The COSMIC Functional Size Measurement Method
Version 4.0.1

Guideline for Sizing Real-time Software

VERSION 1.1
April 2015

Acknowledgements

Editors and reviewers of version 1.1 of this Guideline for versions 4.0/4.0.1 of the COSMIC method		
Peter Fagg Pentad United Kingdom	Arlan Lesterhuis*, Netherlands	Hassan Soubra Ecole Supérieure des Techniques Aéronautiques et de Construction Automobile France
Charles Symons*, United Kingdom	Frank Vogelezang Ordina The Netherlands	Chris Woodward Chris Woodward Associates Ltd. United Kingdom

* Editors of this guideline

Authors and reviewers of version 1.0, 2012		
Alain Abran, École de Technologie Supérieure, Université du Québec, Canada	Juan J. Cuadrado-Gallego, University of Alcalá, Madrid, Spain	Jean-Marc Desharnais*, École de Technologie Supérieure, Université du Québec, Canada
Cigdem Gencel, Free University of Bozen/Bolzano, Italy	Arlan Lesterhuis*, Netherlands	Kenneth Lind, Viktoria University, Sweden
Bernard Londeix*, Telmaco Ltd, United Kingdom	François Perron, Pyxis Technologies, Canada	Charles Symons*, United Kingdom
Sylvie Trudel, Pyxis Technologies, Canada	Frank Vogelezang, Ordina, Netherlands	Steve Webb, independent consultant, United Kingdom

Version Control

The following table gives the history of the versions of this document.

DATE	REVIEWER(S)	Modifications / Additions
June 2012	COSMIC Measurement Practices Committee	First version 1.0 issued
April 2015	COSMIC Measurement Practices Committee	Version 1.1 brought in line with the Measurement Manual v4.0.1

Foreword

Purpose of this guideline and relationship to the COSMIC Measurement Manual

The purpose of this guideline is to help those working in the domain of real-time software to map the concepts they typically use to determine and model the requirements of real-time software, to the concepts of the COSMIC method of measuring a functional size of software. The guideline also provides many measurement examples and illustrative cases.

The guideline is hence an aid-to-translation from the terminology used by real-time software practitioners to the terminology of the COSMIC method. No additional principle or rule is required to apply the COSMIC method to the real-time domain, beyond those that are provided in the COSMIC Measurement Manual [1].

Intended readership of the guideline

The guideline is intended to be used by anyone involved in defining, specifying, developing and managing software products in the real-time domain. This includes the members of the software metrics team and/or developers who have the task of measuring the functional size of real-time software according to the COSMIC method. It should also be of interest to those who have to interpret and use the results of such measurements for project performance measurement, software contract control, project effort estimation, etc. The guideline is not tied to any particular real-time software development methodology or life-cycle model though some examples refer to specific real-time requirements determination or modeling methods. Note that COSMIC does not endorse any particular method or tool.

Readers of this guideline are assumed to be familiar with the COSMIC Measurement Manual [1]. For ease of maintenance, there is little duplication of material between that document and this guideline.

Scope of applicability of this guideline

This guideline concerns the measurement of 'real-time' software, where we use this term in a broad sense. According to Wikipedia, 'a system is said to be *real-time* if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed. Real-time systems, as well as their deadlines, are classified as 'hard', 'firm' or 'soft' depending on the consequence of missing a deadline;'

For the purpose of this Guideline, we also include any software whose operation is controlled by a clock mechanism. The COSMIC method can be used to measure the functionality of all these various types of 'real-time' software. (However, it should be noted that a specific timing constarint, or 'deadline' such as 'all commands must be satisfied within 1 millisecond' is a non-functional requirement. COSMIC functional sizing can measure any functionality needed to achieve this constraint but the specific numerical value of the constraint (1 millisecond, or 1 microsecond, or whatever) does not actually affect the software functional size.)

Examples of real-time software include the monitoring and control of industrial systems, automated acquisition of data from the environment and scientific experiments, the monitoring and control of vehicle systems such as engines, ventilation, collision-avoidance, etc. and of household appliances. On the large scale, real-time systems control the world's telephone networks, individual aircraft and air traffic, power plants and such-like. Some software systems such as hotel or airline reservation systems may be described as hybrids of business application software and real-time software, because they must process enquiries and bookings with real-time constraints. Finally, middleware and infrastructure software such as operating systems provide basic tasks and services for real-time applications and hence operate within real-time constraints.

Introduction to the contents of the guideline

Chapter 1 discusses the characteristics of real-time software systems, the way their requirements are stated and how they can be mapped to COSMIC method concepts. Chapter 2 deals with the measurement strategy and in particular with identifying the functional users of the software to be measured. Chapter 3 discusses the mapping and measurement phases. Chapter 4 presents a number of illustrative cases.

For definitions of the terms of the COSMIC method in general, please refer to the glossary in the Measurement Manual [1]. Terms specific to the real-time software domain are defined in a separate glossary at the end of this guideline. Note that the literature on information technology uses a number of terms that are used with various meanings, but which are defined in the COSMIC method with very specific meanings. The Measurer must therefore be careful to correctly apply the terminology of the COSMIC method when using this guideline.

This version 1.1 of this Guideline differs from the v1.0 published in June 2012 mainly in that some of the descriptions of how to apply the COSMIC method have been improved in line with versions 4.0 and 4.0.1 of the method, plus some editorial improvements and two more examples have been added in section 4.7. Some errors in an example 3 in section 2.2 have been corrected.

For a full list of the significant changes that have been made, see Appendix A.

Table of Contents

1

MAPPING REQUIREMENTS OF REAL-TIME SYSTEMS SOFTWARE TO COSMIC CONCEPTS

The purpose of this chapter is to relate the terminology and concepts from the domain of real-time software, as used in various methods of expressing real-time system requirements, to the concepts of the COSMIC functional size measurement method. If a functional size must be measured from the artefacts of some existing operational software, the Measurer should be able to use this same mapping of concepts to reverse engineer to the original functional user requirements (FUR), expressed in COSMIC terms, which can then be sized.

Recalling the key features of the COSMIC method, a measurement should proceed in three phases, see [1]:

In the *Measurement Strategy* phase, the aim is to determine the purpose of the measurement, hence the scope of the software to be measured and its functional users, i.e. the people or 'things' that are the senders or intended recipients of data to/from the software to be measured.

The level of granularity of the requirements and the level of decomposition of the software to be measured are also determined in this phase.

In the *Mapping* phase, the aim is to map the FUR of the software to the concepts of the COSMIC method. [Note that from v4.0 of the COSMIC method, we use 'FUR' to mean only the functional user requirements that are completely defined so that a precise COSMIC functional size measurement is possible. For situations where requirements have been specified at a 'higher' level of granularity, i.e. they have not yet evolved to a level of detail where a precise COSMIC size measurement is possible, we will use 'requirements' or 'functional requirements', as appropriate.]

The *Mapping* phase has two main steps.

- Identify the 'triggering events' detected by (or generated by) the functional users that the software must respond to, and hence the corresponding 'functional processes' (see section 3.1)
- Identify the 'objects of interest' and 'data groups' referenced by the piece of software to be measured and hence the 'data movements' (Entries, Exits, Reads and Writes) in each functional process (see section 3.2)

In the *Measurement* phase (see section 3.3), the functional size of a piece of software is measured by counting the total number of data movements summed over all its functional processes. The functional size of a change to the requirements is measured by counting the total number of data movements that must be added, modified or deleted to satisfy the change requirement.

Note that whenever we mention 'triggering events', 'functional processes', 'data movements', etc., we mean 'types' of these, *not* 'occurrences' (see the Measurement Manual [1], section 1.3.3).

1.1 Characteristics of real-time systems software

Some of the characteristics of real-time systems relevant to measuring the functional size of the software are treated in the following sections.

1.1.1 Event-driven systems

Real-time software is often characterized as 'event-driven', i.e. its functionality must respond to events with real-time constraints. Its behaviour can be illustrated as a finite state machine where each event that the software must respond to may affect the state(s) of the software. The state does not necessarily change; for instance an enquiry functional process leaves the state of the machine unchanged on completion.

A key concept of the COSMIC method is that an event is sensed by, or in real-time software sometimes generated by, a functional user of the software being measured. A functional user

communicates the occurrence of an event to the software by sending a data group that is moved into a functional process by an Entry data movement, which triggers the functional process to start executing. This is shown in the following diagram taken from the COSMIC Measurement Manual [1].

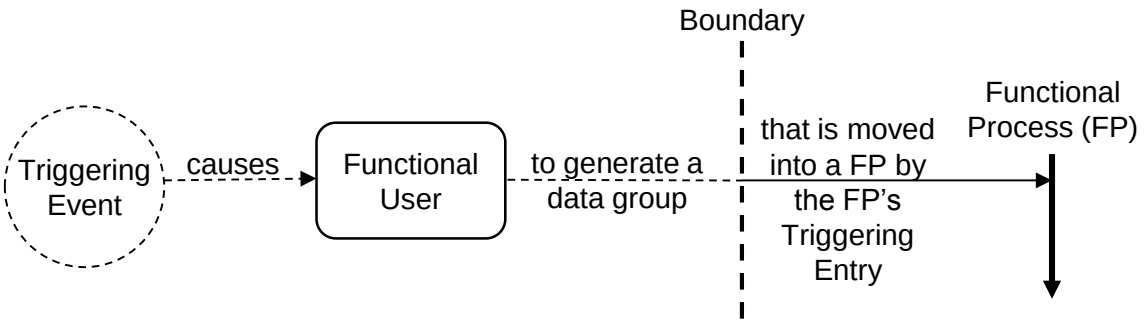

Figure 1.1 – Relation between triggering event, functional user and functional process

Three examples of triggering events help to explain the relationships shown in Figure 1.1:

EXAMPLE 1: A sensor detects a stimulus to which the software must respond

- *the triggering event is the stimulus that the sensor is designed to detect*
- *the functional user is the sensor*
- *the sensor generates and sends a message (a data group), which is moved into a functional process by its triggering Entry data movement, informing that the event has occurred; this message may also carry other data about the triggering event.*

EXAMPLE 2: A piece of software A must pass a request to a piece of software B for a service

- *software A effectively generates the triggering event when it needs the service from software B by generating the request for service (a data group that provides the input data needed for the service)*
- *software A is the functional user of software B*
- *the request for service message is moved into a functional process in software B by its triggering Entry; the functional process can then provide the service*

EXAMPLE 3: A piece of software must execute a control process each time a clock 'ticks'

- *the clock effectively generates the triggering event by generating a 'tick' (a data group)*
- *the clock is a functional user of the software*
- *the 'tick data group' is moved into a functional process by its triggering Entry to start its task.*

In all three examples the 'task' (the service) that the software must undertake, to respond to a triggering event, is a 'functional process', which is a sequence of data movements that is not complete until it has done all that is needed to meet its FUR for all possible responses to its triggering Entry.

Note some important points:

- Each functional process is independent of any other functional process, However, there can be various cardinalities (1:n, 1:1, or n:1) along the chain of triggering event – functional user – data group - functional process of Figure 1.1 (see the Measurement Manual [1] for examples).
- It may be that a series of functional processes can only occur in a particular sequence but this does not affect the fact that each functional process must be separately triggered. Example: a 'stop process' cannot occur before a 'start process' but each process is triggered by a separate event.
- A measured functional size does not take into account a) any specific real-time timing constraint, e.g. that a response is required in less than one millisecond, as this is a Non-functional requirement (however, see section 4.2 for the measurement of timer functionality), nor b) that some functional processes may be triggered by 'routine' triggering events, when the software is waiting for the triggering Entry, and others by 'exceptional' triggering events that give rise to interrupts (see next section 1.1.2).

1.1.2 Interrupts

An Interrupt Message is generated when a Functional User detects an event that requires the current activity of a system to be changed[1]. If a functional process has been triggered and started executing but has not yet terminated normally, an Interrupt Message may have one of two possible consequences.

- If interrupts are handled by an interrupt handler outside the scope of the software being measured, e.g. in another layer than that of the software being measured, then by definition the functional processes of the software being measured do not need to take any account of the fact that interrupts may occur.
- Alternatively the interrupt may be passed to the executing functional process as an additional Entry to its normal triggering Entry. The action the functional process must take on receipt of the Interrupt Entry will depend on its FUR. The action may result in additional data movements to those needed for normal execution and should be measured according to the normal rules.

1.1.3 Functional size may vary with the functional users of real-time software

The most common purpose of measuring a functional size of real-time software is for project performance measurement and/or project effort estimation. For most measurement purposes and scopes, the functional users of real-time software will be identified as the 'things' that interact *directly* with the software being measured. Typically these may be:

- hardware input devices (e.g. sensors, measurement devices, a clock);
- hardware output devices (e.g. an actuator, a display, a communications line);
- other pieces of software or hardware that may send input and/or receive output.

However, the measurement purpose might be to size the functionality as seen by a human operator of a real-time system (e.g. the functionality provided at the operator workstation of a process control system or by the operator interface for a simple copier or mobile telephone).

Humans interact *indirectly* with software. They are only aware of the functionality that is made available to them via their input-output interface. This functionality may be much less than the total functionality that the software must provide. It is therefore very important when measuring the functional size of real-time software to define clearly the functional users for which the measurement is valid.

For more on types of functional users, see section 2.2.

1.1.4 Real-time application software: embedded, executing on an operating system or federated?

We use the term 'application software' for any software developed by order of a stakeholder to perform a particular set of tasks, and the term 'infrastructure software' for operating systems, software data handlers and device drivers that support applications.

Real-time application software may be embedded on a chip (System on Chip, SoC), such as a field-programmable gate array (FPGA) or a programmable logic controller (PLC), which may itself be specialized for the application, e.g. to survive rugged operating conditions. Very commonly, a simple embedded software application interacts directly with various input/output hardware devices, so does not need an operating system.

Alternatively, real-time application software may be installed on a general purpose or specialized processor and execute with the support of a real-time operating system (RTOS). The RTOS may handle all messages to and from the various hardware input/output devices; it will reside in a different (infrastructure or 'lower') layer of the software architecture from the application. If the purpose is to measure the application software, the presence of the RTOS and any other layers of infrastructure software must be ignored, since a principle of the COSMIC method is that the scope of a measurement must be confined to one layer. The same principle applies to a need to measure some

[1] The ISO/IEC/IEEE 24765:2010 Systems and software engineering—Vocabulary standard defines 'interrupt' as '(1) the suspension (or termination) of a process to handle an event external to the process (2) to cause the suspension (or termination) of a process (3) loosely, an interrupt request.' (The words in brackets 'or termination' have been added for this Guideline.

software in any of the infrastructure software layers; the scope of a measurement must always be confined to software within one layer.

Federated systems, or a 'system of systems' comprise multiple elements of the above types of processors that communicate over a common bus or over a network. Each element in the system should be measured separately.

1.2 Statements of requirements

1.2.1 The problem of allocation of requirements to hardware or software

A difficulty for measuring a functional size of real-time software is that often the requirements are stated at the 'system' level, i.e. before they are allocated to hardware or software (e.g. in the System on Chip concept), rather than explicitly at the software level. Obviously, until a decision has been made - and documented - on which requirements will be allocated to software, it is very difficult to agree on measurement results because various Measurers may make different assumptions on what the software will do.

In principle the COSMIC method can be applied to functional requirements for information processing before they are allocated to software or to hardware, regardless of the eventual allocation decision. For example, it is straightforward to size the functionality of a pocket calculator using COSMIC without any knowledge of what hardware or software (if any) is involved. However, the assertion that the COSMIC method can be used to size functional user requirements allocated to hardware needs more testing in practice before it can be considered as fully validated.

If system requirements must be measured for which their allocation between software and hardware is not clear and no expert advice is available to decide on the allocation, the Measurer should document any assumptions about the allocations and measure the software part. The lower and upper limits of the expected software functional size should also be indicated.

1.2.2 Requirements in the EARS syntax

It is a well-known fact that requirements in natural language are inherently imprecise. Therefore various methods have been developed that support the structuring of natural language writing to improve the clarity of requirements. An example is EARS (Easy Approach to Requirements Syntax [4]), of which its generic requirements syntax fits perfectly with the COSMIC model for identifying functional processes. Some of the EARS syntax is equivalent to COSMIC concepts:

- WHEN <trigger> is equivalent to 'triggering event'
- <system response> is equivalent to 'functional process'
- WHILE <in a specific state> may involve a state inspection requiring an 'Entry' data movement or 'Exit/Entry' data movements, or a 'Read' if the state has been stored persistently by an earlier occurrence of a functional process.
- IF <trigger> is equivalent to 'triggering event'
- 'WHERE <feature is included>' belongs to a complex requirement, in which one or more of the aforementioned clauses are combined.

1.2.3 Requirements in a finite state machine

Requirements of real-time software are sometimes documented with help of a model called a 'finite state machine' (or 'finite automaton'). When a finite state machine is used to represent software requirements, it shows the finite number of states that the software is expected to handle, and the triggering events or conditions which move the software from one state to another (possibly returning to the same state). A change from one state to another is called a 'transition'. Finite state machines are often visualized by a state transition diagram and/or by means of a table. Note that one triggering event may correspond to one or more state transitions depending on the state the machine is in when the event occurs. Chapter 4 gives a worked example of the measurement of requirements expressed in a finite state machine; figure 4.3.2 shows an example of a state transition diagram.

Regardless of the conventions used to document a finite state machine, the task of identifying separate functional processes depends only on identifying unique triggering events – 'something has changed or must now change in the external world (i.e. outside the software being measured), that may give rise to one or more state transitions'.

Two examples

EXAMPLE 1. The 'Next' button on a car radio/CD player is a functional user of the software that controls the device. Pressing the Next button has a different effect depending on whether the device is in the 'Radio state' or in the 'CD state'. When in the 'Radio state', the device must move to the next station; in the 'CD state', to the next CD track. The control software has one functional process that deals with both situations and is triggered externally by pressing the 'Next' button. If the functional process, having started, needs to determine the current state, this would require:

- *an Entry data movement if the software must obtain the state from a hardware functional user (assuming it does not need to be told what data to send),*
- *or a Read if the current state is available in persistent storage. (The state could have been written to persistent storage by a previously occurring functional process.)*

The functional process that deals with the response to the triggering event 'Next button pressed' is complete when it has done all that is needed to respond to the triggering event, i.e. including dealing with both the Radio and CD states.

EXAMPLE 2. When an elevator door is in the 'Open' state and the 'close door' button is pressed, this starts a motor to close the door. Whilst the motor is operating, the door is in a 'Closing' state. When the door is closed, it will be in a 'Closed' state.

Suppose that software is required to control this process. The 'close door' button, the door motor and the 'door closed' sensor are then functional users of the software. The whole door-closing sequence would require two functional processes. The first process would be triggered by the external event of the 'Close door' button being pressed and would start the door motor. The door state transitions from 'Open' to 'Closing'. A second functional process would be triggered by the 'door closed' sensor. This process stops the motor; the door state transitions from 'Closing' to 'Closed'. (Clearly, this example could in practice be much more complex, with the complete required response depending on the interaction with safety devices, etc.)

1.2.4 Requirements for a programmable logic controller.

The IEC 61131-3:2013 standard [5] specifies the syntax and semantics of a unified suite of programming languages for programmable logic controllers (PLCs). This suite consists of two textual languages, Instruction List (IL) and Structured Text (ST), and two graphical languages, Ladder Diagram (LD) and Function Block Diagram (FBD). Execution of actions depends on conditions, the combinations of which are described in the textual languages with Boolean algebra.

Besides these, PC functionality is sometimes also documented with the help of other models, such as the finite state machine (FSM) model and/or decision tables.

The three graphical languages have been defined to improve the clarity of requirements, each with an associated set of diagram conventions. Although equivalences can be recognized between the conventions of these languages and some COSMIC concepts (see table 1.1 below for examples), the Measurer must be very careful in converting graphical language concepts into COSMIC concepts:

- The languages allow a wide variety of levels of granularity. For instance, an element in the one diagram may represent a functional process whereas in another it may represent some data manipulation (COSMIC regards data manipulation as accounted for by a data movement (sub-process) of a functional process; data manipulation is not measured separately).
- The languages may be combined, such that for instance conditions in one language may be expressed using conventions of another language.
- Determining functional processes is difficult because it isn't always obvious whether a transition is caused by a functional user or by the software itself.
- Sometimes functional users and conditions aren't distinguished in text.

Function block diagram	Ladder diagram	Sequential function chart	COSMIC concept to be expected
Text left of input variables, text right of output variables (arrows)	Logical checkers (contacts), actuators (coils)	Text right of links between steps	Functional user or data movement from/to functional user
Text left of input variables, text right of output variables (arrows)	-	Transition	Triggering event or data manipulation
Block	-	-	Functional user or piece of software
Function (number of blocks)	Whole ladder, number of adjacent rungs, rung	Step	Functional process or piece of software
Text above input and output variables (arrows)	Rung	Action	Data movement to functional user or data manipulation

Table 1.1 - Correspondence of graphical language concepts and COSMIC concepts

1.2.5 Requirements in specialized tools

Many software tools exist for supporting the management of requirements, i.e. for capturing, analyzing and modeling of requirements and for evolving these into designs. Capturing and structuring requirements in such tools ought in principle to make it easier to measure a functional size. But typically, the requirements are captured in a 'top-down' manner at successively lower levels of granularity. If a size measurement is needed early in the life of the project, when the requirements are known only at a higher level of granularity than is needed for a precise COSMIC size measurement, then an approximation variant of the COSMIC method will have to be used (see [6] for examples).

Whenever requirements are captured in a tool, the diagramming conventions of the tool will have to be mapped to the concepts of the COSMIC method, so that functional users, triggering events, functional processes and types of data movements can be identified.

A recent survey of requirements engineering tools [7] indicates that most tool suppliers have not yet developed their products to deliver a software size measure according to a standard method such as the COSMIC method. However, users of such tools are starting to develop their own processes for automating COSMIC functional size measurement directly from the contents of a requirements engineering tool, especially in the real-time domain. See [8], [9], [10], [22], [23] for examples from the automotive industry for software embedded in electronic control units. See also section 4.5 and [11], [12] and [21].

1.2.6 Requirements in UML

Several studies have shown the close correspondence between concepts of the Unified Modeling Language (UML) and those of the COSMIC method. Lavazza et al. [13] give an example of a case study fully documented in UML and measured with COSMIC.

Mapping from UML to COSMIC needs to be done carefully as a Use Case diagram may be drawn at any level of granularity, i.e. at any level of detail. A Use Case diagram may correspond to a few, or one, or a part of a functional process. Once the correct level of granularity has been determined, measurement of corresponding Message Sequence Diagrams is very straightforward.

In [8] and [9] a UML profile based on components is defined capturing all information needed for measuring COSMIC functional sizes of software and the use of these sizes to estimate the software code size. Also it is shown how the mapping between Component Diagrams (an alternative to Use Case diagrams) and COSMIC was implemented in a tool, and how the information modeled in the UML profile is imported into the tool. The tool automates the choice of an estimation model for code size and of code size estimation itself, given a number of measurements.

Another approach to automated COSMIC size measurement of requirements for new software and for enhancements held in UML models is described in [19].

1.2.7 Non-functional requirements

A standard view of non-functional requirements [1] is that they 'include but are not limited to:

- quality constraints (for example usability, reliability, efficiency and portability);

- organizational constraints (for example locations for operation, target hardware and compliance to standards);
- environmental constraints (for example interoperability, security, privacy and safety);
- implementation constraints (for example development language, delivery schedule)'.

Note that some non-functional requirements apply to the hardware/software system, e.g. response time, whereas others almost always apply wholly to the software, e.g. 'portability'.

'Non-functional' is a misleading term. Al Sarayreh and Abran have examined the non-functional requirements standards of the European Cooperation on Space Systems and of the IEEE for space systems software. These address topics such as maintainability, operability, usability, portability, etc. They have shown that in all cases much of the requirements evolve into functional requirements that are allocated to software and can be sized with the COSMIC method, e.g. [14]. Symons, in a survey of non-functional requirements [15], concludes that a very high proportion of requirements that are conventionally thought of as non-functional are best described as 'quasi' non-functional. This is because most of them evolve wholly or partly, as a project progresses, into software functional requirements that can be measured. (The parts of a non-functional requirement that do not evolve into functional requirements remain as specific quantifiable or named constraints, e.g. for response time, availability, hardware platform, named interfaces, etc.).

Butcher [16], when discussing mission-critical systems, e.g. for air traffic control or for real-time financial trading systems, stated that up to half of the documented requirements initially concern non-functional requirements. However, most of these evolve into functional requirements as a project progresses. He preferred to distinguish 'direct' and 'indirect' functional requirements, rather than refer to them as functional versus non-functional respectively.

If a measurement is required early in the life of a project, the Measurer should seek to establish which non-functional requirements and constraints may exist that could lead to functionality allocated to software (in addition to the explicit functional requirements), and attempt to allow for these non-functional requirements in the measurement of total software functional size. All assumptions made should be documented.

For more on Non-functional Requirements, please see the Measurement Manual [1] and the 'Guideline on how to account for Non-Functional Requirements in software project performance measurement, benchmarking and estimating' [24].

2

THE MEASUREMENT STRATEGY PHASE

Determining the 'strategy' for a COSMIC functional size measurement requires that various parameters be considered before starting an actual measurement. The parameters, to be discussed in this chapter, are:

- the purpose and scope(s) of the measurement;
- the functional users of the software to be measured (the 'things' that are the sources and intended recipients of the data to/from the software to be measured);
- the level of granularity of the functional requirements of the software to be measured and the level of decomposition of the software itself.

The effort needed to determine the strategy is usually trivial but recording these parameters helps ensure that the resulting size measurement can always be interpreted reliably, i.e. future users of the measurement can always be sure they are comparing 'apples with apples'.

As an aid to determining a measurement strategy, the Guideline for 'Measurement Strategy Patterns' [20] describes, for each of several different types of software, a standard set of parameters for measuring software sizes, called a 'measurement strategy pattern'.

2.1 The purpose and scope of the measurement

2.1.1 The measurement purpose

The purpose of a measurement defines why a measurement is required, and what the result will be used for.

2.1.2 The measurement scope

The purpose of the measurement determines the scope of the measurement (which defines the extent of the functionality to be measured). A particular purpose may require more than one piece of software to be measured separately, i.e. there would be more than one measurement scope.

The scope of a piece of software to be measured must be confined to a single software layer. For more on distinguishing layers, see the Measurement Manual [1].

2.2 Identifying the functional users

When measuring real-time software, the functional users ('the senders and/or intended recipients of data') that interact with the software being measured will be typically any of the following:

- a clock (See the Glossary for use of this term in this Guideline);
- sensors (e.g. of temperature, pressure, voltage) that provide input, either when polled, or via interrupts, or by sending their data and/or status at intervals;
- hardware devices that receive output (e.g. a valve or motor actuator, switch, lamp, heater);
- hardware chips, having the ability to trigger functional processes (e.g. watchdog chips);
- 'dumb' hardware memory such as a ROM which can only respond to a request for data;
- communications devices (e.g. telephone lines, computer ports, aerials, loudspeakers, microphones);
- hardware devices with which humans interact (e.g. push buttons, keyboards or displays);
- other pieces of software that supply data to or require data from the software being measured.

A ‘context diagram' shows the interaction of the software being measured with its functional users. When drawing a context diagram, it may be helpful to distinguish functional users that are:

- sources of triggering events (and therefore of data groups that are moved by triggering Entries)
- sources of other input data (e.g. that may be polled to provide data groups for non-triggering Entries)
- intended recipients, or destinations of data (to which Exits are sent)

Some functional users may fulfill more than one of these roles, e.g. other pieces of software that interact with the software being measured, intelligent hardware devices or communication lines.

All the above types of functional users may interact with the software being measured either directly, or indirectly, e.g. via an operating system or simple device driver software. However, the functionality of this ‘enabling’ software should be ignored (unless, of course, it is the subject of the measurement).

Real-time software may also be measured from the viewpoint of humans as functional users (‘the senders and/or intended recipients of data’) that interact indirectly with the software being measured e.g. operators that start and stop the system, set parameters, monitor displays of operational performance, need to be notified of emergency conditions, etc.

Consider the case where a button is pushed by a human operator. Is the functional user taken to be the button that interacts directly with the software to be measured, or the human that presses the button that interacts indirectly with the software? (They ’see’ different events. For the button, the event is ‘I have been pressed’. For the human operator, the event is perhaps an emergency condition that means he/she must raise an alarm). The choice depends on the purpose of the measurement. This choice of functional users must be one or the other; it makes no sense to measure a size, or to sum two sizes, as seen by a mix of functional user types (humans, hardware devices or other pieces of software). The following Examples illustrate where there is a choice of functional user.

EXAMPLE 1. Section 4.1.1 describes an industry process which is controlled by a programmable logic controller (PLC). The purpose is to measure the size of all the embedded software functionality needed to make the system work, not just the limited view of the functionality as seen by a human operator. The process is started by a human operator pushing a start button. But in this example, given the measurement purpose, the start button is considered to be a functional user, not the operator who pushes it to start the process.

EXAMPLE 2. The embedded software of a mobile phone (‘cellphone’) has to interact with several types of buttons, a screen (which may serve as an input device as well as output display), its battery, loudspeaker, aerial, etc. A human user of such a phone sees only a small part of the functionality that the software needs to provide its services. So it is possible to measure two functional sizes, depending on the choice of functional users.

Toivonen [17] measured the functionality as seen by human users of two mobile phones in order to compare their ‘packing density’ (functional size / memory size). This was an important economic measure for the phone manufacturer. But the software sizes measured by Toivonen were much smaller than the sizes that the software engineers would have to develop to provide all of the functionality necessary for the phone to meet all its requirements of all its hardware/software functional users.

Determining the functional users depends on the requirements, as the following example shows.

EXAMPLE 3. One or more buttons (-types)?

Consider a factory that has a moving production line that can be stopped by pushing a button; there are buttons at several different locations along the line. Should the Measurer identify one or several functional users (types)? The answer depends on the functional ‘user’ requirements that must be measured. The issue from this example is whether pressing the buttons leads to different triggering events and separate functional processes, e.g.

a) Requirements: Any operator may press a button to stop the line in an emergency. When a button is pressed, the system logs the time at which the line was stopped and the button that was pressed. There are many identical buttons along the line that all have the same effect. Identify only one functional user type and one functional process type;

b) Requirements as case a) but there is also a requirement for a button in a supervisor's office which is used by the supervisor to stop the line at the end of the work-day. If it has only the same effect as a), then still identify only one functional user type and one functional process type.

c) Requirements as case b) but in addition to its use for stopping the line at any time, there is a requirement that when the button in the supervisor's office is pressed AND held down for three seconds, the system stops the line and then produces a log of the day's stop/start events. (The timing of the three seconds is controlled by the button itself.) We now have two functional users (any stop button on the line, and the supervisor's stop button) and two functional processes. The two functional processes share some functionality (stopping the line) but are invoked by different triggering events (emergency stop, and end-of day stop) and have different effects.

d) Requirements as case c) but there is an additional requirement that the supervisor has a second button that when pressed will start or re-start the line after it had been stopped and log the start time. Now we have three functional user types (any stop button on the line, and the supervisor's stop and start buttons) and 3 functional processes (stop the line, stop the line and produce a report from the supervisor's first button, and start or re-start the line from the supervisor's second button).

See also section 2.3 and real-time example 1 in section 3.5.9, both of the Measurement Manual [1]. See also the Tire Pressure Monitoring System case in section 4.5 of this Guideline.

2.3 Identifying the level of decomposition and the level of granularity

Before starting a measurement of requirements, two aspects of the requirement artifacts must be considered to ensure that the measurement will satisfy its purpose, be as accurate as needed, and be interpreted with certainty by future users:

- The 'level of decomposition' of the software. This refers to the sub-division of the software into separate components that may exchange or share data. The process of sub-division may start during the requirements definition stage when an initial requirement is seen to be too large to be handled by one team and so it is decided to sub-divide the system into different sub-systems, sub-sub-systems, etc., which may be implemented at different times.
- The 'level of granularity' of the requirements for the software and/or its components, which concerns their level of detail. Often, in the life of a project, requirements are produced in a 'top-down' way, i.e. first in outline form, then as the project progresses being worked out in more and more detail (in other words at lower and lower levels of granularity).

  A precise COSMIC size measurement is possible only when the detail is sufficient to identify all the individual events that the software must respond to and hence the functional processes and their data movements. If a size measurement is required before these details are available, then approximation of the COSMIC method are used. These involve scaling sizes measured at the level of granularity of the requirements to the level of the functional processes. Care must be taken with these methods since, at a given point in time, different parts of the requirements may have been worked out at different levels of detail.

Note that any decomposition of the software should be determined *before* the level of granularity of the requirements, as the latter might vary from one component to another. As requirements and the software design evolve, both parameters should be monitored for their effect on the measurement approach.

There is nothing specific to real-time software in considering these two factors. More detail on these two factors is given in the Measurement Manual [1]. The 'Guideline for early or rapid COSMIC sizing of functional requirements' [6] discusses several approaches to approximate sizing. It includes a worked example of sizing the requirements of a telecoms software system at successively lower levels of granularity

3

THE MAPPING AND MEASUREMENT PHASES

3.1 Identifying the triggering events and functional processes

As described in section 1.1.1, software is triggered to do something by 'events' in the world of its functional users. Identifying the triggering events is therefore of critical importance because it enables the Measurer to identify the 'something', namely the functional processes. The steps for identifying functional processes in the functional user requirements (FUR) of a piece of software are as follows:

1. Identify the separate events in the world of the functional users that the software being measured must respond to – the 'triggering events' (triggering events can be identified in state diagrams and in entity life-cycle diagrams, since some state transitions and some entity life-cycle transitions that the software must react to correspond to triggering events);
2. Identify which functional user(s) of the software may respond to each triggering event;
3. Identify the triggering Entry that each functional user may initiate in response to the event (there may be other Entries as well);
4. Identify the functional process started by each triggering Entry.

EXAMPLE: The speedometer software of a car is connected to a rotation measurement sensor located on the drive shaft that measures its revolutions per minute (rpm), and to a key-in sensor, a clock, and a display unit for the driver. The software's persistent storage contains the parameters needed to send messages to a pre-defined variety of display units. The speedometer software is required to capture at key-in time the display parameters and initialize the installed display unit. A clock at 5 millisecond intervals triggers the software to capture rpm information from the drive shaft, calculate the speed, and send the speed to update the display unit using parameters appropriate for this display unit.

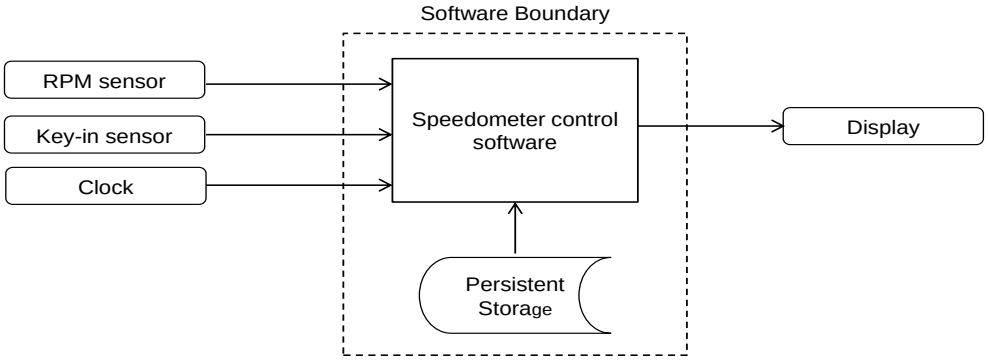

Figure 3.1 – Context diagram for the speedometer software

The context diagram shows the four functional users of the speedometer software, namely three input devices (the rpm sensor, the key-in sensor and the clock) and the one output device (the driver display).

There are two events that need to be responded to by the speedometer software (i.e. are triggering events), the key-in event and the 5 millisecond clock tick. Hence the speedometer control software has two functional processes, FP1 and FP2.

- *FP1 initializes the speedometer control software on the event of 'key-in' detected by the key-in sensor, which includes reading the parameter data for the display;*
- *FP2 measures the speed on the event of the tick generated by the clock every 5 ms and sends the speed to the display.*

3.2 Identifying objects of interest, data groups and data movements

3.2.1 Objects of interest and data groups

Any functional process comprises sub-processes, called 'data movements' that both move data and are considered to account for related data manipulation. A data movement moves a data group whose attributes describe a single 'object of interest'.

In real-time software a data group often comprises one or a few data attributes, sent from an input device, e.g. a sensor to software, or of a signal sent from software to an output device, e.g. an actuator. The object of interest described by the data group can be determined from the devices involved. For instance, the speedometer control software in the example of section 3.1 has the 'RPM sensor' as a functional user. This sensor sends a data group to the software, which has one attribute 'current rpm'. The object of interest of this data group could be considered as the drive shaft or the rpm sensor. It is often the case in real-time software that a functional user (the RPM sensor in the example above) is also the object of interest of a data group that it sends (i.e. it is sending data about itself). For more on 'The functional user as object of interest' see section 3.3.5 of the Measurement Manual [1].

3.2.2 Data movements

There are four types of data movements: Entry, Exit, Read, and Write.

Entry and Exit data movements move a data group across a software boundary, from or to a functional user respectively. Read and Write data movements move a data group from or to persistent storage respectively.

For real-time software, the rules of section 3.5.9 of the Measurement Manual [1] 'When a functional process requires data from a functional user' are particularly important. Figure 3.2 shows the various ways in which real-time software can receive or 'get' data from its functional users (which varies with their capabilities) and from persistent storage.

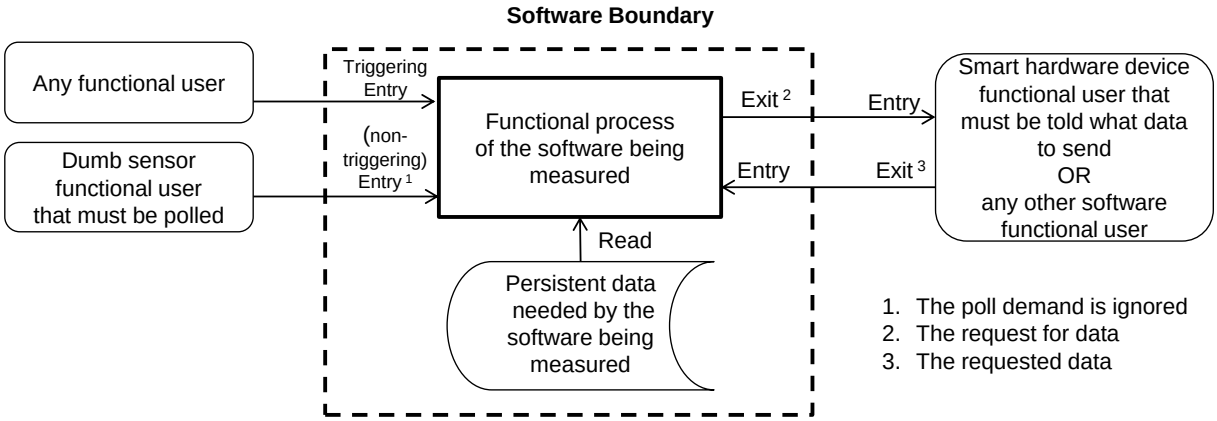

Figure 3.2 - The various ways in which a functional process can receive or get data

Data may be present in persistent storage and be available to be read either:

- by the data having been made persistent (by a Write data movement) of another functional process, or by an earlier occurrence of the functional process that will read the data, or
- by being stored in physical read-only memory during the manufacture of a chip in which software will be embedded. Parameter data needed by the embedded software may be stored this way.

It may be queried why a whole Entry data movement (worth one COSMIC Function Point, or CFP) is measured when the signal may be a single bit, as in the case of a clock tick. But remembering that each data movement is assumed to account for the associated data manipulation, a triggering Entry must account for initializing sub-processes, not just the movement of one bit.

As well as receiving or getting data, a functional process can, of course, also:

- send out data to a functional user via an Exit data movement where no response is expected,
- and can 'put' data to persistent storage via a Write data movement.

3.2.3 Data manipulation

The COSMIC method was not designed to account explicitly for data manipulation. As noted above, the method assumes that data manipulation is accounted for by each data movement.

However, the method may be reasonably used to measure certain types of software that require extensive data manipulation. See section 4.5.2 of the Measurement Manual [1]. This is true, for example, where the software must handle high volumes of data, leading to very large numbers of data movement types. The latter may effectively account for any mathematically-complex data manipulation that may also be present. By 'reasonably used', we mean that the method has produced meaningful and useful sizes in relation to the purpose of the measurement, e.g. project performance measurement, estimating, benchmarking and such-like. Examples include the sizing of expert systems, software to digitally process continuous variables, software that collects and analyzes data from scientific experiments or from engineering measurements, etc. See section 4.7 of this Guideline for two examples of the measurement of software that include significant data manipulation.

If the software that must be measured is mostly 'movement–rich' but includes some significant, localized mathematical algorithms, the COSMIC method allows for a 'local extension' whereby an organization can define its own local scale for sizing algorithms alongside the CFP scale for sizing software functionality. Alternatively, if the purpose is estimating project effort, the measurement scope can be defined to exclude the algorithms. The sizing and estimating process can then be applied only to the 'movement-rich' functionality whilst estimating for developing the algorithms can be dealt with by another appropriate process. For more on this subject see section 4.5.5 of the Measurement Manual [1], entitled 'Local extension with complex algorithms'.

3.2.4 Error or fault messages in real-time software

Messages containing error or fault indications in real-time software must be analyzed in the same way as any other data movement.

> *EXAMPLE 1: If a message issued by the software being measured may include an indication of a fault or of an error in the data group describing a particular object of interest, in addition to or in place of the expected data, this fault/error indication describes the same object of interest as the expected data and hence there is only one Exit, i.e. the fault/error indication is not identified as a separate Exit.*
>
> *EXAMPLE 2: If the reason for a fault or error condition is issed by the software being measured as a separate message, e.g. 'Error_source = Sensor_failure, Internal_error, etc, then a separate Exit should be identified.*

For definitions and other examples of error messages in real-time software, see the Measurement Manual [1], section 3.5.11, Real-time examples 1 and 2.

See also the cases in sections 4.1, 4.2, 4.3 and 4.5 of this Guideline, which all include messages indicating error or fault conditions that are measured as normal Exits.

3.3 Measurement and measurement reporting

See the Measurement Manual [1] for all principles and rules for

- aggregating measurement results
- measuring the size of changes to software
- measurement reporting

All of these topics are domain-independent, i.e. there is nothing specific to real-time software.

4

CASES

4.1 Industry automation and the PLC

4.1.1 The programmable logic controller (PLC)

Programmable Logic Controllers (PLC's) are computers with extensive input and output facilities that can be connected to sensors, actuators and such-like. Many industrial processes are controlled by PLC's including conveyor belts and associated machinery, flat-product fabrication, assembly-line manufacturing and chemical processes.

4.1.2 Measurement of the PLC software for controlling a process in a chemical factory

Requirements

A process in a chemical factory is controlled by a PLC. The process consists of filling a tank with a liquid, heating the liquid and then emptying the tank when a temperature is reached that is pre-set in the temperature sensor device.

In the following description of the requirements of the process (system) control we assume that all mentioned functionalities are allocated to the PLC software, unless stated otherwise.

- The process is started by a human operator pressing the start button of the PLC which controls all the subsequent steps.
- The software opens the inlet valve of the tank and the tank fills with liquid under gravity.
- When the tank is full ('high level reached' is detected by the high level sensor) the software closes the inlet valve and starts the heater to heat the liquid.
- When the software is informed that the pre-set temperature is reached, it stops the heater, opens the outlet valve and starts the pump to empty the tank.
- The pump continues emptying until 'low level reached' is detected by the low level sensor, when the software stops the pump.
- During the entire the process, the process status ('Filling', 'Heating', 'Pumping') is shown on an operator display. When the process is finished, an audible alarm is produced and the message 'Process finished' is shown on the display.
- When the process is started and whilst the process is running, the PLC software polls the valves, the heater and the pump asking for their status at regular intervals to detect any fault conditions.
- If the PLC software is informed that an error is detected, it starts the audible alarm and displays a message to the operator showing the device(s) concerned. If an operator receives an error message, the operator deals with it manually, outside the software system.
- The polling frequency is determined by signals from a clock.

Context diagram

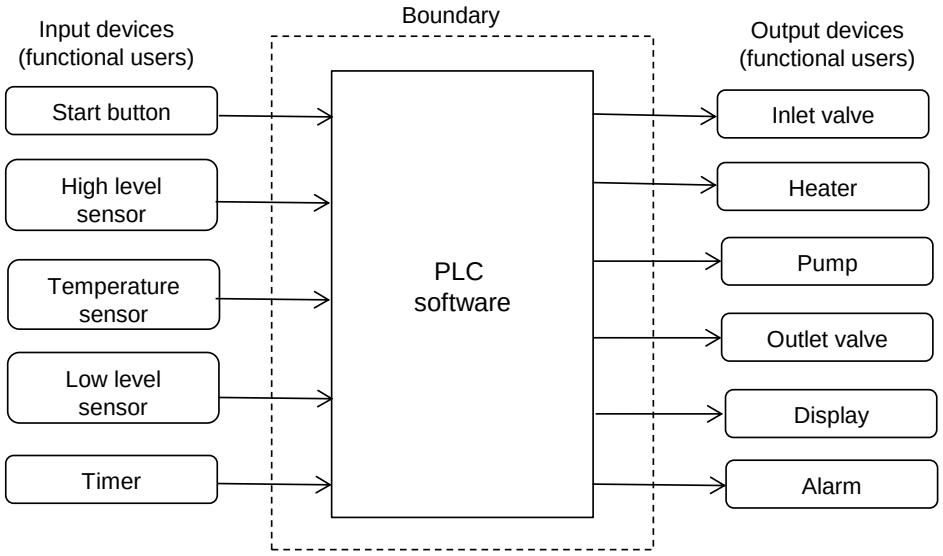

Figure 4.1 - Chemical factory process, PLC software and devices

Analysis

The measurement assumes all the hardware devices that interact directly with the software are the functional users, as shown in the context diagram. The PLC does not have an operating system.

In the requirements as described above, the software is not decomposed in any way and is not a component of another piece of software. The level of granularity of the requirements is at the 'functional process level of granularity', i.e. the level of individual functional users and events (rather than groups of these).

There are several triggering events (events the software is required to respond to) and corresponding functional processes (that respond to the triggering events):

Triggering event	Functional user that initiates the functional process	Corresponding Functional process
Start button pushed	Start button	Start process/Fill tank
High level reached	High level sensor	Heat liquid
Pre-set temperature reached	Temperature sensor	Stop heating/Empty tank
Low level reached	Low level sensor	Finish process
Clock tick (= time to poll)	Clock	Fault check

Table 4.1 - Chemical factory, triggering events and functional processes

Each data group moved must describe an aspect of a single object of interest. The data groups consist of the signals from the start button, the sensors and the clock to the software and the signals from the software to the actuators, valves and the devices for the operator.

As noted in section 3.2.1, in this case the object of interest of each data group entering the software is the functional user that sent the group (i.e. the functional user is sending data about itself). Similarly the object of interest of each data group that leaves the software is the functional user that receives the group (i.e. the functional user is being sent data about itself). For instance, the data movements starting or stopping the pump move data groups that specify the (desired) states of the pump. The pump is therefore the object of interest of these data groups.

The software determines the process status to be displayed from the triggering Entry for each functional process (except the Fault Check process). For instance, from the start button signal the software determines that the current status is 'Filling' and displays this status.

The functional processes of the PLC software are as follows. The data movements, the data groups moved and an explanation are shown for each functional process. We assume that the devices that the software polls to determine their status are ‘dumb’, i.e. the software inspects the state of these devices, which requires only one Entry per device type for the poll (see the Measurement Manual [1], section 3.5.9). In the following functional processes, 'DM' means 'data movement'.

Functional process: Start process/Fill tank

DM	Data group	Remark
E	Start signal	Triggering Entry, from start button
X	Open inlet valve	Signal to inlet valve, start entering liquid
X	Signal to clock	Activate clock for fault detection at regular intervals
X	Process status to display	Display ‘Filling’

The size of this functional process is 4 CFP.

Functional process: Heat liquid

DM	Data group	Remark
E	High level reached	Triggering Entry, signal from high level sensor, tank full
X	Close inlet valve	Signal to inlet valve, stop liquid entering
X	Heater on	Signal to heater, start heating
X	Process status to display	Display ‘Heating’

The size of this functional process is 4 CFP.

Functional process: Stop heating/Empty tank

DM	Data group	Remark
E	Pre-set temperature reached	Triggering Entry, signal from temperature sensor, liquid heated
X	Heater off	Signal to heater, stop heating
X	Open outlet valve	Signal to open the outlet valve
X	Pump on	Signal to pump, to start emptying the tank
X	Process status to display	Display ‘Pumping’

The size of this functional process is 5 CFP.

Functional process: Finish process

DM	Data group	Remark
E	Low level reached	Triggering Entry, signal from the low level sensor, tank empty
X	Stop pump	Signal to pump to stop
X	Close outlet valve	Signal to close outlet valve
X	Process status to display	Display ‘Finished’
X	Audible alarm	Signal to operator, process finished
X	Signal to clock	Stop clock

The size of this functional process is 6 CFP.

(For polling the devices, 'prompt message Entries' are assumed (see Measurement Manual section 3.5.9., Rule a))

Functional process: Fault check

DM	Data group	Remark
E	Start signal	Triggering Entry, signal from clock to start fault check
E	Inlet valve status	Signal resulting from inlet valve status polling
E	Outlet valve status	Signal resulting from outlet valve status polling
E	Heater status	Signal resulting from heater status polling
E	Pump status	Signal resulting from pump status polling
X	Audible alarm	Signal to operator, if device fault(s) detected
X	Fault info to display	Display device(s) if there is an fault[2]

The size of this functional process is 7 CFP.

The total software functional size of the PLC software is 4 + 4 + 5 + 6 + 7 = 26 CFP.

4.1.3 Measurement of a change to the PLC software

Requirements

It has been decided to remove the audible alarm device and to adapt the software accordingly.

Analysis

The signals from the software to the alarm device can be removed, i.e. the Exit data movements of the audible alarm data group in the last two functional processes must be removed. The functional size of the change is 2 CFP. The resulting software functional size will be 24 CFP once the change is made.

4.2 Timing functionality

(See the Glossary for the definitions of 'clock' and 'timer' as used in this Guideline.)

Measuring timing functionality requires clear specifications on what functions are allocated to the hardware part of the functionality, and what are specifically allocated to the software part.

EXAMPLE 1. Timing functionality needed, for example to control a pre-set time interval can be implemented in several ways, with different divisions between the hardware and software:

- *A hardware clock generates pulses ('clock ticks') at regular defined intervals each of which triggers a functional process of the software. The software keeps track of the pulses, may convert them to seconds or minutes if needed, and increments the elapsed time until the pre-set time is reached.*
- *A hardware timer both generates and keeps track of the pulses and transforms them into seconds, minutes etc., in an internal register if needed. The software may start the timer which informs the software when the desired time is reached. This mechanism is used in the next Example 2.*

EXAMPLE 2. A web-server must access a customer information system to retrieve some customer data. In addition to handling this request, the server starts a monitoring process to check that the request for customer information is handled within a set time. The aim is to ensure that the human user who seeks the customer information is not left hanging indefinitely if the customer information system fails to respond. Figure 4.2 shows a message sequence diagram for a simple example (no re-tries) of how this might be done via the interactions of the functional processes of the four participants, which are functional users of each other:

[2] In this case, the PLC does not have an operating system and the display is stated to be a functional user (not the human that reads the display). This Exit is therefore NOT an 'error/confirmation message', as defined in the Measurement Manual,

- *Web-server (functional user 1)*
- *Customer information system (functional user 2)*
- *Monitor (functional user 3)*
- *Real-time Timer (functional user 4)*

The web-server, after issuing the request to the customer information system, issues another message to the monitor, requesting it to respond if the given time-out period is exceeded. If the web-server receives the data from the customer information system within the time-out period, it tells the monitor to stop monitoring. Otherwise, if the web-server first receives a reply from the monitor that the time-out period has passed, the web-server issues a time-out message to the functional user that requested the customer data.

The monitor logs the request from the web-server and issues a request to the real-time timer, asking for a response within the given time-out period. (The timer may be implemented in hardware and/or software of the RTOS; it does not matter.) The monitor next receives either a message from the web-server to stop monitoring, or a message from the timer that the time-out period is complete. If the latter, the monitor sends a time-out message to the web-server. On completion, the monitor cancels the request from its log.

In Figure 4.2, the data movements of the timing functionality are shown as dashed lines. The functionality requires 4 CFP for the web-server to request the monitoring function, in addition to the 2 CFP to obtain the customer data. The monitor requires 8 CFP to fulfill its requirement. (The 'delete request' Write is counted only once, although it may be issued at two alternative times, depending on whether the customer data are returned within the given time-out period or not.)

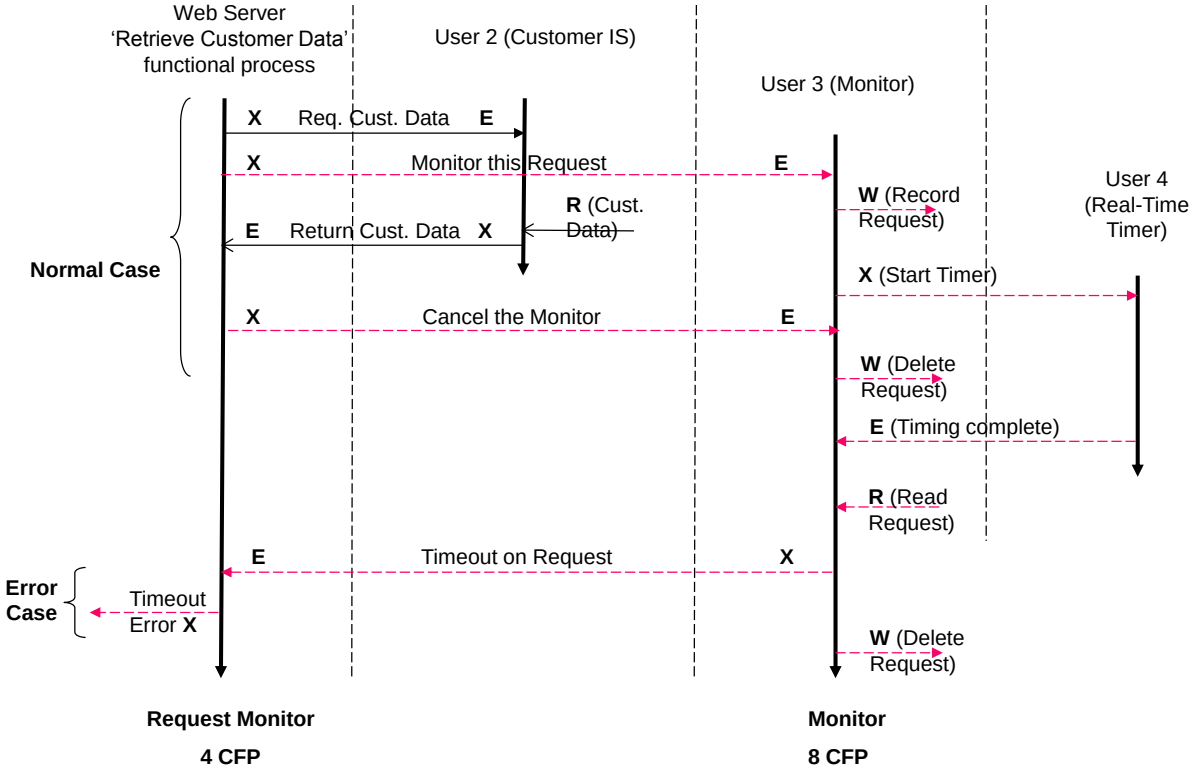

Figure - 4.2 The functionality for a web-server to monitor 'time-out'

4.3 Intruder alarm system

Figure 4.3 shows the context diagram for the embedded software of a basic domestic intruder alarm system and its functional users.

(Assumed) Requirements:

(These have been deduced from knowledge of how to use the system and by examining it physically. We are only interested in the functionality available to the normal house occupants, not to the alarm maintenance engineer).

- The software is connected to a keypad and red/green LED's for the human interface.
- The software is also connected to a device that senses whether the main front door of the house is open or not, several internal movement detectors, and an internal and an external alarm.
- There is a battery to take over if the main power supply fails, so there must be a power voltage detector.
- Finally, the PIN (Personal Identification Number) code is stored by the software and can be changed, so there must be some persistent storage.
- There must be a timing mechanism, since certain functions must be completed within pre-set times. For example, having set the 'exit code' before leaving the house, the front door must be closed within a given time, or alarms will sound. The timer is started in the software each time it is needed. We do not know how the timer is implemented but a hardware timer seems unlikely. We have assumed a software implementation of the timer, so this functionality to keep track of elapsed times is a form of data manipulation, which we can ignore.
- There is a legal requirement that the external alarm must stop after 20 minutes.

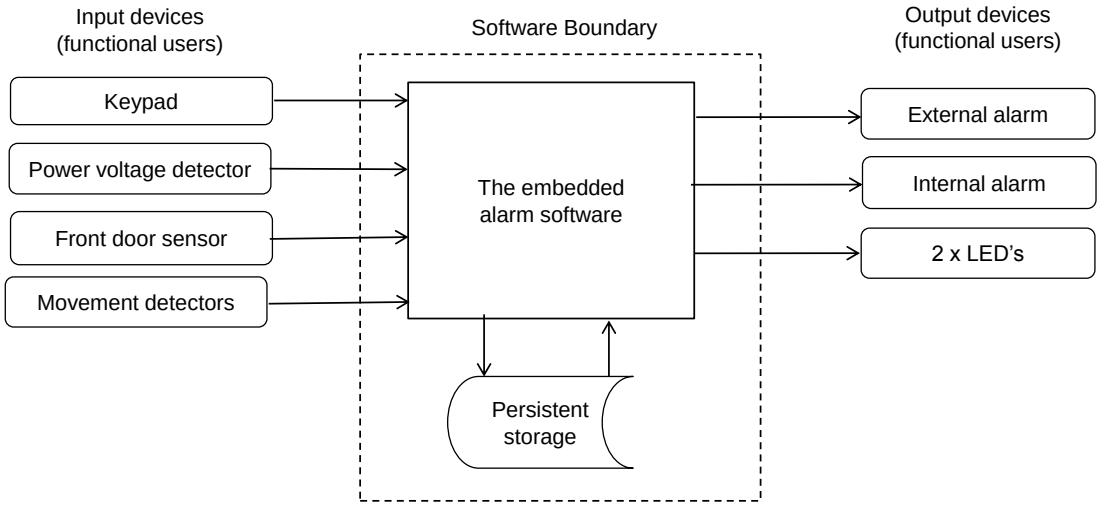

Figure 4.3 - The intruder alarm system, context diagram

Analysis

Although input originates from a human operator and output is displayed for the operator, this is again a form of PLC software that interacts directly with its various input and output devices. We do not know whether the embedded alarm software includes an operating system, but its presence or absence would make no difference to the measurement of the functional size of the alarm application software.

The following are the unique types of triggering events that the software in the alarm can respond to, each of which triggers a unique functional process. There are probably more event types, known only to the maintenance engineer, but these are not covered by the measurement scope.

- A new, or changed, PIN code is entered via the keypad (2 events).
- The ‘exit code’ is entered via the keypad thereby activating the system (before leaving the house and closing the front door within a pre-set time).
- The front door sensor detects that the door has been opened whilst the alarm system is activated.
- The system is activated, or de-activated, by the occupants whilst they are in the house (2 events), e.g. when retiring at night, out of range of the movement detectors.
- A movement detector signals a movement whilst the alarm system is activated.
- The power voltage detector signals electrical failure, or its restoration (2 events)

As an example, we show the analysis of the situation where the alarm is active and the front door is opened from outside. When someone opens the front door, the internal alarm starts beeping, and the PIN code must be entered within a pre-set time to de-activate the system and to stop the internal alarm beeping. If the PIN code isn’t entered before the pre-set time, or the wrong code is entered more than three times, the external alarm starts wailing.

There is one triggering event involved, 'Opening the front door from outside'. The functional user, the front door sensor, detects the event and informs the software. As noted in section 3.2.1, in this case the object of interest of each data group entering the software is also the functional user that sent the group (i.e. the functional user is sending data about itself) and similarly the object of interest of each data group that leaves the software is also the functional user that receives the group (i.e. the functional user is being sent data about itself).

The functional process is assumed to work as follows.

Functional process: Response to opening the front door from outside

DM	Data group	Remark
E	Signal ‘front door open’	Triggering event, sensor detects front door open
R	Read PIN from persistent storage	Current PIN code is stored in a persistent data group
-	Start internal timer	We assume this is data manipulation. Ignore
X	Signal to internal alarm	Start internal beep
E	Enter PIN code	Code validation is data manipulation, associated with the Entry
X	Software switches red LED from ‘on’ to ‘off’	PIN code entered in time and correct. We suppose 2 Exits are required. This Exit switches the red LED ‘off’.
X	Software switches green LED from ‘off’ to ‘on’	PIN code entered in time and correct.. This Exit switches the green LED ‘on’
X	Signal to internal alarm to stop beeping	PIN code entered in time and correct
-	Enter PIN code again, previous PIN code wrong	This is a repeat occurrence of the previous Entry. It is recorded to show consistency with the specification but the repeated occurrences of the data movement are ignored
X	Signal to external alarm	Wrong PIN-code entered more than three times and/or not entered in time, start external wailer alarm
X	Stop external alarm	Legal requirement, after 20 minutes

The size of this functional process is 9 CFP.

Discussion

- Could the Entry 'Enter PIN code' be considered as indicating another triggering event? In other words, is this really one functional process or should it be split up into two functional processes? The answer is ‘no’. 'Enter PIN code' is not another triggering event for two reasons. First this functional process was triggered by the ‘front door open’ message. According to the definition of a functional process, it is not complete until it has met its FUR for all possible responses to that triggering event. Second, this functional process invites (or enables) the keyboard functional user to enter the PIN code. (The keyboard functional user does not interrupt the functional process that has started, asking to allow a PIN code to be entered.)

- The entry of the PIN could be considered as four repeated Entries of a single digit, or as one Entry of a four-digit number. Either way, there is only one Entry. (We have assumed that the check that the four digits are entered within a given time-limit is carried out by hardware.)
- Note that in this example of measuring software from observing how it works, it isn't necessary to understand the detailed logic or the precise sequence of steps of the process. The size measured accounts for all the data movements arising from all possible paths through the process. The example illustrates that any software engineer with a clear understanding of the COSMIC concepts should be able to apply the measurement process to the functionality of an existing software system with which they are familiar.

4.4 Cooker software defined as a finite state machine

Requirements

- The cooker software can receive input from its door and from a start button, and can send signals to switch an internal light, and the heater, on or off. The software can also send signals to a timer to set the cooking time and can receive a signal from the timer when cooking is complete.
- Cooking starts with pressing the start button, provided the door is closed.
- Opening the door during cooking turns the heater off.
- Whilst cooking or whilst the door is open, the cooker light is on.
- The cooking time is set in multiples of a minute.
- Each time the start button is pushed adds one minute to the cooking time.
- When the timer stops, either because the door is opened whilst cooking is in progress, or because the timer signals that cooking is completed, the timer resets itself to zero.
- The initialization of the cooker software is out of the scope of this case. Assume the power is on and the cooker is in a 'standby' state.

Context diagram

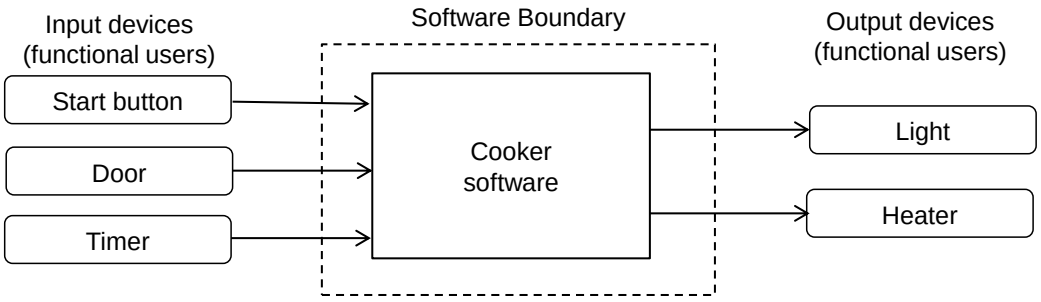

Figure 4.4 – Cooker, context diagram

The state transition diagram of the cooker is shown in Figure 4.5. Boxes represent states and arrows represent the transitions from one state to another (possibly the same state). The events which cause the cooker to move between its states are triggering events. These are prefixed by 'TE' and the functional users that sense the events by 'FU'.

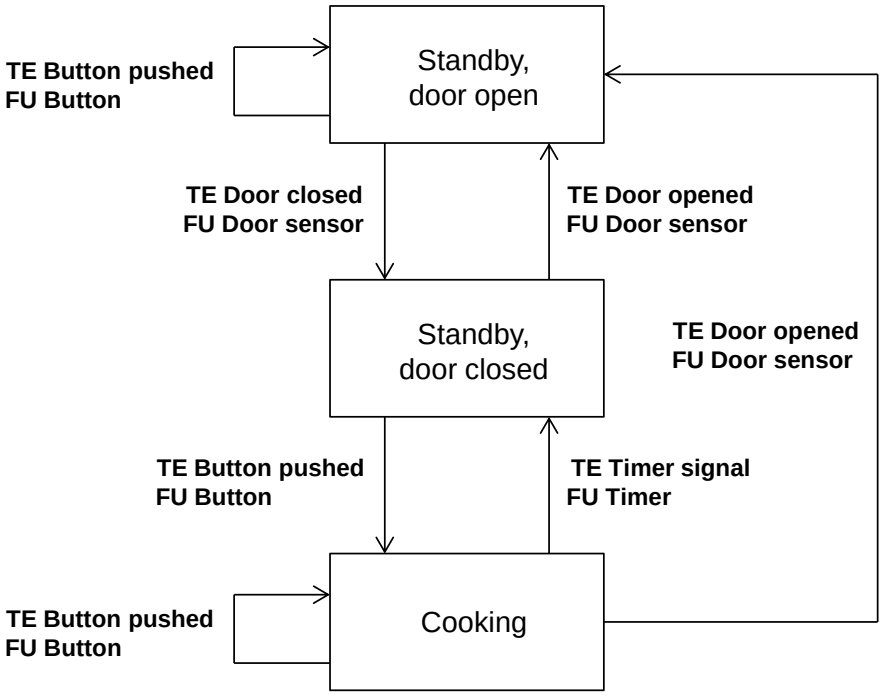

Figure 4.5 – Cooker, state transition diagram

Analysis

The functional users of the cooker software on the input side are the door sensor and the push button. On the output side the functional users are the cooker light and the heater. The functional user that is on both the input and the output side is the timer. As noted in section 3.2.1, in this case the object of interest of each data group entering the software is also the functional user that sent the group (i.e. the functional user is sending data about itself) and similarly the object of interest of each data group that leaves the software is also the functional user that receives the group (i.e. the functional user is being sent data about itself).

The events that actually trigger the software to start a functional process are as follows. As there is here a one-one correspondence between triggering events and functional processes, the same name is used for both.

Triggering event	Functional user that initiates the functional process	Functional process
Door closed	Door sensor	Door closed
Button pushed	Push button	Button pushed
Timer signal (cooking ended)	Clock	Timer signal (cooking ended)
Door opened	Door sensor	Door opened

Table 4.2 - Cooker, triggering events and functional processes

The functional processes of the cooker are as follows:

Functional process: Door closed

DM	Data group	Remark
E	Door closed signal	Triggering Entry, from door sensor
X	Cooker light off signal	To cooker light

The size of this functional process is 2 CFP.

Functional process: Button pushed

DM	Data group	Remark
E	Push button signal	Triggering Entry, from push button
E	Request door status	Verify if door closed[3]
X	Heater on signal	To heater, if door is closed
X	Cooker light on signal	To cooker light, if door is closed
X	Start and/or increment cooking time by one minute	To timer, if door is closed

The size of this functional process is 5 CFP.

Functional process: Clock signal (cooking ended)

DM	Data group	Remark
E	Timer signal	Triggering Entry, from timer
X	Heater off signal	To heater
X	Cooker light off signal	To cooker light

The size of this functional process is 3 CFP.

Functional process: Door opened

DM	Data group	Remark
E	Door opened signal	Triggering Entry, from door sensor
X	Cooker light on signal	To cooker light
X	Heater off signal	To heater
X	Stop timer	To timer

The size of this functional process is 4 CFP.

The total functional size of the cooker software in the scope is 2 + 5 + 3 + 4 = 14 CFP.

Discussion

Note an important point about interpreting state transition diagrams. Not all state transitions correspond to functional processes. In this example there are seven state transitions but only four functional processes. Only events detected by or generated by a functional user external to the software can trigger a functional process. Each functional process must deal with all states and state combinations that it can encounter when responding to a given triggering event.

As an example, the triggering event ‘button pushed’ can occur when the cooker is in each of the three states. The event of the button being pushed takes place in the external world of the hardware and is entirely independent of the state of the machine. The one functional process that must handle the ‘button pushed’ event responds in three ways dependent on the state of the machine at the time the button is pushed namely:

- In the ’standby, door open’ state, it stops after having found that the door is open;
- In the ‘standby, door closed’ state, it sends signals to start the heater and switch on the light, and to start the timer for one minute of cooking;
- In the ‘cooking state’, it executes the same data movements as in the previous state but since the heater has already started and the light is already on, the effect is only to add one minute to the total cooking time.

In this example, we have assumed that the cooker can perform its functions by simply checking if the door is open or closed. In a more complex case, software may need to record the state of the machine

[3] An Entry suffices, assuming the door sensor is ‘dumb’. The functional process, having started, inspects the state of a functional user and retrieves the status data it requires. See Measurement Manual section 3.5.9

and to update it in persistent storage every time the state changes. This would avoid the need for the software to determine the state of the machine each time a new event is signaled.

Similarly, the ‘door opened’ event can occur when the machine is in two states. The one corresponding functional process must deal with the two states.

4.5 Tire-pressure monitoring system

Requirements

- A tire-pressure monitoring system (TPMS) monitors the pressure of each of the four tires of a car.
- Each wheel has a sensor which obtains the pressure of its tire.
- As soon as the car’s electrical power supply is turned on, a clock activates the TPMS software once per minute to retrieve the status of the four sensors, whether the car is moving or not. The sensors return their status, consisting of the sensor id (which identifies the particular wheel) and tire pressure.
- If the pressure is too low or too high - the values are in the software - the TPMS turns on the relevant red warning LED(s) at the dashboard (the sensor location is therefore relevant).
- If the pressure becomes normal again, the TPMS switches off the relevant red warning LED(s) at the dashboard.
- The TPMS electronic control unit (ECU), the clock, the tire pressure sensors and the dashboard LED’s are coupled by a CAN-bus (CAN = controller–area network).

The purpose of the measurement is to size the functionality of the TPMS.

Context diagram

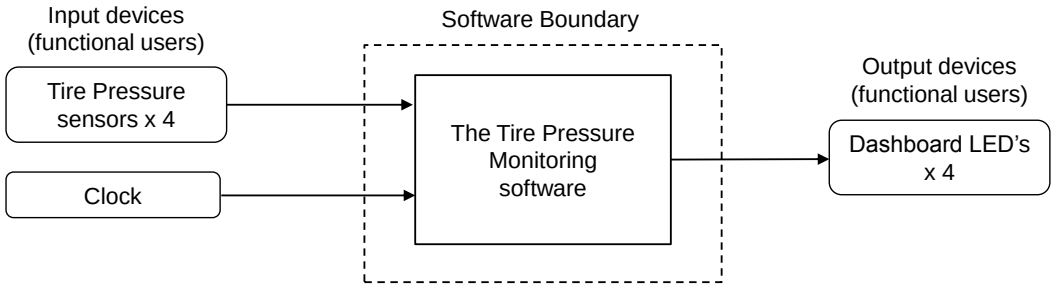

Figure 4.6 – TPMS, context diagram

Analysis

The two functional users of the TPMS software on the input side are the clock and the array of the four tire sensors. On the output side the functional user is the array of the four dashboard LED’s, which correspond with the four wheels of the car.

The CAN-bus controllers form a collection of software that together provides a cohesive set of services that the TPMS software can utilize and are therefore in a software layer that is separate from the layer in which the TPMS software resides. The network controllers are therefore not in the scope of this measurement. Note that if they were within the scope of the measurement, they must be measured separately as the controllers are software in another layer.

The software must respond to one triggering event, the clock signal that is sent every 60 seconds, so has one functional process:

Triggering event	Functional user that initiates the functional process	Functional process
Clock signal	Clock	Start TPMS software

Table 4.3 - TPMS, triggering events and functional processes

As noted in section 3.2.1, in this case the object of interest of each data group entering the software is also the functional user that sent the group (i.e. the functional user is sending data about itself) and similarly the object of interest of each data group that leaves the software is also the functional user that receives the group (i.e. the functional user is being sent data about itself). The data groups are

- the clock signal, with object of interest the clock
- the tire pressure which occurs four times, each group consisting of the sensor id and associated tire pressure, with one object of interest, the wheel pressure sensor
- the signals to turn on or off one or more warning LED's of the dashboard, with one object of interest, the warning LED's

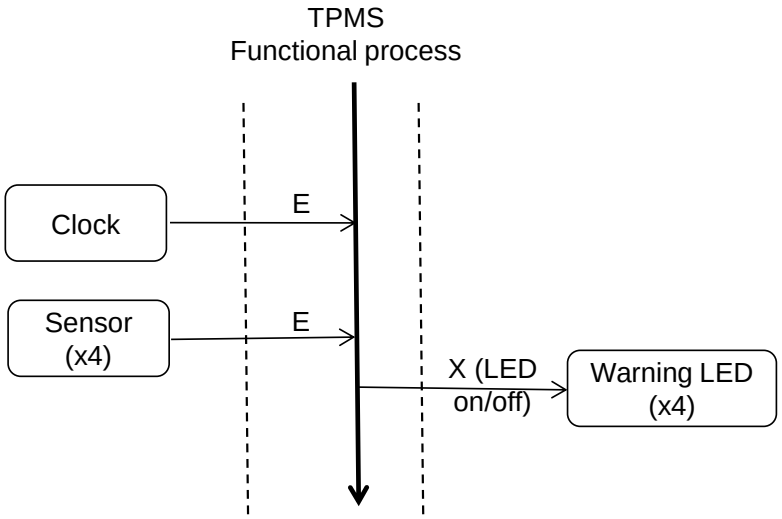

Figure 4.7 - Message sequence diagram of the one functional process of the TPMS

There is no requirement to store or retrieve any persistent data. The functional process and data movements of the process are as follows. The data movements, the data groups moved and an explanation are shown for the functional process.

Functional process: Start TPMS software

DM	Data group	Remark
E	Start signal	Triggering Entry, by clock
E	Tire pressures	Acquisition of the four 'dumb' sensors' data, one for each wheel
X	Signal to array of four warning LED's	LED on, if needed, one for each wheel. Check of pressure is data manipulation and accounted for by this Exit data movement

The size of this functional process is 3 CFP.

Note that although the four 'dumb' sensors are not identical they must be separately identified by the software However, the four requests for tire pressure data are separate occurrences of the movement of the same data group. Therefore, according to the data movement uniqueness rules (see the Measurement Manual [1], only one Entry data movement must be identified. The data group moved by this Entry would carry two attributes – the wheel id and the corresponding tire pressure value. As there is one Entry, there also is one object of interest involved, tire pressure sensor'. The same applies to the Exit that conveys the status data to the four LED's.

This simple case illustrates an important aspect of many real-time systems that have multiple occurrences of identical sensors or output devices. Examples would be multiple sensors (all identical apart from their location) distributed across a sheet of material passing through a set of rollers controlled by a process control system. Where such sensors (or output display devices) are identical and the functional processing of all Entry (or Exit) data movements is identical, count the number of types of data movements for size measurement purpose, not the number of occurrences.

4.6 Automation of sizing real-time requirements

Automation of the measurement of requirements for real-time embedded software of vehicle Electronic Control Units modeled with the Matlab Simulink tool is described in [25] and in the PhD thesis of Hassan Soubra [13]. A concise English description of the method, copyright Renault, is available from the download section of www.cosmic-sizing.org [18].

Automation of the measurement of requirements expressed in UML (not specifically of real-time software) is described in [19].

4.7 Measurement of data manipulation-rich real-time software

An example in this section illustrates that the assumption is reasonable that data manipulation functionality (or 'algorithms') of real-time software can be accounted for by the COSMIC method. The example does not, of course, prove that the assumption is always reasonable. For ways in which to deal with software for which it is known that certain areas of the functionality have a high concentration of data manipulation, see section 3.2.3 of this Guideline.

The distribution of algorithms in some avionics software

A large component of the software of a very complex real-time avionics system was measured using the COSMIC method. The total size of the requirements (held in a modelling tool) for the component was over 8000 CFP. Implementation required over 80,000 lines of source code in the Ada language.

This one system component consisted of 33 sub-components[4]. Within each sub-component, the number of lines of Ada code associated with each data movement was also counted. This is known as the 'NOLA', for 'number of lines of algorithm'. Hence the 'NOLA per Data Movement' could be calculated for each of the 8000+ data movements.

Figure 4.8 shows a histogram of the frequency of the 'NOLA per Data Movement' for all except five of the data movements. The five exceptional data movements had 28 (x2), 36, 40 and 138 NOLA. (Example: the histogram shows that 20 of the 8000+ data movements had seven NOLA.)

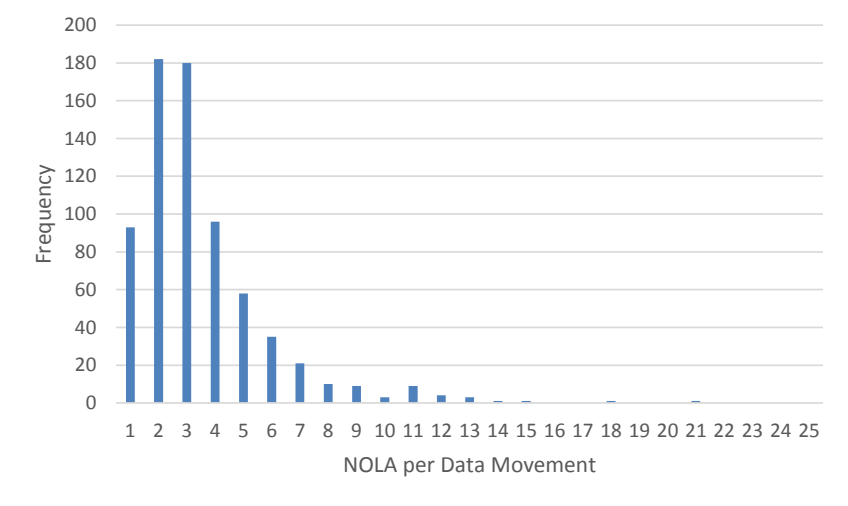

Figure 4.8 Frequency of 'Number of Lines of Algorithm (NOLA) per Data Movement

The following parameters were derived from these data:

Parameter	Value
Median NOLA per Data Movement	2.4
Mean average NOLA per Data Movement	3.5
Data Movement upper size limit which accounts for 95% of the total NOLA	8 CFP
Data Movement upper size limit which accounts for 99% of the total NOLA	14 CFP

[4] Private client data.

The data and the analysis indicate that the NOLA per Data Movement values have a limited range, apart from a very few exceptions. This finding supports the COSMIC method assumption that a count of data movements reflects the amount of data manipulation and thus is a good reflection of the functional size, at least for this particular piece of real-time software.

4.8 Sizing the memory requirements of the functionality of vehicle Electronic Control Units

A paper entitled ' On the Conversion between the Sizes of Software Products in the Life Cycle' by C. Gencel, R. Heldal and K. Lind, presented at the International Workshop on Software Measurement in Stuttgart, November 2010, describes the application of the COSMIC method to size the software embedded in Electronic Control Units of Saab cars, manufactured in Sweden. The purpose of the study was to examine the relationship between the COSMIC-measured functional size and the resulting memory space needed by the object code, measured in bytes. An extremely good linear correlation was found.

In the paper, the authors state: 'This paper shows that it is possible to obtain accurate code size estimates even for software components containing complex calculations, as long as the components contain similar complexity proportional to the number of component interfaces.'

Renault [26] also reported a good correlation of code size in bytes versus COSMIC-measured Functional size in units of CFP, as in the graph below.

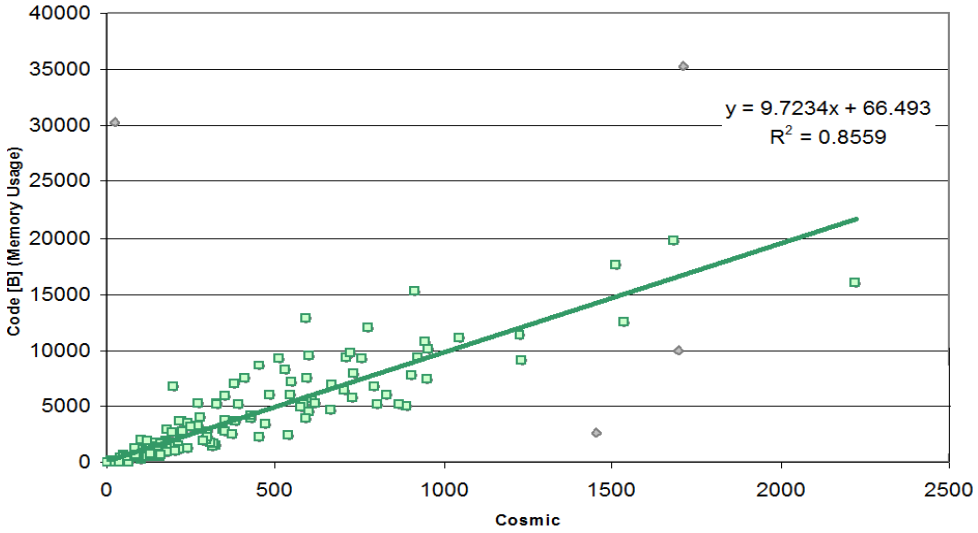

Figure 10. Code size (bytes) versus COSMIC functional size (CFP)

REFERENCES

All the COSMIC documents and the documents indicated by (*) listed below, including translations into other languages, can be obtained from the download section of www.cosmic-sizing.org.

[1] Measurement Manual, v4.0, April 2014 and v4.0.1.*

[2] ISO 14143:2007 Software Engineering – Software Measurement – Functional Size Measurement, Part 5, Determination of Functional Domains for Use with Functional Size Measurement.

[3] Sommerville, I., Software Engineering, Addison-Wesley Publishing Company, 1995.

[4] EARS (Easy Approach to Requirements Syntax). 17th IEEE Engineers Requirements Conference, 2009.

[5] ISO 61131-3 Programmable controllers – Part 3: Programming languages, 2003, URL: www.iec.ch.

[6] Guideline for approximate COSMIC functional size measurement (*).

[7] Requirements Engineering Tools, IEEE Computer Society, 2010.

[8] Lind, K., Heldal, R., Harutyunyan, T., Heimdahl, T., CompSize: Automated Size Estimation of Embedded Software Components, IWSM conference 2011, Nara, Japan.

[9] Lind, K., Heldal, R., A Practical Approach to Size Estimation of Embedded Software Components, Approved for publication in IEEE Transactions on Software Engineering, 2011.

[10] Stern, S., Practical Experimentations with the COSMIC Method in the Automotive Embedded Software Field, section 4.2 of COSMIC Function Points: Theory and Advanced Practices, CRC Press, 2011, pp. 237-246.

[11] Soubra, H., Abran, A., Stern, S., Ramdan-Cherif, A., Design of a Functional Size Measurement Procedure for Real-Time Embedded Software Requirements Expressed using the Simulink Model, 21st International Workshop on Software Measurement – 6th International Conference on Software Process and Product Measurement – IWSM-Mensura 2011, Nara (Japan), Nov. 3-4, 2011, Editor: IEEE Computer Society, pp. 76-85.

[12] Soubra, H., PhD thesis Automation de la mesure fonctionnelle COSMIC-ISO 19761 des logiciels temps-réel embarqué, en se basant sur leurs 34pecifications fonctionnelles. Ecole de technologie supérieure (ETS), Université du Québec and Université de Versailles at St-Quentin, in collaboration with Renaults SAS.

[13] Lavazza, L., Del Bianco, V., A case study in COSMIC functional size measurement: the Rice Cooker revisited, IWSM/Mensura conference 2009 (*).

[14] Al-Sarayreh, K.T. and A. Abran, Specification and Measurement of System Configuration Non Functional Requirements, 20th International Workshop on Software Measurement (IWSM 2010), Stuttgart, Germany, 2010.

[15] Symons, C., Accounting for non-functional requirements in productivity measurement, benchmarking and estimating, UKSMA/COSMIC International Conference on Software Metrics & Estimating, London, UK, October 2011, URL: www.uksma.co.uk.

[16] Butcher, C., Delivering Mission-Critical Systems,,British Computer Society meeting, London, 18th November 2010.

[17] Toivonen, H., Defining measures for memory efficiency of the software in mobile terminals International Workshop on Software Metrics, 2002.

[18] COSMIC Rules for Embedded Software Requirements Expressed using Simulink, published by Renault 2012 (*).

[19] ‘Automatic COSMIC sizing of requirements held in UML’, Jaroslaw Swierczek, COSMIC Masterclass, IWSM 2014, Rotterdam (*)
[20] ‘Guideline for ‘Measurement Strategy Patterns’ (*)
[21] Diab, H., Koukane F., Frappier M., St-Denis R. “µcROSE: Functional Size Measurement for Rational Rose RealTime.” (2002)
[22] Soubra, H., and Chaaban, K. “Functional Size Measurement of Electronic Control Units Software Designed Following the AUTOSAR Standard: A Measurement Guideline Based on the COSMIC ISO 19761 Standard.” (2012).
[23] Soubra, H. “Fast Functional Size Measurement with Synchronous Languages: An Approach Based on LUSTRE and on the Cosmic ISO 19761 Standard.” IWSM-MENSURA, 2013. (*)
[24] Guideline on how to account for Non-Functional Requirements in software project performance measurement, benchmarking and estimating’ (under development) *
[25] Soubra, H., Abran, A., Stern, S., Ramdan-Cherif, A., “Design of a Functional Size Measurement Procedure for Real-Time Embedded Software Requirements Expressed using the Simulink Model”. IWSM-MENSURA, Nara, Japan, 2011.
[26] Stern, S., Gencel, C., Embedded software memory size estimation using COSMIC: a case study, IWSM 2010 (*)

REAL-TIME DOMAIN GLOSSARY

Actuator. A device that converts an electrical signal to a mechanical movement.

Clock: (as used in this Guideline) A device that generates a stream of pulses at constant frequency.

Control. Directing or guiding

Dumb device. Any type of device that sends its data automatically on receipt of a prompt message.

NOTE: For measurement purposes, an Entry data movement to the software needing the data suffices to obtain the data from a dumb device.

Intelligent device. Any type of device that must be told explicitly what data is required in order for it to produce the required output data.

NOTE: For measurement purposes, an Exit data movement from the software specifying the required data is received by an intelligent device as an Entry. The device issues an Exit conveying the requested data which is received by the requesting software as an Entry.

Monitor. Keeping a check on something

Sensor. A device that converts a physical signal to an electrical signal that is made available to the software in the form of a data group.

Timer: (as used in this Guideline) A device or software process that measures a time duration.

APPENDIX A – THE MAIN CHANGES FROM VERSION 1.0 TO VERSION 1.1

This Appendix contains a summary of the principal changes made in the evolution of this 'Guideline for sizing real-time software' from version 1.0 to the present version 1.1.

V4.0 Ref	Change from version 1.0
Foreword	The definition of 'real-time' as used in this Guideline is discussed in more detail.
1.2.3	Example 2 of a lift door closing has been made more realistic by describing a closing sequence requiring two functional processes rather than one.
1.2.4	The reference to the IEC 61131-3 standard has been updated to the 2013 version.
2.2	The text of v1.0 used 'clock' and 'timer' inconsistently. (In 2.2 an example of a functional user was given as 'clock (timer)'. Use of the two terms has been rationalized and they are now defined in the Glossary.
2.2	Example 3c) wrongly stated that there is only one functional user. In fact, as the button in the supervisor's office has a function that is not available to any button on the line, the software must be able to distinguish the button in the supervisor's office from the other buttons. So there are two functional users. The same reasoning applies to Example 3d) which now has three functional users. Example 3d) wrongly referred to 'As example b)'. It has been corrected to 'As example c)'.
3.1	The description of the process to identify triggering events and functional processes has been modified to bring it in line with v4.0/4.0.1.
3.2.2	A note in the top left-hand corner of Figure 3.2 stated 'Triggering Entry plus any other Entries for the same FP from the same functional user'. This was probably misleading in the context of real-time software, so has been deleted. In real-time software, a hardware or software functional user that initiates a functional process will probably send one data group conveying all its data describing the event to the one triggering Entry. It would not normally send any subsequent data describing a different object of interest.
3.2.4	A new section discusses the measurement of error or fault messages in real-time software.
4.1.2	The statement of requirements for the PLC has been re-written to make it clearer. In the functional process 'Stop Pump', two rows have been switched round. It is safer to stop the pump before closing the outlet valve.
4.2 (formerly section 3.2.4)	Examples 1 and 2 have been corrected to make the distinction between a 'clock' and 'timer' functionality clearer. (See also the correction for section 2.2 above). In Figure 3.3, the 'delete request' is correctly shown as a Write. In the text it is wrongly described as an Exit. The text has been corrected to state 'Write'.
4.3	In the context diagram of Figure 4.2, 'Mains' has been replaced by the more commonly-understood 'Power'.
4.3	'Clock' has been replaced by 'timer'
4,5	'Persistent storage' has been removed from the Context Diagram for the Tire Pressure Monitoring System (it is not used, and the explanation improved.
4.5	In the context diagram of Figure 4.5, 'timer' has been replaced by 'clock' and the 'Ignition key' has been removed as it is not a functional user for the purposes of this

	Case. Turning the ignition key starts the clock that controls the TPMS software.
4.7	A new section has been added giving an example of real-time software where it is shown that a count of data movements is a good measure of the software's functionality because it can also reasonably account for data manipulation functionality.
4.8	A new section gives examples of how the COSMIC method has been used successfully to estimate the memory size requirements for vehicle electronic control units.
Glossary	The terms 'clock' and 'timer' are now defined as used in this Guideline.

APPENDIX B - COSMIC CHANGE REQUEST AND COMMENT PROCEDURE

The COSMIC Measurement Practices Committee (MPC) is very eager to receive feedback, comments and, if needed, Change Requests for this guideline. This appendix sets out how to communicate with the COSMIC MPC.

All communications to the COSMIC MPC should be sent by e-mail to the following address:

mpc-chair@cosmic-sizing.org

Informal general feedback and comments

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

Formal change requests

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

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

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

A form for submitting a CR is available from the www.cosmic-sizing.org site.

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

Questions on the application of the COSMIC method

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