistent storage.
    e) Each line (-type, not occurrence) output to a report always corresponds to one Exit.
    f) The print-driver component of an operating system communicates with its printer hardware functional user by Exit and Entry data movements.
    g) The answer you gave for question f) (True or False) is also valid if the print-driver component communicates with the printer firmware (which behaves like software), rather than directly with the hardware.
    h) The answers you gave for questions f) and g) (True or False) are also valid for the disk-driver component of an operating system that communicates with a disk, where he disk acts as persistent storage for an application relying on the operating system.
    i) A triggering Entry to a piece of software may be a message from one of its software functional users comprising a header and its 'payload' of other records.

11. The following questions concern the topics dealt with in **sections 3.5.7 to 3.5.11**. Decide whether each statement is 'TRUE' or 'FALSE'.
    a) A functional process is required to print a list of customers showing their name and their total debt owed. Depending on an input parameter, the output can be either in alphabetic sequence of customer name or in the sequence of decreasing total debt. The process has one Exit for this data.
    b) The process in the preceding example has been changed to print two lists of debtor names and contact details: i) of debtors owing amounts greater than $10,000 (to be sent to a debt-collection agency) and ii) all remaining debtors in the sequence of decreasing total debt (for the sales manager). The functional process has two Exits for this data.
    c) A functional process is required to retrieve a data group 'X' that is stored persistently. A Read data movement is needed for this step regardless of whether the process can access persistent storage directly or must obtain the data via a process of another application because of access-control reasons.
    d) Refer to the 'Real-time systems' example of section 1.3.3. A functional process of the control software continuously polls the 200 sensors to check if any of them have detected a hole in the paper moving beneath. If a hole is detected, the process stops the machine. A single Entry of the process is needed for this purpose.
    e) Refer again to the 'Real-time systems' example of section 1.3.3. A functional process 'A' of the control software continuously monitors the quality of paper being produced, adjusting the paper-making machine parameters as necessary. But in this case the process does not poll the hole-detecting sensors. Instead, if a sensor detects a hole, it sends a message to the control system to interrupt the functional process 'A'. This message is an Entry for the process 'A'.
    f) 'Control commands' are Exits, typically sent to actuate hardware device functional users.

g) The functionality of clicking on a menu to select a functional process and to display a ‘blank’ screen for entering data to a functional process does not contribute to the functional size of the process.

h) One error/conformation message (-type) per object of interest (-type) shall be identified to account for all error/confirmation messages that must be issued to a human functional user of a functional process.

12. The following questions concern the topics dealt with in **section 3.5.12,** i.e. the identification of **data movements that must be modified.** Decide whether each statement is ‘TRUE’ or ‘FALSE’

a) The functionality to check the valid size range of an attribute of the data group moved by an Entry must be changed. The Entry should be counted as a modified data movement.

b) A functional process exists to add a new customer’s details (name, address, e-mail, etc.) to a system’s customer file. Before adding the customer details to the file, the process must check if the customer name is already stored in the customer file. A Change Request states that in future the customer address as well as the customer name must be checked. The Entry, Read and Write data movements of the existing process must be counted as modified.

c) A functional process ‘A’ enables the entry and persistent storage of a data group ‘X’. Another process ‘B’ of the same software enables the data group to be retrieved and displayed. A Change Request states that:

- the validation of a non-key attribute of the data group ‘X’ must be changed to accept any alphanumeric character instead of accepting only numeric values;
- the field heading for this attribute must be changed on both the input of process ‘A’ and the output of process ‘B’.

Four data movements must be modified to implement this change request.

d) The correct answer to question c) would be the same if the attribute to be changed were a key attribute of the data group.

13. The following questions concern the topics dealt with in Chapter 4.

a) A project delivered an application of total size 523 CFP and a piece of software of size 35 CFP that was used once to convert a file to the format required by the new application. This piece was then discarded. The customer requested that seven of the new application’s components (of total size 42 CFP) were developed so that they could be re-used by future applications. What would you measure as the size that the project *delivered* to the customer?

b) Referring to question a), suppose the seven components had been developed earlier by other projects and were re-used in developing the new application of total size 523 CFP. What size would you measure that the project had *developed*?

c) An application of size 439 CFP must be enhanced. The FUR for the enhancement specified that five new functional processes of size 27 CFP must be added, 16 data movements of other processes must be changed and two existing functional processes of 9 CFP must be deleted. What would be the size of the work-output of the enhancement project?

d) To satisfy the FUR to delete two functional processes in the previous question c), the developers decided to simply remove the names of the two functional processes from the application’s menu. The code for the two functional processes was left in the

program. What would be a 'fair' measure of the size of the work-output of the developers for this enhancement project for productivity-measurement purposes?

e) For the previous question c), what was the size of the application after the enhancement project was completed?

7.2 Mini Case Studies

7.2.1 The Branch Library System ('BLS').

The requirements for this system were written in the form of User Stories. The following is an extract of some of the Stories; shown in the order in which they were written:

"As a Librarian I want:

1. to create and maintain a book catalogue for all books stocked by the BLS.
2. to add an author's details to the BLS and link the author to all his/her books.
3. to add a new borrower's details to the BLS and send a message to the Central Library System requesting it to issue a Library Member plastic card.
4. a Library Member's plastic card to show the member's name in text and his/her ID in a bar code.
5. to search for all the books in the catalogue by a given author, and to print the list if asked by a borrower.
6. To be able to check-out a borrower and the books they want to borrow.
7. the BLS to send e-mails each night automatically to borrowers about overdue books and the accumulated overdue charges.
8. to set a local limit on the maximum number of books that may be borrowed at any time.
9. not to have to worry about back-ups.
10. to be able to request a report at any time for a given time-period which shows the top-ten most-lent books by title, and the total number of books we lent in the period".

Note: These are intended to be typical User Stories that you might find in practice. So they are not written necessarily all at the same level of granularity, may not use consistent terminology, and do not have all the details needed to properly answer the questions that follow. You may need to make some assumptions to answer some of the questions.

Questions.

a) Which of the User Stories describe a part of a functional process, or one, or more processes?

b) Draw a context diagram for the BLS.

c) Draw a data model showing all the objects of interest about which data must be stored by the BLS and the degree of their relationships to meet the requirements of these User Stories.

d) Analyse and measure User Story 3.

e) Analyse and measure User Story 5.

f) Analyse and measure User Story 6.

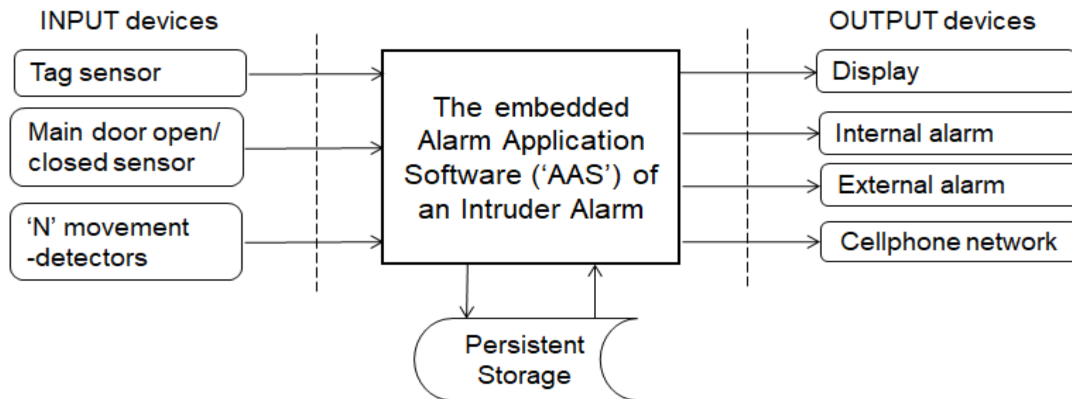
g) Analyse and measure User Story 7.

h) Analyse and measure User Story 10.

7.2.2 The Domestic Intruder (or Burglar) Alarm System.

The main function of the system is, when in a 'set' state, to start an alarm if any movement is detected inside a house or if its main door has been opened.

We do not have a statement of requirements, so we deduce the functionality of the Alarm Application Software ('AAS') available to the house occupants from its user manual. Figure 2.5.2 for the context diagram of the AAS, copied below, shows its hardware functional users.



The AAS has the following capabilities for the purpose of this exercise. It can:

- support the human interface via a 'tag sensor', and a small screen to display information useful to the occupants, e.g. for the reason if the internal alarm has started;
- receive input from a sensor fitted to the main door of the house which informs whether the door is open or not, and from up to 10 identical movement-detector sensors located throughout the ground floor of the house.
- switch on or off two alarms, one internal and the other external (sirens that make a wailing noise);
- send messages to a cellphone app.

The 'tag' is a small device belonging to the house occupants. When an occupant presses a button on the tag close to the tag sensor, the sensor detects the tag's unique ID (or ignores the ID if it is not recognized). An occupant uses the tag to 'set' the system (i.e. to move its state from 'standby' to 'active') or to 'unset' the system.

The 'unset' state: When the system is powered 'on' and the occupants are at home, the system is 'unset', i.e. all hardware devices except the tag sensor and the display are disabled.

'Setting' and 'unsetting' the AAS: The AAS can be 'set' by the occupants using the tag when they are inside the house and intend to move out-of-range of the movement-detectors (e.g. to go upstairs to bed), or if they intend to leave the house via the main door.

When the system is 'set', all the movement detectors and the main door sensor are activated, and the internal alarm is started. This alarm can be stopped by any of the following actions of the occupants:

- They move out of range of the movement detectors within a pre-set 'Exit-time'.
- They open the main door to leave the house and then close it again within the pre-set Exit-time.
- They use the tag if they fail to complete either of the above actions within the Exit-time.

In the first two cases, the AAS remains 'set'. In the last case the AAS is returned to the 'unset' state.

Starting or stopping the alarms when the system is 'set'.

The internal alarm: The internal alarm starts if a movement-detector sensor is activated or if the main door is opened.

The internal alarm can be stopped by using the tag. The system is then 'unset'.

The external alarm: If the internal alarm has been started and the tag is not sensed within a further pre-set 'Wait-time', the external alarm also starts and the AAS sends a simple alert message to the cellphone app.

The external alarm must be stopped after 20 minutes (a legal requirement). However, the internal alarm continues to wail and the system remains 'set' until it is 'unset' using the tag.

Assumptions: As certain functions must be completed within pre-set elapsed times, there must be a timer function. We assume for the purpose of this exercise that functionality to control pre-set times is allocated to a 'countdown-timer' plug-in software component. When needed, the AAS sends a pre-set time to the countdown-timer component. The latter returns a 'countdown time ended' message when the pre-set time is reached. *(With this assumption, the countdown-timer component must be added to the context diagram of the AAS as a functional user of the AAS.)*

The AAS must have some random-access memory to hold the cellphone number, the tag ID, the pre-set times, standard messages for the display and for the cellphone, etc. For this exercise. ignore the functionality needed to maintain these data.

When the AAS system waits in a 'set' state, it could either receive signals from its sensors, or it must poll the sensors to obtain their state. We do not know which process is used but it does not matter for the functional size measurement.

Questions:

a) Analyse and measure the AAS functional process that changes the status of the AAS from 'unset' to 'set' when the occupants intend to go upstairs or to leave the house.

b) Analyse and measure any AAS process(es) that will stop the alarm if the occupants *succeed* in their intention to go upstairs before the end of the Exit-time, or to leave the house with the main door closed.

c) What happens if the occupants *fail* in their intentions to go upstairs or to leave the house *before the end of the Exit-time*? Analyse and measure the AAS functional process that the occupants need to stop the internal alarm from wailing and to 'unset' the system.

d) List all the triggering events that the AAS must respond to (hence identify the complete set of functional processes)

e) From the list produced in answer to question d) and the size(s) measured in answer to questions a) to c), estimate the total size of the AAS in units of CFP.

7.3 Answers and discussions of the Questions of section 7.1

1. a) – c): TRUE.

d) TRUE, assuming the data collection system is a type of 'Internet of Things' system.

e) FALSE. The functionality of weather forecasting systems is totally dominated by complex mathematical algorithms. A size in units of CFP would not be very meaningful as it does not account for the algorithms.

f) TRUE, except for any specialized components that rely on machine-learning and/or complex mathematical processing, e.g. speech recognition or gestures on touch screens. These components are generally used by apps; they are not usually part of an app.

g) TRUE if the software provides e.g. pre-programmed enquiries chosen from a menu, or enables searching of records that match input parameters. FALSE if developing the

search software involves developing complex mathematical algorithms such as for neural networks.

h) FALSE. Video-game development is a largely creative process, exploiting pre-built animation routines etc.

i) – j) TRUE.

k) YES if the premiums are obtained from a look-up table, or by simple algorithms. Maybe only PARTLY TRUE, if the premium-calculation depends on a complex mathematical algorithm.

2. a) FALSE. In principle, and usually in practice, you can extract or derive at least some concepts by examining <u>any</u> software artefact.

b) TRUE. Rules for extracting concepts from local software artefacts are beyond the COSMIC Method. COSMIC can only give examples. (But much of the time, rules are not needed; the concepts are obvious.)

c) FALSE. The functional size of a piece of software depends only on the total count of its data movements. See the principles of the Generic Software Model.

d) TRUE.

e) FALSE. See the footnote of section 1.3.3 for an example where knowing the *relative* number of occurrences of a concept can be relevant to the process of measuring a CFP size.

3. a) TRUE. The statement is a Quality NFR. It may result in specific FUR that can be measured in CFP, but it does not specify any functionality directly.

b) FALSE. Statements of FUR should be totally independent of implementation requirements, which are NFR

c) FALSE. This is a NFR. The statement has no direct implications for application software functional requirements; it might be satisfied entirely by hardware. It might (eventually) result in some FUR of the application.

d) FALSE? The Company's standard login control is probably in another layer than the application being measured, with no implications for the FUR of the application.

e) FALSE. This is a project governance requirement. It is a constraint on the project; it says nothing about the software or how it should be developed.

f) FALSE? This is likely a functional requirement for the *system* (of which the software is part) that will be allocated to hardware. If so, the switch to and from the back-up generator is likely to have no implications for the FUR of the software. But you should investigate further before this assumption about the allocation of functionality is true or not.

4. a) FALSE Do not confuse the scope and the boundary. They are different concepts.

b) TRUE. If the software extends over more than one layer, you must define different measurement scopes for the pieces of software in each layer.

c) TRUE. The nature of the architectural relationship between the two pieces is immaterial to the CFP measurement. They exchange data across a boundary.

d) FALSE. Persistent storage is an abstract concept available to all functional processes. Data that is Written, i.e. made persistent, by one functional process is available to be Read by any other process.

e) TRUE. Any piece of software can consist of an assembly of components at different levels of decomposition. But if the purpose is to measure the size of the whole piece of

software, then the scope of the measurement is the whole piece; its internal structure is irrelevant to the measurement of the total size.

f) FALSE. The FUR for an enhancement project may affect several independent pieces of software. Use the Software Context model principles to determine the scope of any one of the pieces of software for which enhancements must be measured. The method does not limit the FUR for enhancements that may be measured.

5. a) could be TRUE if we assume that all HR staff can access all of the HR database. This is unlikely for privacy reasons. More likely, the statement is FALSE because not all the various categories of HR staff will have the right to access all of the personal data, e.g. salary data. This could be significant for the scope of a size measurement that was limited to the view of certain categories of HR staff, each with their own functional processes. The result would be the need to define more than one functional user type.

b) FALSE. Cows cannot interact with software. The hardware devices that, for example, detect the cow ID (from an embedded chip) and the devices that measure the quantity of the milk she supplies on each visit for milking are functional users of the automated milking system.

c) FALSE because the sub-sets of functionality may overlap.

d) TRUE.

e) FALSE. A functional user is a user of software, not of hardware. The hardware device Y is a functional user of the software component X.

f) TRUE.

6. a) FALSE, This statement confuses 'level of granularity' with 'level of decomposition'.

b) TRUE. As FUR are analyzed into more detail, a group of functional users identified at a higher level of granularity may need to be distinguished at the lower level. For example, a functional user labelled 'Central Systems' at a high level of granularity of some FUR may need to be distinguished as separately-identifiable software functional users at a lower level of granularity of the FUR.

c) FALSE, though it is often TRUE, it is never 'safe to assume' the level of granularity of statements of FUR

d) TRUE – by definition for an existing, operational system.

7. a) is a **group** of functional processes. (Remember the 'CRUD' acronym if you see the word 'maintain'.)

b) is a **single** functional process. (The triggering event is the end of a day.)

c) is **part** of a functional process. It is a rule. No event is mentioned on when foreign income shall be credited

d) is a **single** functional process. (The triggering event is the one-second 'tick'

e) The FUR to stop the belt definitely specifies a **single** functional process. But it is not clear from the FUR if the functionality to time whether an emergency stop button has been pressed for two seconds is allocated to hardware or to software. If the latter, this would require a separate timing functional process, in addition to the process to stop the belt.

f) The FUR specify a **group** of functional processes, e.g. for on or off commands, flashing of headlights, full-beam on or off, etc.

8. a) FALSE. The last phrase of the definition is not 'data entered by its triggering Entry', but 'data entered as a result of a triggering event'.

b) TRUE. Think of an earthquake detected by seismometers around the world.

c) FALSE. What appears to be a real-world event depends on the FUR for the software that must process data about the event. (A sporting match may be a single event for a newspaper that reports the match result. The same match may be reported as a series of events by an on-line news-feed.)

d) TRUE.

e) TRUE. A functional process exists entirely within the scope of one piece of software.

f) TRUE.

g) FALSE. The start of a batch process (or of an on-line system) is not part of the FUR of the functional process of the application being started. Each functional process of an application has one triggering Entry moving one data group.

h) FALSE. All data movements required for each functional processes must be accounted for in the size of each functional process.

i) FALSE. You only need to identify the number of data movements needed by the process to meet its FUR. You do not need to identify the different processing paths, which will depend on the input data values.

9. a) FALSE. An object-class always includes its 'methods' (that may or may not apply to a single object of interest).

b) TRUE

c) TRUE for the data group (user ID and password) moved by the triggering Entry of the typical login functional process.

d) TRUE. The value of the sales in dollars, is unlikely to be a physically-identifiable pile of money.

e) FALSE. Data groups may be moved in Entries and Exits that are never persistently stored. So these would not appear in a 'definition of all stored data'. Examples include objects of interest that are the subject of data groups derived by enquiring on stored data (known as 'transient' data groups), or that simply pass through a software component without being stored.

f) FALSE. What often happens as high-level FUR are worked out in more detail to lower levels of granularity is that the FUR reveal <u>more</u> objects of interest, rather than objects of interest at lower levels of granularity. Example: an 'employee' object of interest identified at a high level of granularity does not change into some other object of interest as FUR are worked out at lower levels of granularity.

g) FALSE. 'Country of birth' is an attribute in the data group 'applicant data' that describes the object of interest 'applicant'. However, 'country' must be an object of interest for the functional processes of the PAS that maintain the table of standard country names. NOTE: There is nothing absolute about what are objects of interest. It depends on the FUR. Remember the aphorism 'one man's attribute is another man's entity'.

h) TRUE. The six objects of interest could be named: job-seeker, job-seeker qualification. Job-seeker employment-history, employer, job vacancy, interview. All these objects of interest have different frequencies of occurrence. The Agency could also be an object of interest on a web-site which displays 'About us' information.

10. a) TRUE

b) FALSE. An Entry accounts for the data manipulation associated with entering and validating the entered data. But validation may require other data movements. Example: the process to enter a 'user-name' when signing-up to use a new system may require a Read of the file of user names to check if the name has already been taken by an existing user. Validation may also result in issuing error/confirmation messages, i.e. one more data movement.

c) FALSE. Because the COSMIC method requires a logical model of software functionality, the functionality to display physical screens for data entry is of no interest.

d) TRUE.

e) FALSE. A single line of a report might, for example, list the value of ‘sales-per-month’ and, at the end of the line, the ‘total-sales’ for the time period. These are attributes of two different data groups, hence describing two different objects of interest. So producing this line of the report requires two Exits.

f) TRUE.

g) TRUE.

h) TRUE. In all these cases f), g) and h), the hardware or firmware is a functional user of the device-driver software. The device-driver software must communicate with the device via the manufacturer’s standard interface, across a boundary via Exits and Entries. This applies regardless of whether the device is a printer or disk-drive, etc.

i) FALSE. The triggering Entry is the header of the message. The ‘payload’ comprises one or more additional Entries.

11. a) TRUE. The Exit concerns the object of interest ‘debtor-customer’. The sorting of the data into one sequence or the other is a data manipulation sub-process that is invoked, or not, depending on the value of an input parameter.

b) TRUE. The process has two Exits even though both Exits move a data group describing the same object of interest (debtor-customer), as it is a functional requirement for this process to produce reports for two types of functional users.

c) FALSE. A functional process needs a Read to retrieve a data group from persistent storage within its boundary. But to obtain the data group via another functional process needs an Exit/Entry pair, crossing the boundary between the two processes.

d) TRUE

e) FALSE? An interrupt is normally handled by the real-time operating system, not by the functional process that is interrupted. So the operating system may switch control to another process to action the ‘hole-detected’ message.

f) FALSE. A ‘control command’ is a term used by the COSMIC method only for interactions of human functional users with software that do not involve entering or receiving data about an object of interest. Such commands are not measured. The term has its normal meaning in any other circumstances, such as an Exit to a hardware actuator.

g) TRUE.

h) FALSE. One Exit must be counted for <u>all</u> error/confirmation messages that must be issued by a single functional process to a human user.

12. a) TRUE.

b) FALSE? The data movements that enter the customer details, search the customer file and write the customer details are (probably) unaffected by the required change. But the error/confirmation message will probably need to be modified. So one data movement must be changed.

c) FALSE. The Entry data movement of the functional process ‘A’ and the Exit data movement ‘B’ must be changed, as well as the error/confirmation message. The Write of ‘A’ and the Read of ‘B’ should not be affected by the Change Request. So three data movements must be changed.

d) TRUE. Again, the Write and the Read should not be affected by the Change Request.

13 a) Assuming that the application and the data-conversion software both reside in the same application layer, the project delivered 523 + 35 = 558 CFP of application software

and 42 CFP of reusable components. (It makes no sense to add 558 and 42 together as the sizes are of software at different levels of decomposition.)

b) The seven components of size 42 CFP form part of the application but did not have to be developed. However the application must call the components in order to be able to use their functionality and this needs a minimum of seven Exit/Entry pairs, i.e. 7 x 2 = 14 CFP. So the minimum size of the software developed was 523 – 42 + 14 +35 = 530 CFP.

c) The work-output of the enhancement project was 27 + 16 + 9 = 52 CFP.

d) If the goal is to have a 'fair' measure of the work-output of deleting the two functional processes, there is an argument for not counting the CFPs for the deletion because In this case the two functional processes were 'disabled' by removing them from the application menu; they were not actually deleted. A 'fairer' measure of the work-output would therefore be 27 + 16 = 43 CFP. (This is an opinion; to give a precise answer, the FUR should make clear what was actually intended by the requirement to 'delete'.)

e) The size of the application after the enhancement was 439 + 27 – 9 = 457 CFP.

7.4 Analysis and discussion of the Mini Case Studies of section 7.2

7.4.1 The Branch Library System

The first thing to note about the User Stories is that they do not use consistent terminology, which is very common in real-life Stories. In particular:

- 'book' can mean a unique work defined by an International Standard Book Number (ISBN), or a physical copy of the book. To distinguish these we will use the words 'book' and 'book-copy',
- a 'Library Member' also becomes a 'borrower' when he/she borrows a book-copy;
- When the Branch Library 'lends' a book-copy that is the same transaction as when a Member 'borrows' a book-copy.

a) Story 1 defines a group of functional processes to Create, Read, Update and Delete books in the catalogue. (Note the word 'maintain', and remember the 'CRUD' acronym).

Story 2 could be satisfied by one or by two processes. The latter could be e.g. 'Add author' and 'Link author to his/her book(s)'.

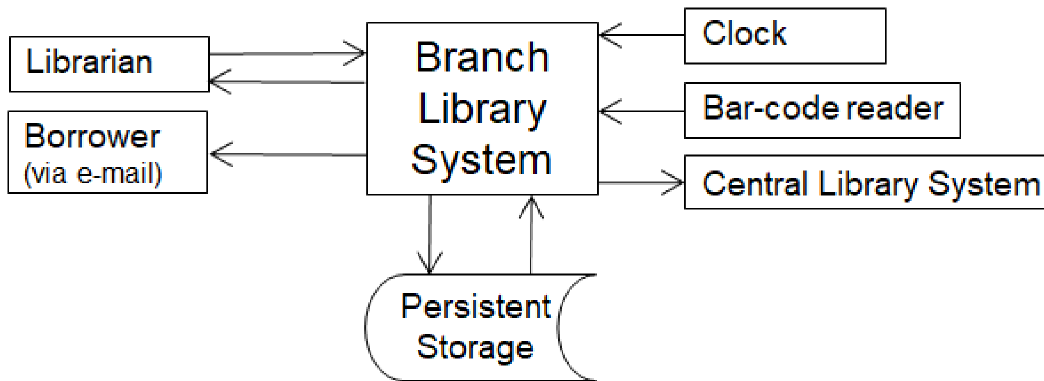
Stories 3, 5, 6, 7 and 10 each define a complete single process. (The 'print' part of Story 5 is a separate process but is probably a function provided by the operating system, so not part of the BLS.)

Story 4 specifies the Member ID data attributes and their formatting as they must appear on a member's plastic card, so Story 4 is part of a process. Story 4 also implies that the BLS must interface with a bar-code reader, which is a Non-Functional Requirement (NFR).

Story 8 is a rule (i.e. part of a process). This Story could require re-work of e.g. the 'Check-out a borrowing' process, if the latter had already been implemented via Story 6.

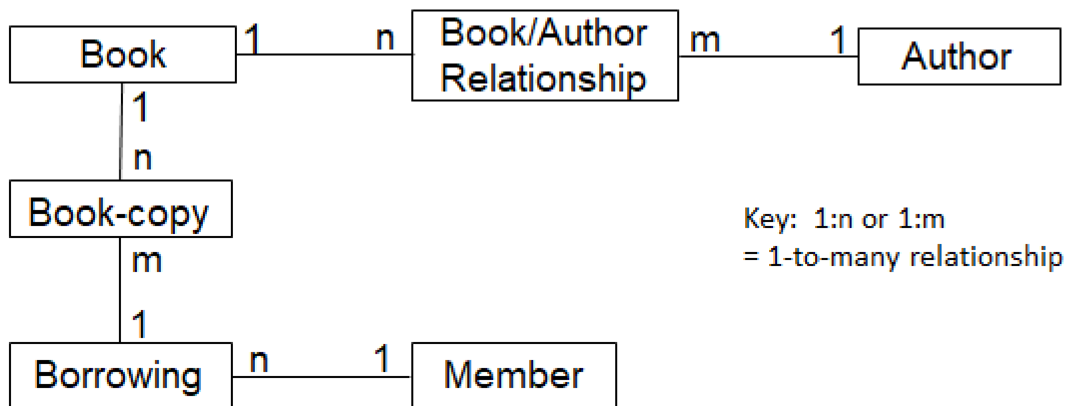
Story 9 concerning a requirement for automatic back-ups seems like a NFR of the BLS that could be provided by the operating system.

b) The context diagram for the Branch Library System is shown below.



Note: A clock is needed to start the automatic overnight sending of e-mails to borrowers with overdue books, as per Story 7.

c) The data model for the objects of interest described in the User Stories of the Branch Library System is shown below.



The objects of interest and their possible attributes are shown below. Key attributes are underlined.

Book (<u>ISBN,</u> Book-title, publisher name, date of publication, UDC Code, etc.)

Author (<u>Author name</u>, date of birth, etc.)

Book/Author Relationship (<u>ISBN</u>, <u>Author name</u>)

Book-copy (<u>ISBN</u>, <u>BLS Copy-number</u>, date of purchase, purchase price, etc.)

Member (<u>Member ID</u>, member-name, title, home-address, e-mail address, etc.)

Borrowing (<u>Date of borrowing</u>, <u>Member ID</u>, <u>ISBN</u>, <u>BLS Copy-number</u>, borrowing-period)

Note that the 'book catalogue' referred to in User Story 1 is the set ('or file') of books stocked by the library branch.

Physically, the book-copy key attributes are printed as a bar code on a label inside the book-copy.

A book may have one or more authors and an author may write one or more books. The 'book/author relationship' object of interest simply records the book/author link.

d) User Story 3 defines the functional process 'Register new member'. The triggering event is that a person wishes to join the Branch Library. The size = 5 CFP.

DM	Functional User / (object of interest)	Data Group Name (Comment)
Entry	Librarian (member)	Member details (triggering Entry)
Read	(member)	Member name + address (assume BLS needs to check that the person is not already a member)
Write	(member)	Member details
Exit	Central Library System (member)	Member name (request to produce member plastic card by e-mail)
Exit	Librarian (Errors)	Error/confirmation message (in case member details fail validations, etc.)

e) User Story 5 defines the functional process 'Search for books in the catalogue by author'. (The analysis below ignores the possible need for a general search tool to handle input of mis-spelled author names, incomplete book titles, etc.) Size = 8 CFP

DM	Functional User / (object of interest)	Data Group Name (Comment)
Entry	Librarian (author)	Author name (triggering Entry)
Read	(author)	Author details
Exit	Librarian (author)	Author details (for the Librarian to verify the correct author has been found)
Entry	Librarian (book)	Book title
Read	(book)	Book details (to obtain the book ISBN)
Read	(book/author relationship)	Book ISBN / Author name
Exit	Librarian (book/author relationship)	Book title (one or more occurrences for the given author)
Exit	Librarian (errors)	Error/confirmation message (in case entered Author name or Book title fail validations, etc.)

f) User Story 6 defines a functional process 'Check-out a borrowing'. Size = 6 CFP.

DM	Functional User / (object of interest)	Data Group Name (Comment)
Entry	Bar-code reader (member)	Member ID (triggering Entry)
Read	(member)	Member details (assume needed to check that the member's plastic card is valid)
Entry	Bar-code reader (book)	Book ID (ISBN, BLS Copy-number)
Read	(book)	Book details (assume needed to check that the book bar-code is valid, repeated for each book being borrowed at this time)
Write	(borrowing)	Borrowing details
Exit	Librarian (errors)	Error/confirmation message (in case errors in entered data)

g) User Story 7 defines the functional process 'Send overdue books message', to be executed automatically overnight. The analysis assumes that the 'overdue charges' are

calculated by the software for each overdue borrowing. Size = 6 CFP. (If a charge must be calculated from data in a look-up table, an extra Read would be needed.)

DM	Functional User / (object of interest)	Data Group Name (Comment)
Entry	Clock (time to start process)	Clock 'tick' (triggering Entry)
Read	(borrowing)	Borrowing details
Read	(member)	Member details (to get member's name, e-mail address)
Exit	Member (member)	Overdue book message, i.e. the e-mail header (Attributes: member's e-mail address, explanatory text, total overdue charges, etc.)
Exit	Member (book - that is overdue)	Overdue book message-item (for each book that is overdue)
Exit	Librarian (errors)	Error/confirmation message (in case errors occur in the process)

h) User Story 10 defines a functional process 'Report books lent in a given time-period. Its size = 5 CFP.

DM	Functional User / (object of interest)	Data Group Name (Comment)
Entry	Librarian (report time-period)	Report time-period (e.g. start and end dates) (triggering Entry)
Read	(borrowing)	Borrowing details
Read	(book)	Book details (to obtain the book title)
Exit	Librarian (book)	Book title (for the top 10 most-lent books)
Exit	Librarian (the set of all books lent in the report time-period)	Count of books lent in the Report time-period (Attributes: time-period definition, the count)

Note that the sorting of the borrowings to find the top-ten most borrowed books is data manipulation, which is not measured.

7.4.2 The Domestic Intruder (or Burglar) Alarm System.

Note that for this real-time application, when an Entry is received from a functional user or an Exit is sent to a functional user, the object of interest of the data group received or sent and the functional user are the 'same thing'.

In the analyses shown below, therefore, the object of interest is only shown after the functional user when they are not the 'same thing'

a) The size of the functional process 'Set the alarm system' is 9 CFP.

DM	Functional User / (object of interest)	Data Group Name (Comment)
Entry	Tag sensor	Tag ID (triggering Entry)
Read	(tag)	Tag ID (Process terminates if wrong tag ID)
Exit	Internal alarm	Start wailing

Exit	Movement-detector	Activate movement-detector (‘N’ occurrences)
Exit	Main door open/closed sensor	Activate main door sensor
Read	(exit-time)	Exit time
Exit	Countdown-timer component (exit-time)	Start the Exit-time countdown
Read	(message)	Message for display (e.g. ‘Alarm set’)
Exit	Display (message)	Message for display

b) Once the AAS is set, we assume two functional processes can stop the internal alarm:

- either the ‘Exit-time ended’ process starts if the occupants have gone upstairs,
- or the ‘Main door closed’ process starts if the door is closed before the end of the Exit-time.

(Note, however, we assume that the alarm would immediately re-start after completion of either of these two processes if a movement were detected, or if the main-door sensor continued to report that it was open after the end of the Exit-time.)

The size of the ‘Exit-time ended’ process = 4 CFP.

DM	Functional User / (object of interest)	Data Group Name (Comment)
Entry	Countdown-timer component (exit-time)	Exit-time countdown ended (triggering Entry)
Exit	Internal alarm	Stop internal alarm
Read	(message)	Message for display (e.g. ‘Alarm set’)
Exit	Display (message)	Message for display

The size of the ‘Main door closed’ process = 5 CFP

DM	Functional User / (object of interest)	Data Group Name (Comment)
Entry	Main door open/closed sensor	Main door closed (triggering Entry)
Exit	Countdown-timer component (exit-time)	Stop countdown-timer
Exit	Internal alarm	Stop internal alarm
Read	(message)	Message for display (e.g. ‘Alarm set’)
Exit	Display (message)	Message for display

c) To stop the internal alarm and to unset the AAS in any circumstances, the occupants must use the tag to start the ‘Unset alarm’ process. Size = 8 CFP

DM	Functional User / (object of interest)	Data Group Name (Comment)
Entry	Tag sensor	Tag ID (triggering Entry)
Read	(tag)	Tag ID (Process terminates if wrong tag ID)
Exit	Internal alarm	Stop internal alarm
Exit	Movement-detector	Deactivate movement-detector (‘N’ occurrences)

Exit	Main door open/closed sensor	Deactivate main door sensor
Exit	Countdown-timer component (wait-time)	Stop timing (in case Wait-time timing had been started)
Read	(message)	Message for display (‘System unset’)
Exit	Display (message)	Message for display

c) There are eight triggering events, namely:
- Set the AAS
- Unset the AAS
- Movement detected
- Main door open detected
- Exit-time ended
- Main door closed
- Start external alarm
- Stop external alarm

e) From the analysis so far, there are two ‘large’ functional processes of average size 8.5 CFP and two ‘small’ processes of average size 4.5 CFP. From a quick examination, the other four processes also seem to be ‘small’ processes. A quickly-estimated approximate size of the AAS would therefore be 2 x 8.5 + 6 x 4.5 = 44 CFP.

REFERENCES

All the COSMIC documents listed below, sometimes including translations into other languages besides English, can be found on www.cosmic-sizing.org .EXCEPT where a reference is given to another source.

[1] For COSMIC certification exam details see https://cosmic-sizing.org/certification/

[2] 'Measurement Manual' (The COSMIC Implementation Guide for ISO/IEC 19761: 2017), version 4.0.2.

[3] Guideline for Sizing Real-time Software, v2.

[4] Guideline for Sizing Business Application Software, v1.3.

[5] Guideline for Sizing Data Warehouse and Big Data Application Software.

[6] Guideline for Sizing Service-Oriented Architecture Software.

[7] 'Glossary of terms for Non-Functional Requirements and Project Requirements used in software project performance measurement, benchmarking and estimating', 2015, published by COSMIC and IFPUG.

[8] Non-functional Requirements and COSMIC sizing: a Practitioners Guide

[9] Guideline for 'Measurement Strategy Patterns'.

[10] 'An approach for a fast of web-based APIs supported by functional size measurement with COSMC FP.', Schmietendorf, A., Schmidt, S., Nadobny, K., Hartenstein, S., Haarlem 2019, https://www.iwsm-mensura.org/wp-content/uploads/2019/10/R15.45-Sandro-Hartenstein-Cost-Estimation-of-web-based-APIs.pdf

[11] Early Software Sizing with COSMIC: a Practitioners Guide.

[12] Guideline for assuring the accuracy of measurements.

[13] 'Experience of using COSMIC sizing in agile projects', Abran, A., Demirors O., Symons C.R.

[14] 'COSMIC FSM Adoption at Eurofins', Aravind Gundarao, IWSM-Menura, Haarlem 2019, https://www.iwsm-mensura.org/wp-content/uploads/2019/11/COSMIC-Adoption-At-Eurofins.pdf

[15] 'The performance of real-time, business application and component software projects: an analysis of COSMIC-measured projects in the ISBSG database'. www.isbsg.org

[16] 'A 'scatter-gun' or 'rifle-shot' approach to managing and estimating software processes?', Symons, C.R., IWSM-Mensura Conference, Beijing 2018

[17] 'Progress with COSMIC in China', (experience of the China Continent Property and Casualty Insurance Company Ltd using COSMIC), reported by Dylan Ren, IWSM-Mensura Conference, Beijing 2018.

[18] 'Manage the automotive embedded software development cost and productivity with the automation of a functional size measurement method (COSMIC)', Oriou, A., Bronca, E., Bouzid, B., Guetta, O., Guillard. K., IWSM-Mensura Conference, Rotterdam, 2014

[19] 'Web effort estimation: function point analysis vs. COSMIC", S. Di Martino, F. Ferruci. C. Gravion, F. Sarro, Information and Software Technology 72 (2016) 90–109

[20] 'Embedded software memory size estimation using COSMIC: a case study,' Gencel C., Stern S., IWSM/Metrikon/Mensura conference, Stuttgart, 2010

[21] 'A practical approach to size estimation of embedded software components' Lind, K., Heldal, R., IEEE Transactions on Software Engineering, 2012, Volume 38, Issue 5.

[22] 'Investigating Functional and Code Size Measures for Mobile Applications: A Replicated Study', F. Ferrucci, C. Gravino, P. Salza, F. Sarro, in Proceedings of the 16th International Conference on Product-Focused Software Process Improvement (PROFES 2015), pp. 271-287.