COSMIC

The COSMIC Functional Size Measurement Method

Version 4.0.2

Guideline for Sizing
Service-Oriented Architecture Software

VERSION 1.1.1

January 2019

Foreword

Purpose of the Guideline and relationship to the Measurement Manual

The purpose of this Guideline is to provide additional advice beyond that given in the COSMIC Measurement Manual [1] on how to apply the COSMIC Functional Size Measurement (FSM) method v4.0.2 to size software called 'services' from the architecture generally referred to as the 'Service Oriented Architecture' (SOA).

The COSMIC Measurement Manual contains the concept definitions, principles, rules, and measurement processes of the COSMIC method. It also contains much explanatory text on the concepts, plus examples of application of the method. This Guideline expands on the explanatory text and provides additional detailed guidance and more examples for sizing services than can be provided in the Measurement Manual

SOA-based software is a special type of software. Sizing service-oriented software using traditional ('1st generation') FSM methods raises particular difficulties when reconstructing or mapping the Functional User Requirements (FUR) of the software services to the measurement model(s), due to the nature of the SOA designs. The COSMIC method defines and standardizes particular concepts, such as layers, components, the unlimited size of a functional process, and that pieces of software can be functional users of each other. These concepts are perfectly suited for measuring SOA-based software designs [2], without needing to adapt the method in any way.

Intended readership of the Guideline

The Guideline is intended to be used by expert Measurers who have the task of measuring the functional size of software services according to the COSMIC method. It should also be of interest to those who have to interpret and use the results of such measurements in the context of project performance measurement, software contract control, estimating, etc. The Guideline is not tied to any particular development methodology or life-cycle model.

Readers of this Guideline are assumed to be familiar with the COSMIC Measurement Manual, version 4.0.2 [1], and with the Business Applications Guideline version 1.3 [3]. For ease of maintenance, there is little duplication of material between these two documents and this Guideline.

Scope of applicability of this Guideline

The content of this Guideline is intended to apply to the measurement of services of a SOA based software application. The concept of SOA is being widely adopted in the domain of business application software. All examples in this Guideline relate to that domain. Some examples of message system functionality are also given, where 'message system' may be loosely defined for this Guideline as 'utility software that supports SOA software by enabling communication between SOA components'.

> *N.B. When sizing a whole business application from the viewpoint of its functional users, e.g. humans and/or other pieces of software, the normal rules of the Measurement Manual apply independently of whether the underlying application architecture is based on a SOA or any other structure. This Guideline is only concerned with sizing the various types of SOA components*

Introduction to the terminology and contents of the Guideline

For definitions of the terms of the COSMIC method in general, please refer to the Glossary in the Measurement Manual [1]. Terms specific for the service oriented architecture are defined in the glossary in Appendix B of this Guideline. We note that literature on information technology uses several terms with various meanings, but which are defined in the COSMIC method with one very specific meaning. The measurer must therefore be careful to correctly apply the terminology of the COSMIC method when using this Guideline.

Chapter 1 discusses the SOA in general terms and the properties that characterize the functionality of software services. It treats some aspects of how FUR for such services are developed, as far as relevant for measurement. It provides practical guidance and examples on applying the COSMIC Generic Software Model to software services.

Chapter 2 discusses the Measurement Strategy phase for sizing services, by considering the five key parameters of the measurement that must be addressed, namely the *purpose* and *scope* of the measurement, the identification of *functional users*, the *level of decomposition* of the software and the *level of granularity* of the FUR that should be measured.

Chapter 3 discusses the sizing of services through both the Mapping and the Measurement phases. In the Mapping phase the FUR of the software to be measured must be mapped to four main concepts of the COSMIC method, namely *events* and *functional processes*, *objects of interest*, *data groups* and the individual *data movements* of the functional processes, as the basis for measuring the functional size. In the next Measurement phase, the data movements must be counted to obtain the functional size. As almost all examples treat both the functional processes and data movements together, both phases are considered in this one chapter.

Chapter 4 discusses a number of important points on performance measurement and project estimating with SOA software.

Table of Contents

1 SOA AND THE COSMIC GENERIC SOFTWARE MODEL 5

1.1 The Service Oriented Architecture 5

1.1.1 Services 5

1.1.2 Functional User Requirements and Non-functional Requirements of services 6

1.1.3 Service Oriented Architectures in practice 7

1.1.4 SOA, business processes and the Enterprise Service Bus 8

1.2 Characteristics of services in the Service Oriented Architecture 8

1.2.1 Application services 9

1.2.2 Orchestration services 9

1.2.3 Intermediary services 10

1.2.4 Utility services 11

1.2.5 Web services 11

2 THE MEASUREMENT STRATEGY PHASE 12

2.1 The purpose of the measurement 12

2.2 The scope of the measurement 12

2.2.1 Layers and components 13

2.2.2 Data movements of data exchanges between components 14

2.3 The functional users of services 15

2.4 The level of decomposition of the software 16

2.5 The level of granularity of the measurement 16

2.5.1 FUR at the conceptual level of granularity 16

2.5.2 FUR at the detailed level of granularity 16

3 THE MAPPING AND MEASUREMENT PHASES 17

3.1 Identifying functional processes 17

3.1.1 The impact of communications requirements on functional processes 17

3.1.2 The called functional process(es) of the services 18

3.2 Identification of objects of interest, data groups and data movements 19

3.2.1 Messages with headers 19

3.2.2 Business-related services 20

3.2.3 Error/confirmation and fault messages 22

3.3 Measurement of the size of functional changes to services 23

4 PERFORMANCE MEASUREMENT AND PROJECT ESTIMATING WITH SOA 25

4.1 Productivity of developing services 25

4.2 Estimating 25

4.2.1 Sizing the whole, or at component level 25

4.2.2 Creating and modifying services and estimating effort 26

REFERENCES 27

GLOSSARY OF SOA TERMS 28

APPENDICES 29

A.1 ACKNOWLEDGEMENTS 29

A.2 VERSION CONTROL 29

A.3 GUIDELINE CHANGE HISTORY 30

A.4 CHANGE REQUESTS, COMMENTS, QUESTIONS 31

SOA AND THE COSMIC GENERIC SOFTWARE MODEL

1.1 The Service Oriented Architecture

1.1.1 Services

A 'software architecture' is defined as the 'fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.' [4]

There is no widely-agreed definition of 'Service Oriented Architecture' (SOA) other than its literal translation that it is an architecture that relies on 'service orientation' as its fundamental design principle. The Organization for the Advancement of Structured Information Standards (OASIS) defines SOA as 'a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations' [5]. The software 'capabilities' referred to above will be called 'services' in this Guideline.

The key characteristic of SOA is that independent services with standard interfaces can be called to perform their tasks in a standard way, without the service having foreknowledge of the calling software (an application or another service), and without the calling software having or needing knowledge of how the service actually performs its tasks.

Services are also made available without the software that uses the service needing to know anything about the service's underlying platform implementation or programming language. SOA may be implemented using a wide range of technologies and protocols.

In SOA, any application or any service requiring commonly-used information from another service or application sends a request to the service or application that can handle the request. Requests and replies are implemented as 'messages'. SOA standards govern the message exchange e.g. message structure, data formats, routing, security etc.

These messages between applications and services are commonly physically handled by means of an 'Enterprise Service Bus', which in this Guideline will be indicated by 'message system' (see more in sections 1.1.4 and 3.2.1) or by 'Intermediary Services' (see section 1.2.3). However, neither the calling software nor the called service need to know anything about the support from a message system or any other software that enables communication between the calling software and the called service.

Upon receipt of a request message the service processes it and, if needed, returns the results to the requesting software in the form of a reply message. An application may call its own services; such calls also have the form of messages. Note that a message may consist of one or more data movement types.

The model for a common form of exchange of messages between an application and a service is shown in Figure 1.1. It uses the standard COSMIC model for the exchange of data between two peer pieces of software.

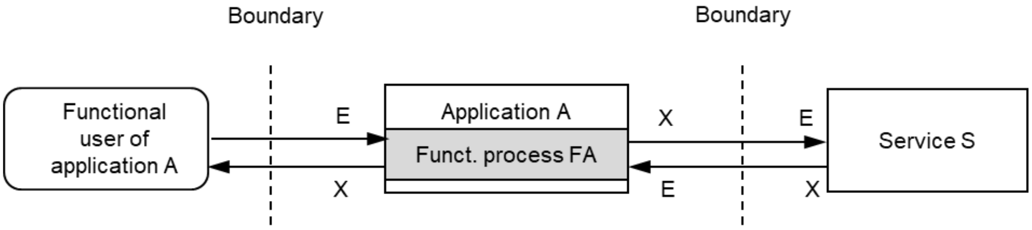


Figure 1.1 – The interactions between an application and a service

An application A requires some data from a service S. Application A is a functional user of the service S, and vice versa.

A functional process FA is triggered in the requesting application A which issues a request message as an Exit. This message is received by the service S (which we assume consists of a single functional process) as a triggering Entry. The service S replies to FA with a message containing the requested data or a message that indicates an error (for error messages from SOA components, see section 3.2.3). In COSMIC terminology, the exchange of messages takes place via an 'Exit/Entry pair' of data movements from/to the requesting application A (outgoing request/incoming reply). These messages are seen as an 'Entry/Exit pair' of data movements by the supplying service S (incoming request/outgoing reply).

Note that the service S could be independent, or part of another piece of software or application. Equally the functional process of the calling application A could be a service of that application. Figure 1.1 does NOT show how the functional process of service S obtains the data that it must provide to the application A. The service S may obtain the needed data:

- by computation (no additional data movements needed),
- or from persistent storage within its boundary (requiring a Read),
- or from another application or service outside its boundary (needing another Exit/Entry pair),
- or by a combination of any of these three possibilities.

1.1.2 Functional User Requirements and Non-functional Requirements of services

The use of the term 'FUR' is restricted to mean 'the functional user requirements that are derived from the available software artefacts (requirements, designs, physical artifacts, etc.), after adjustment by assumptions to overcome uncertainties in the available artefacts'. FUR must therefore contain all the information needed for a precise COSMIC Functional Size Measurement. (If we need to describe a situation where there is not yet enough detail for a precise COSMIC measurement, we use the term 'functional requirements'.)

The FUR of a service define:

a) the capability (service) it provides for a service requestor,

b) how to request the capability, and

c) the form and content of the request and reply messages.

Such an information set is usually called the 'service contract'.

Any other requirement for software than a functional requirement is a non-functional requirement (NFR). COSMIC defines NFR's as below.

DEFINITION – Non-Functional Requirement (of a software/system product)

Any requirement for a hardware/software system or for a software product, including how it should be developed and maintained, and how it should perform in operation, except any functional user requirement for the software. Non-functional requirements concern:

- the software system or software product quality;
- the environment in which the software system or software product must be implemented and which it must serve;
- the processes and technology to be used to develop and maintain the software system or software product, and the technology to be used for their execution.

Most NFR's are stated at the level of a whole hardware/software system or of a whole piece of software, e.g. an application, rather than at the level of individual SOA components. However, examples of requirements for a SOA component that are clearly NFR's according to the COSMIC definition are that the component must be 'written in a particular programming language, conform to specific SOA protocols and be subject to certain timing constraints. Such 'true' NFR are irrelevant to software functional size measurement.

However, some requirements that are often considered as non-functional, such as security requirements, may be implemented as software when needed for SOA components. The size of such requirements can be measured using COSMIC, if within the measurement scope.

> *EXAMPLE: WS-Security (standard for the SOAP protocol) or Oath (popular in RESTful SOA environments) define security mechanisms and information (e.g. user.name, password) that must be exchanged between services.*

1.1.3 Service Oriented Architectures in practice

As there is no widely-agreed definition of 'SOA', it is not surprising to find that the ideas that drive SOA have been solved in different types of service architectures. In this Guideline the COSMIC method is applied to a typical service architecture, which should be clear enough for the Measurer to adapt and apply the approach described here to his/her own situation. This Guideline thus intends 'to show the wood by means of its trees'[1].

Communication with or between services may take place via 'intermediary' services. Intermediary services (see more in section 1.2.3) fulfil two important purposes, which at first sight seem to contradict. The first and obvious purpose of the intermediary services is to enable service 'requestors' to communicate with service 'providers', where some translation of the request and the response is needed. The second purpose of the intermediary service is to ensure that service requestors and service providers are independent.

From a practical viewpoint, this second purpose is no less important than the first. Independence of service requestors and service providers ensures flexibility as it allows service-providing applications to be changed without needing to change service-requesting applications, and vice versa. Also it allows changes to be made to the intermediary services, when required, without affecting the applications themselves.

> *EXAMPLE. Suppose a service architecture in which the main services are generic parts of existing applications, performing tasks like 'Register customer' or 'Show customer details' (these are denoted as 'application services'). When another application needs*

[1] Bertrand Russell.

these service(s) the application may call an 'intermediary service' which in turn calls the required application service(s). Suppose the application service 'Show customer details' supplies customer data without the country code that is needed by another application. An intermediary service could add the country code. The intermediary service ensures the flexibility referred to above.

1.1.4 SOA, business processes and the Enterprise Service Bus

In SOA, applications and services may interact by sending messages via an 'Enterprise Service Bus' (ESB). An ESB is the central communication infrastructure at the Enterprise level, consisting of an advanced [2] message system, and a variety of functionality for connecting applications, such as data and message transformation and work flow management (definition and execution of business process flows).

ESB software is regarded as 'middleware' and resides in a different layer from application services. (For more on layers, see section 2.2.1,' Layers and components'.)

In SOA terminology, the definition and execution of business process flows is termed *orchestration.* This may be provided at the Enterprise level, i.e. across applications from software in another layer, or at the level of individual applications in the application layer. Orchestration enables processes to be implemented by calling applications and services in the desired order.

The business process flow controlled by the orchestration capability of an ESB is independent of any other application functionality and the flow may be changed without changing the underlying applications and services. A consequence of the ESB is that pieces of software don't communicate directly with each other, rather they communicate through the ESB, as shown in Figure 1.2

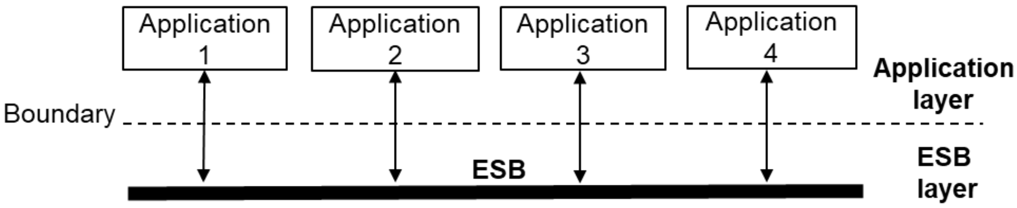


Figure 1.2 - Software interacting via the Enterprise Service Bus

1.2 Characteristics of services in the Service Oriented Architecture

In this Guideline services are classified as shown in the Table below. This classification is one of many possibilities, see for instance the classification in [5], and is introduced simply to illustrate SOA measurement principles. Some alternative commonly-used terms for these same characteristics are also listed.

Software services are distinguished by the following characteristics:

- A service offers functionality to its users (normally other software), using a standard interface in the form of messages. For discussion of the different types of communication, see section 3.1.1.
- Services may be present in the same or different layers. For a discussion, see section 2.2.1.

[2] Advanced' refers to software location independency by using abstract names for each service and other pieces of software ('abstract endpoints'), so that when they are moved the sender doesn't need to change its physical location).

Term used	Alternative terms	Description
Application service	Business service, entity service	Provides business functionality of an application
Orchestration service	Process service, task service	Controls ('orchestrates') application services to implement a (business) process
Intermediary service	Internal service, mediation service	Ensures independence of service requestors and service providers
Utility service	Public service, software service	Provides common functionality (business or non-business) independent of, but made available to, any other applications
Web services	--	Services that can be called using internet protocol standards enabling applications to interact over the world wide web

Table - Terms used in this Guideline and alternative industry terms.

1.2.1 Application services

An application service provides a specific, limited business operation. Examples would be 'Create booking' or 'Receive client information'. Such a service may be regarded as an application function of which its 'classical' interface (such as a GUI) has been replaced by an interface consisting of a request/reply mechanism for messages.

Functionality in existing custom-built systems and software packages can be made available as 'services' by either modifying the software or by adding API's (application programming interfaces). When defined as a service, the functionality of the application can be easily re-used, for instance via orchestration services, and therefore needs to be developed only once for use by other applications.

> *EXAMPLE. All applications in a large company use a central security application to satisfy the security needs. Whenever any functional process in an application is accessed by a human user, the application requests user authentication and receives the authorization credentials of the user from the security application. The security application checks authorization via one of its application services. With this approach the need to have specific security checking code in other applications is avoided.*

1.2.2 Orchestration services

As noted in section 1.1.4, orchestration services call and control other services when the latter are needed to implement a (business) process or sub-process. Orchestration services are just services, i.e. are constructed with help of the same standards as other services.

> *EXAMPLE. The process for settling mortgage applications includes a sub-process for risk analysis. The work-flow of this process requires the applicant's credit worthiness test, fraud test, bankruptcy test and identity test. Each of the four tests consists of sending a message to the application in question (i.e. to its specific application service), some of them outside the financial institution, as shown in Figure 1.3. Each message contains the required data and triggers the service to reply 'OK' or 'Not_OK'. The application is rejected if one or more Not_OK's are received. An orchestration service of the mortgage application sub-process controls the sequence of the message calls.*

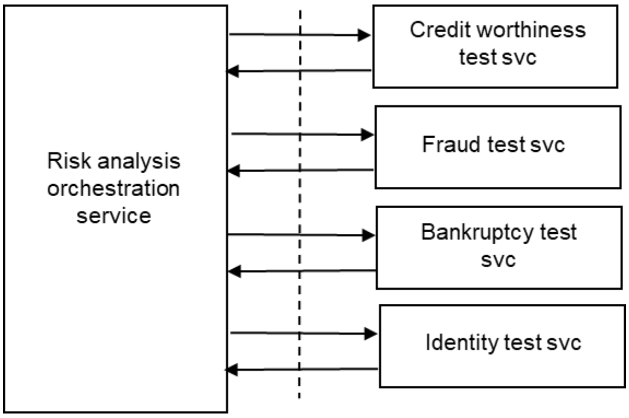

Risk analysis orchestration service
Credit worthiness test svc
Fraud test svc
Bankruptcy test svc
Identity test svc

Figure 1.3 – Risk analysis orchestration service

1.2.3 Intermediary services

As discussed in Example 2 of section 1.1.3, intermediary services are used in some SOA implementations to interconnect a requester's message with one or more application service(s), as shown in Figure. 1.4.

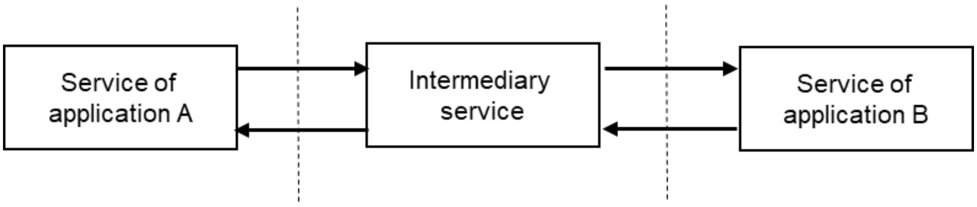


Figure 1.4 – Application services and interconnecting intermediary service

When an application service in application A requires data that is available via a service in application B, the former application service calls the intermediary service, which may fulfil any or all the following tasks, any of which may themselves have been realized as a separate utility service:

a) Control the handling of the request received from (a service of) application A.

b) Translate the ‘language’ of the message from application A into the ‘language’ of application B (and possibly applications C, D, ..., if services of other applications are involved) that must fulfill the request.

c) Call a functional process of the application service of application B by means of the translated message.

d) Receive the reply message from the functional process of application B (and possibly reply messages from applications C, D, …).

e) Translate the results into a message in the language of application A.

f) Send the reply message to the service of application A.

g) Finalize exception situations.

h) Log data about the handling of services.

With regard to task b), differences in ’language’ may refer to a great variety of aspects, such as different programming languages, different field formats, and also to different ‘business languages’. For instance, the meaning of ‘customer’ in a customer management application and in an accounting application may well differ in some aspect. One request for customer information may concern both kinds of customers, whereas for another request only one of those applies. For these reasons the request in the language of application A has to be translated into a request in the language of application B. Conversely, application B's reply 'in B's language' has to be translated back into application A's language. When this translation functionality is also needed by other applications in the overall SOA framework, it may be realized itself in the form of a utility service.

EXAMPLE: (Intermediary and application services) Bank account data may consist of two parts: data of the account agreement and data of the account itself, such as the account number. The agreement details are stored in one application, whilst the account data are stored in the application that processes the payment transactions.

When a bank customer cancels his/her account, the 'Cancel account' function (application service) calls the intermediary service, e.g. to complete the data, which calls the application service of the payment transactions application to close the account. Also, the intermediary service calls the application service of the agreements application to end the agreement (after the necessary checks in both cases, of course).

1.2.4 Utility services

Utility services provide common functionality (business or non-business) independently of, but available to, other applications or services. There is a wide variety of utility services; two examples are given below.

EXAMPLE 1. (Utility service for file parsing) Some applications communicate with external systems through data files. File processing involves reading from a data file and making it eligible for processing. A file parser utility service may have been developed that parses the data files (i.e. removes, adds, combines and/or splits data attributes) in a flexible and maintainable way, supporting the applications by making the changes in the file attributes needed by the external systems.

EXAMPLE 2. (Logging utility service) Logging may be required for a number of purposes: statistics, performance measurement, troubleshooting, charging, and/or usage measurement for service level agreements. Examples are:

- *Application logging* — *Logs status information that allows operators to monitor application status.*
- *Error logging* — *Logs errors in line with company standards.*
- *Performance logging* — *Logs performance data to monitor the performance of the application and/or of external interfaces.*
- *Trace logging* — *Logs information for developers and 3^{rd} line support (maintenance).*

1.2.5 Web services

A web service is a type of software service that relies on protocols and formats frequently used on the web for message transport and data presentation such as HTTP, XML and JSON. Note that a web service need NOT necessarily exist on the World Wide Web; it can run anywhere on a network, intra- or internet.

2

THE MEASUREMENT STRATEGY PHASE

The Measurement Strategy considers five key parameters of the measurement that must be addressed, namely the *purpose* of the measurement, its *scope*, the identification of the *functional users*, the *level of decomposition* of the software and the *level of granularity* of the FUR. See also the ‘Guideline for Measurement Strategy Patterns’ [7] sections on re-usable software components.

2.1 The purpose of the measurement

Often the primary purpose of measuring a functional size of a service is for estimating development or modification effort. However, many other measurement purposes are possible, for instance:

- Comparing the estimated development cost with the purchase price, when the choice is between buying or making a service (‘make-or-buy’).
- Using the size of the services portfolio (i.e. the total of sizes of all services) for estimating application management support.
- Benchmarking, i.e. comparing the productivity of the services development and support department of an organization with the productivity of comparable organizations.

> *EXAMPLE. The purpose is to measure the size of a set of services realized in all projects in some year, for purposes b) and c). It might be sufficiently accurate to measure sizes using an approximation size variant of the COSMIC method to speed up this task [6]. For instance, in an organization it may have been observed that 8 CFP is an acceptable working estimate for an average application service size. Given the number of services in the organization, an estimate of the total size of the services can be determined. An average productivity for replacing the whole portfolio of application services can be used to determine the estimated replacement cost. The same holds for utility services, possibly with a different estimate for an average service size.*

2.2 The scope of the measurement

Defining the scope of a single SOA service in order to measure its size is very obvious. Simply follow the standard COSMIC process for determining the scope and for measuring the size of a single piece of software. However, care is needed to distinguish between cases where the purpose is to measure a piece of software that *uses* multiple SOA services and when the purpose is to measure a *collection* of multiple SOA services.

> *EXAMPLE 1. If the purpose is to measure the size of a whole application that happens to use SOA services then the scope should be defined to include all SOA services that it uses. The total size of the application is then not simply the sum of the size of the application and of all the SOA services that it uses. The internal data movements between the application and its services and between the services themselves must first be eliminated. Follow the measurement aggregation rules of section 4.3 of the Measurement Manual [1] and the explanation of section 4.2.1 of this Guideline.*

EXAMPLE 2: If the purpose is to measure the size of a collection of SOA services, e.g. as part of a process to determine their value or their support costs, then the scope may be defined as the whole collection provided it is sensible to do so, i.e. the services all use the same technology, etc. In this case the size of the individual services may simply be added to get the total size of the collection.

2.2.1 Layers and components

a) A layer as defined by COSMIC consists of software that:

- provides a set of services[3] that software in other layers can utilize and
- there is a relationship which is either *hierarchical* (software in layer A is allowed to use the services provided by software in layer B but not vice versa, or *bi-directional* (software in layer A is allowed to use software in layer B, and vice versa).

b) Figure 2.1 is reproduced from the Measurement Manual [1]. It shows that the definition of what are considered as layers depends on the 'view' of the software architecture. In this Guideline, we adopt the 'view' and terminology of the SOA architect as per the example in column c) of Figure 2.1.

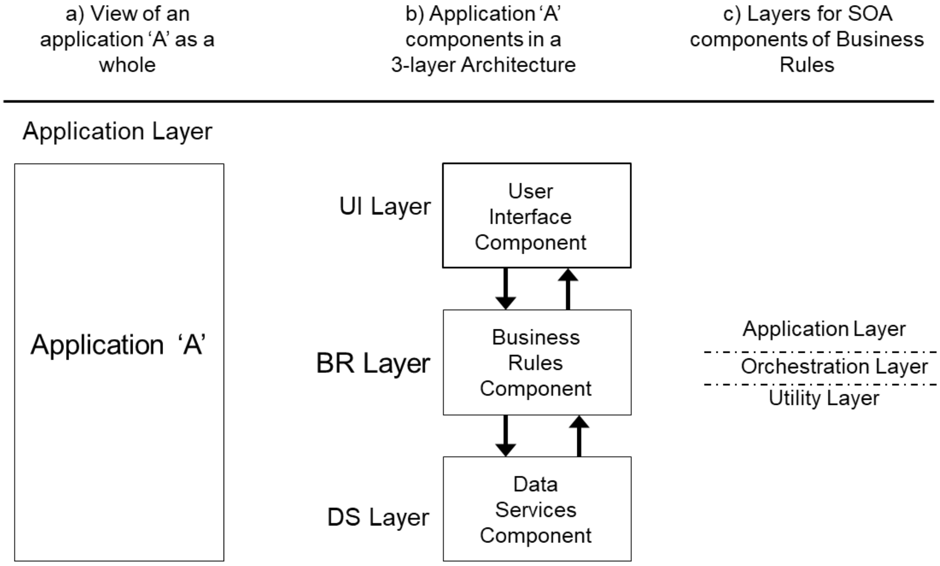


Figure 2.1 SOA Services have their own layered structure (Column c)

In the SOA architect's view:

- Orchestration services can call application services but not vice versa, i.e. there is a hierarchical relationship. These two types of services must therefore be in different layers of an SOA architecture.
- Intermediary services can be called by application services and vice versa. These two types of services are therefore both in the SOA application layer.
- Both orchestration and application services can call utility services. Utility services must therefore be in their own separate SOA layer.
- Web services can be defined in any layer.

When services in different layers must be measured, they must always be measured with separate measurement scopes.

EXAMPLE 1. An application calls a parsing service (as in 1.2.4 above, Example 1) to re-structure some data attributes in preparation for communicating with another application.

[3] 'Service' meant like in normal usage, i.e. not necessarily developed according to SOA principles.

Either the application sends the attributes e.g. X, Y, Z for parsing, or it sends the whole record containing these attributes for a task which is essentially 'parse the attributes X, Y and Z, ignoring the remainder of the record'. Either way, the application uses the parsing service (but not vice versa) in a hierarchical relationship, so they must be in different layers.

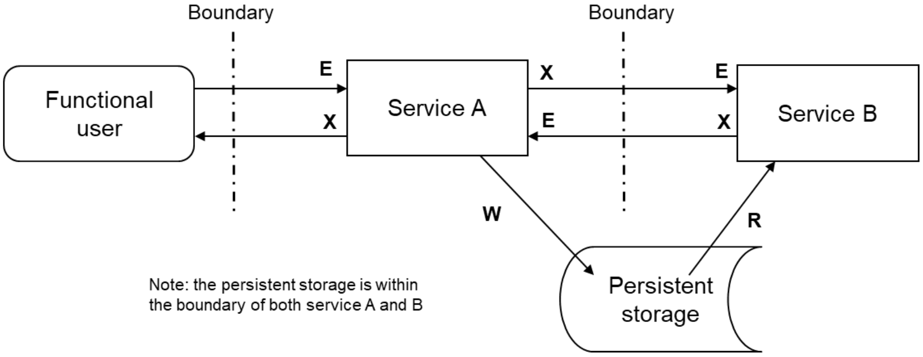
EXAMPLE 2. A data logging utility service (for recovery purposes) may be provided as part of the SOA for any application service that is set up to use it. An application service can be set up to use logging but not vice versa. So there is a hierarchical relationship between them. An application service and the data logging utility service that is used by the application will be in different layers.

EXAMPLE 3. A PDF (Portable Document Format) utility service generates and returns a PDF-file, for a given Word document. The PDF service can operate without needing the services of any specific application, but not vice versa. Therefore, there is a hierarchical relationship between them. The PDF service is therefore in another layer to the application.

EXAMPLE 4: A "reservation service" calls a series of tasks implemented as application services, e.g. "book flight", "book hotel", "book airport transport", "change booking date" "cancel reservation" etc. The sequence in which these services are presented is controlled by an orchestration service. The orchestration service must be in a different layer from the application services.

2.2.2 Data movements of data exchanges between components

Figure 2.2 below shows the possible flows of data movements between components in the same layer (where a component may be an application or a service). For simplicity, data about only one OOI is assumed. It shows *direct* and *indirect* exchange of data between components - one or both forms may be involved when services communicate. If components exchange data *directly,* identify Exit and/or Entry data movements, as per the data movements between service A and service B. But *indirect* exchange of data between components means that a service in one component writes data which is subsequently read by a service in another component. In this situation identify a Write data movement in the former component and a Read data movement by the latter.



Note: the persistent storage is within the boundary of both service A and B

Figure 2.2 - Direct and indirect exchange of data between services in components

Figure 2.3 shows the COSMIC models for data movements when an application service A in an upper layer A writes data which is handled by a utility service B in a lower layer B. Service A is a functional user of service B.

The service B in the lower layer writes the data to persistent storage via the operating system (not shown). The service B in this case could be a logging utility which adds, e.g. a date and time stamp, but is not interested in the attributes of the business data to be logged.

The data is finally physically stored by the device driver software in another layer C. Service B is effectively (i.e. ignoring the operating system) a functional user of the device driver in layer C. The device driver software interacts directly with the storage hardware.

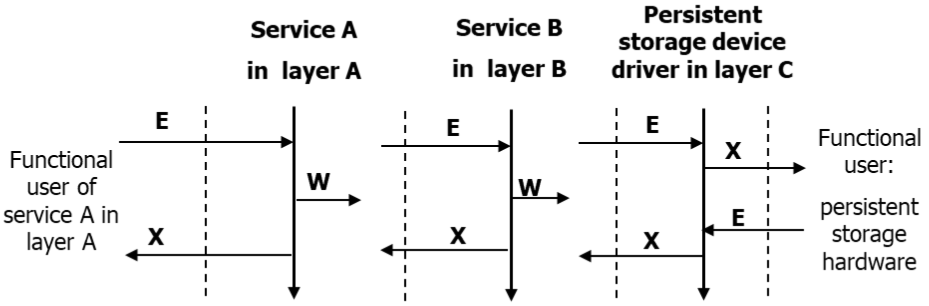

Service A in layer A
Service B in layer B
Persistent storage device driver in layer C
Functional user of service A in layer A
E
W
X
E
W
X
E
X
E
X
Functional user:
persistent storage hardware

Figure 2.3 - Making data persistent between services in a hierarchy of layers

2.3 The functional users of services

Determining the functional users of the software being measured depends on the scope of the measurement, and ultimately on the purpose of the measurement. The functional users must be derived from the FUR of the piece(s) of software being measured. When the purpose requires measurements at the level of individual services, the functional users of services would be identified as follows.

- The functional users of an orchestration service are the application software that calls the orchestration service, and the application services called by and responding to the orchestration service. (The calling ‘application software’ might be a whole application or its component shown in columns a) and b) respectively of Fig. 2.1, i.e. outside the SOA architecture view.)
- The functional users of an application service are the application(s) and any other services that call it, and any intermediary services and utility services that it calls.
- The functional users of an intermediary service are the application services that call the intermediary service and the application service(s) and utility services that is/are called by the intermediary service to fulfill the request and which reply to the intermediary service.
- The functional users of a utility service may be any pieces of software (applications, application services, intermediary services) that call the service.
- When a service A in an upper layer writes data to or reads data from persistent storage which is handled by a service B in a lower layer, then the service A is the functional user of service B (as in Figure 2.3).

EXAMPLE. 1. When services A and B are in the same layer and service A calls service B (without intervention of an intermediary service), service A is a functional user of service B. When service B replies to service A, then service B becomes a functional user of service A.

EXAMPLE 2. When an application invokes one of its own application services the application is a functional user of the invoked service. When the application service replies to the application, then the application service becomes a functional user of the application.

2.4 The level of decomposition of the software

The Guideline for Measurement Strategy Patterns, [7] defines three standard levels of decomposition of software, namely 'Whole application', 'Major component' and 'Minor Component, These correspond to columns a), b) and c) respectively of Figure 2.1. The Minor Component level of this definition applies to any minor component, whether or not it is designed according to SOA conventions. Individual services are therefore 'minor components' on this convention.

2.5 The level of granularity of the measurement

The FUR of business services are often in practice produced at different levels of granularity, ranging from a 'conceptual' level to a 'detailed' level. During the design phase architects will group potentially-reusable functions into services.

Typically in an SOA environment, the specification of FUR for new services involves analysis of existing services that can be re-used, existing services that can be modified, and new services to be developed, as in 'Component-Based Development'. The Measurer may therefore need to apply approximate variants of the COSMIC method [6] as well as the detailed rules for sizing whole services and for changes to existing software services (see section 3.3) at different points in a project to develop SOA services.

2.5.1 FUR at the conceptual level of granularity

The FUR of application services early in the life-cycle of a software development project will probably exist only at the conceptual level. At this level, the FUR usually specify the service names and in the best case a general statement of their functionality in a few sentences.

> *EXAMPLE. Early in a project, the business analysis of an application maintenance project states that the application service 'Register customer' has to be modified and that the new service 'Show customer details' has to be developed. The Measurer can now approximate their sizes with help of one of the approximation variants [6].*

2.5.2 FUR at the detailed level of granularity

Later when the software is being built, there will usually be only an agreed 'specification' and/or 'design document'. The FUR for services at the detailed level of granularity often show a functional design part (the 'what') and a technical design part (the 'how'), with sufficient details to enable a precise COSMIC size measurement. The FUR of utility services are typically produced only at the detailed level of granularity.

> *EXAMPLE. FUR at the detailed level specify the logic of a service, together with a detailed description of its input and output messages.*

3

THE MAPPING AND MEASUREMENT PHASES

As the examples illustrate both the identification of the functional processes and their measurement, both Mapping and Measurement phases are considered in this one chapter.

3.1 Identifying functional processes

Identifying the functional processes of a piece of software to be measured must follow the normal COSMIC process of first identifying the unique event-types in the world of the functional user(s) of the piece of software. Each unique event-type gives rise to one or more functional processes that must respond to the event.

The question is now whether the concepts of 'service' and 'functional process' coincide, i.e. given FUR that describe a service, must the Measurer identify one or multiple functional processes? The reason for this is that there are no standards on how much functionality a service may contain, it is therefore not excluded that one service could lead to multiple functional processes. In this Guideline we assume for simplicity that a service leads to one functional process, as often is the case. The Measurer must however be aware that in practice services may be encountered for which multiple functional processes must be identified.

3.1.1 The impact of communications requirements on functional processes

Developing a service always involves developing the request/reply mechanism for the applications or services that call the service. Non-functional requirements such as for the protocol and timing of communications between components can impact the functionality to be provided and thus the functional size (see the 'Business Application Guideline' [3], section 1.2.3) on non-functional requirements.

In exchanges of messages between components (such as in Figure 1.1 which shows the exchanges between an application A and a service S), the communication may be 'synchronous'. This means that the requesting service waits for the response before it can continue its task.

In 'asynchronous' communications, the requesting functional process does *not* wait for the response message to continue its task. Asynchronous communication is a necessity in situations where the response takes minutes, hours, or weeks to appear, if at all. If there is a requirement for 'asynchronous' communications, this will impact the unique event-types that the software must respond to and hence the functionality and probably also the functional size.

When a functional process A issues a request in asynchronous mode to another service, the arrival of a response message will be another event that triggers a separate functional process B in the requesting software.

> *EXAMPLE 1. A hotel reservation system requires that a credit card ID be given at the time of booking to guarantee the reservation. The credit card ID must be checked for validity with the system of the card supplier. As a response sometimes may take a while (the credit card supplier system might be temporarily unavailable or over-loaded), there is a requirement that this checking is carried out asynchronously, i.e. the 'Make a booking'*

service does not need to wait for a response from the credit card system before accepting the reservation. The model for this set of requirements is shown in Figure 3.1. (Compare this with Figure 1.1 which assumes synchronous communications.)

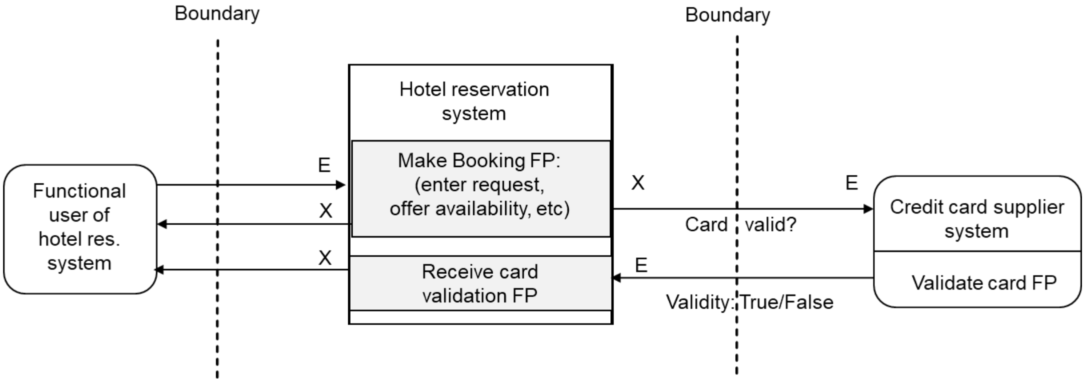

Boundary
Boundary
Hotel reservation system
Make Booking FP: (enter request, offer availability, etc)
Receive card validation FP
Functional user of hotel res. system
E
X
X
X
E
Card valid?
E
Validity: True/False
Credit card supplier system
Validate card FP

Figure 3.1 - Asynchronous communications between a hotel reservation system and a credit card supplier system

Figure 3.1 shows that, because of the requirement for asynchronous communications, the sending of the request for a validity check to the credit card system and the receipt of the return message informing whether the credit card is valid or not must be handled by two separate services (functional processes) of the Hotel Reservation system.

The model shows that the number of data movements of the communications between the two systems is the same, irrespective of whether the communications should take place synchronously (as in Fig. 1.1) or asynchronously (as in Fig. 3.1). However, now that the Hotel Reservation System must have two functional processes (instead of only one if the communications were synchronous), its functionality must be larger. For example, a reservation that was accepted by the first functional process may then need to be cancelled by the second functional process due to a non-acceptance of the credit card. This would not happen with synchronous communication.

EXAMPLE 2. A service may issue a message of the type 'fire and forget', where it does not expect a reply or does not wait for a reply. The functional process of the service that issues the message may continue with other tasks (or may be finished, depending on the FUR) once the message has been issued.

3.1.2 The called functional process(es) of the services

Services most often have the form of simple, single functional processes.However, services may be developed which invoke multiple other services.

An example is a service to control batch-processed services. Such a control service may be an orchestration service invoking batch services that resemble the batch steps in classical batch processing. A batch service is invoked by sending a message with the required data, such as requestor name, password and date/time. See the Business Application Guideline [3] for a detailed discussion of processing in batch mode, section 4.4.3.

EXAMPLE. A controlling service controls a number of batch-processed services each producing one report. The controlling service is triggered each month by a scheduler and starts the report services. Each report service provides feedback to the controlling service whether its report was successfully generated or not.

3.2 Identification of objects of interest, data groups and data movements

Application services represent business functionality, therefore the objects of interest that are referenced in the messages and services are the same as the objects of interest recognized by the applications. The exchanges between any application (or service) and any other application (or service) takes place via Entries and Exits as shown in Figures 1.1 or 3.1, or via Reads and Writes as shown in Figures 2.2 or 2.3, for each object of interest involved in the communication. (However, for fault messages, and the possibility of data being returned about more than one object of interest, see section 3.2.3)

3.2.1 Messages with headers

According to the SOAP protocol, messages contain an optional 'header area' and a mandatory body. The header area may contain more than one header. The body carries the business data.

The sending functional process may add its own header to the data in the message body, and/or intermediate services on the route of the message may add headers, remove them or process data in a header.

> *EXAMPLE; Application or business process management may want to monitor message traffic to or from a service or to monitor service performance or security against an agreed service level. To meet such requirements, a message may be provided with one or more headers. The header data may be processed by the message system, and/or by an intermediate or by the destination service.*

The general form of a message that consists of a header containing data about a single object of interest 'message', followed by the message body is:

> Message Header: (for instance: MessageID, Service_RequestorID, Service_ProviderID, Authentication, Digital Signature, Service_Version, Message_Timestamp, all describing the object of interest 'message').
>
> Message Body: Business data and/or an indication that an error condition has occurred while processing the business data (typically in a return message).

The message body may contain one or more data movements for the business data.

> *EXAMPLE 1: A message with a header which contains routing info, and a body with a file of several multi-item orders for batch processing would consist of three data movements, one for the header, one for the order data and one for order-item data.*
>
> *EXAMPLE 2: A message body may contain invoice data, a summary of outstanding amounts and pricing policy data, each of which gives rise to at least one data movement.*

As headers (if any) can be processed (including 'monitored') in many ways, the Measurer must examine the FUR carefully to decide (a) what functionality is within the scope of the measurement, (b) which functional processes actually handle the headers, in addition to or as well as, the business data, and (c) which data groups should be identified from the header and body (business) data.

> *EXAMPLE 3: Figure 3.2 shows a possible case where a functional process of an application on one processor must communicate business data about a single object of interest with a service on another processor and must expect a reply with data about the object of interest from the service. The security and routing of the communication is enabled by the message system on each processor using data in two message headers which are added to and removed from the business data before and after the transmission respectively, for both the outbound and inbound transmissions.*

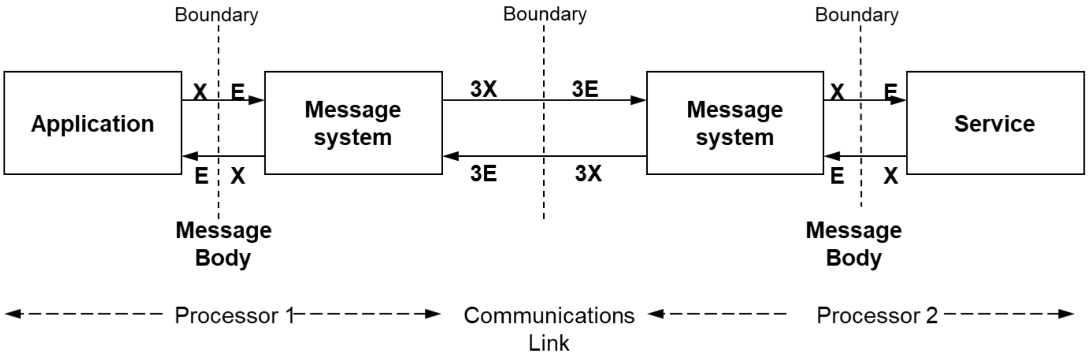


Figure 3.2 Communication between an application and a service on different processors enabled by the message system

The headers describe two objects of interest. The routing information describes the object of interest 'message-route'; the security information describes the message-sender credentials.

As far as the functional processes of the sending application and the receiving service are concerned, they exchange a message consisting of a single data group of business data; they are not involved in the addition and removal of the message headers.

The functional process of the message system on processor 1

- *receives the body message from the application,*
- *adds the two headers and forwards the whole message to its peer message system,*
- *receives the reply, removes the headers and passes the body back to the application,*

This functional process has at least 8 data movements (4 Entries and 4 Exits). It could have further data movements for e.g. the creation of the routing data and for the creation and checking of the security data. The message system on processor 2 has the same size.

This example assumes synchronous communications between the application and the server. If the transmission were asynchronous, the message system would have two functional processes on each processor, one handling the outbound and another handling the inbound transmission separately. The total functional size would be the same as with synchronous communications.

3.2.2 Business-related services

Ignoring any need for headers, two examples are given below of business-related services and one of a utility service.

EXAMPLE 1. A functional process of an application calls one of the application services in the same application in order to make customer data persistent. The call message received by this application service has the form of a triggering Entry moving the customer data. The result of processing (for instance 'OK' or error code) must be issued to the calling functional process. The application service (functional process) has the following data movements:

Data Mvt.	***Object of interest***	***Data Group***
Entry	*Customer*	*Customer data (Incoming message triggering the service)*
Read	*Customer*	*Customer data (exists already?)*
Write	*Customer*	*Customer data (Make customer data persistent)*
Exit	*Result*	*Result of processing to calling functional process*

The size of this application service functional process is 4 CFP

EXAMPLE 2. A customer has obtained a new bank account number, which is registered in the Accounts application's persistent storage. The FUR state that the Agreements application must register the corresponding agreements details for the new account. A functional process that provides a service of the Accounts application calls an Intermediary service which in turn calls a service of the Agreements application to register the agreement details in the Agreements application's persistent storage. The purpose of the measurement is to measure the size of the Intermediary service and of the Agreements application service. (The Example assumes that all services consist of a single functional process and also assumes synchronous communications.)

Figure 3.3 shows the exchanges between the three services.

The service in the Accounts application issues the data to be registered as an Exit which is received as a triggering Entry by the Intermediary service. The latter passes on the data as an Exit, which is received as a triggering Entry by the Agreements application service. This service stores the agreement details and then returns the result of the storage to the Intermediary service which returns the result to the requesting Accounts application service. The 'Result' message is comparable to the final Exit in Example 1 above and confirms the storage or an error.

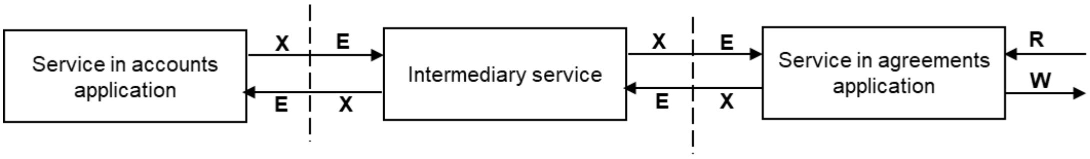

Service in accounts application
X
E
E
X
Intermediary service
X
E
E
X
Service in agreements application
R
W

Figure 3.3 – Application services and Intermediary service

The service of the Agreements application is measured as follows:

Data Mvt.	***Object of interest***	***Data Group***
Entry	*Agreement*	*Account number, agreement data (Request to store agreement details from Intermediary service)*
Read	*Agreement*	*Agreement data (exists already?)*
Write	*Agreement*	*Agreement data (Store agreement details in Agreements application)*
Exit	*Result*	*Result of storage*

The size of the Agreements application service is 4 CFP

The Intermediary service functional process that handles the Agreements application service is measured as follows. It is assumed that the Intermediary service does not need to read or write data, nor to consult other pieces of software:

Data Mvt.	***Object of interest***	***Data Group***
Entry	*Agreement*	*Account number, agreement data (Request to store agreement details from Accounts appl. service)*
Exit	*Agreement*	*Agreement data (Request to store agreement details in Agreement application)*
Entry	*Result*	*Result of storage from service of Agreement application*
Exit	*Result*	*Result of storage to Accounts application service*

The size of the Intermediary service functional process is 4 CFP.

EXAMPLE 3. Business data is logged by a separate utility service in another layer (see service B in layer B of Figure 2.3) via a service, required to return the result of logging with the following data movements:

Data Mvt.	***Object of interest***	***Data Group***
Entry	*Business*	*Data to be logged*
Write	*Business*	*Log data (part of utility software, no result message)*
Exit	*Result*	*Result of processing to requestor*

The size of this utility service functional process is 3 CFP.

3.2.3 Error/confirmation and fault messages

An ‘error/confirmation message’ is defined in the Measurement Manual as:

> “A message issued by a functional process to a human functional user that either confirms only that entered data has been accepted, or only that there is an error in the entered data.
>
> NOTE: If a message to a human functional user provides data in addition to confirming that entered data has been accepted, or that entered data is in error, then this additional data should be identified as a data group moved by an Exit in addition to the error/confirmation message.”

According to the SOAP message protocol there is a standard provision in the body of messages for a ‘fault element’ (normally in the return message of a service request) which, if present, indicates that an error situation has occurred. If there is no error, the fault element is not included in the return message.

If a fault element is present in a return message, it consists of a ‘fault code’, a ‘fault reason’ in human-readable form, and other optional fault-related data. The fault element data attributes describe the same object of interest that would be described in the return message as if no fault had occurred. In other words, if a fault element appears in a return ~~a~~ message, this does not add to the number of data movements.

Measurers should be aware, therefore, that:

- a response from a SOA component containing a fault element is not an error/confirmation message in the sense defined above, as it is not issued directly to a human functional user. This is significant because a response from a SOA component containing a fault element is not subject to rule a) for accounting for error/confirmation messages issued to human functional users – see the Measurement Manual [1], section 3.5.11.
- a response from a SOA component may contain a fault element and 'additional data' (as in the NOTE to the definition of an error/confirmation message). This is significant because the response of one physical message (-type) from a SOA component that includes an Exit with a fault element may include one (or possibly more) additional Exit(s) if the 'additional data' describes one (or more objects of interest that differ from the object of interest described by the fault element.

EXAMPLE 1: A SOA component is required to check for valid postcodes (or 'Zip codes'). If the entered code is valid the response confirms that fact. If the entered code is invalid, the response has a fault element including the reason 'postcode invalid'. Identify one Exit for the response.

EXAMPLE 2: A SOA component is required to provide the postcode for an entered postal address. If the address is recognized, the component returns the valid postcode. If the entered address is not recognized, the return message includes a fault element including the reason 'address not recognized'. Identify one Exit.

EXAMPLE 3: As Example 2, but if the address is only partially recognized, the SOA component includes a fault element including the reason 'address not recognized'. In addition, the response includes a list of possible valid postcodes. Identify one Exit, as all the possible postcodes aim to describe the one entered address, i.e. both the entered message and the response concern the same object of interest.

EXAMPLE 4: A SOA component is required to check if an entered request for a cash withdrawal may be allowed or not. The response indicates that the request is OK or not. Identify one Exit.

EXAMPLE 5: As Example 4, but the SOA component must, if the request is accepted, return the account balance after the cash withdrawal, e.g. 'Withdrawal accepted. Account balance following withdrawal = $749.55'. The response now describes the approval of the withdrawal request and gives the additional data of the account balance. The withdrawal and the account are two separate objects of interest. Identify two Exits.

3.3 Measurement of the size of functional changes to services

The approach to sizing changes provided in the Measurement Manual is fully applicable for changes to services. The measurer may encounter a number of situations

- An existing service is enhanced by adding new features – this change is measured as usual for new functionality.
- An existing service is adapted and implemented again with different behavior – this should be considered as a new service if the different behavior is caused by a different triggering event type.
- A second service is added alongside the current service to accomplish new FUR, and these FUR apply only for specific cases/calls – the second service is considered to be a new service, for the same reason as above.

*EXAMPLE 1. The customer information that is made persistent in section 3.2.2 Example 1 is required to be extended by adding a description of the customer. The message triggering the service is therefore extended with a text field to accommodate the description and the functional process of the service must be changed accordingly. Below the data movements to be changed are indicated by a *-sign (asterisk).*

Data Mvt.	***Object of interest***	***Data Group***
*Entry**	*Customer*	*Customer info (extended message body data)*
Read	*Customer*	*Customer data (Customer exists already?)*
*Write**	*Customer*	*Customer data (Extended customer data made persistent)*
Exit	*Result*	*Result of processing (e.g. 'OK' or fault in data)*

The size of the change is 2 CFP.

EXAMPLE 2. An existing service is implemented again for a different use, with different behaviour, to correspond to a new event type. Assume that the size of the new delivered software service is the same as the existing service. The size of the 'change' will be a count of the number of data movements that must be changed to handle the new event-type. This is an example where re-use must be taken into account in estimating.

4

PERFORMANCE MEASUREMENT AND PROJECT ESTIMATING WITH SOA

4.1 Productivity of developing services

ISBSG Benchmark data show [8] that the productivity of developing minor components such as, individual SOA services is typically ten to twenty times higher than when developing whole business applications. Great care must therefore be taken in activities such as the following:

- When measuring the overall size of an application including services that it uses, the aggregation rules of the Measurement Manual section 4.3.1 must be taken into account. (The total size of the delivered software application as seen by its external functional users must *exclude* all internal exchanges of data movements between the application and its services and between the services themselves.)
- When comparing the productivity of different projects, it is imperative not to mix performance results of projects developing pieces of software at different levels of decomposition.

4.2 Estimating

4.2.1 Sizing the whole, or at component level

A software architect may decompose application software into components when the components will be realized in several development environments.

When approximate application sizes suffice (for instance when the purpose is to use sizes for estimating application management support), it might be sufficient to ignore any decomposition that would make the services visible. The FUR to be sized would define the application as a whole, as seen by its external functional users, and ignoring any internal structure. Without decomposition the software may look like Figure 4.1 (a):

Application
A

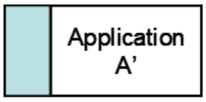


Figure 4.1 (a) Application A 'as a whole'; **Figure 4.1 (b) Application A decomposed**

Alternatively, if the purpose of the measurement is to estimate the effort for developing an application A that consists of an application component and an application services component, then the component structure must be made visible, as shown in Figure 4.1 (b), where the colouring indicates the service part of application A. The overall scope will be divided into a scope for the application component and a scope for the application services component if their productivity factors differ. Separate productivity factors must be applied to the size of each component to obtain the effort to develop the whole application.

4.2.2 Creating and modifying services and estimating effort

A required service may either be newly built, or obtained by modifying an existing service, or functionality of existing applications may be modified to form services.

The process of estimating the effort to modify an existing service is the same as the process of estimating for newly built services, i.e. a productivity for modifying services must be established, which, together with the functional size of the modifications, produces the effort.

If an application provides API's (application-programming interfaces), existing functionality may be modified to convert to services. Also, tools are available to expose the desired functionality as services.

When collecting data for future performance measurement or estimating purposes, it is advisable to distinguish effort and sizes for building new services, modifying existing services, and converting existing software to services.

REFERENCES

All the COSMIC documents listed below, including translations into other languages, can be downloaded from the Knowledge base of www.cosmic-sizing.org.

[1] Measurement Manual.

[2] Erl, T., ‘Principles of Service Design’. Prentice Hall, 2008. ISBN 0-13-234482-3.

[3] Guideline for Sizing Business Application Software.

[4] ISO/IEC/IEEE 42010:2011 Systems and software engineering - Architecture description, 3.2.

[5] Santillo, L., ‘Seizing and Sizing SOA Applications with COSMIC Function Points’, Procs. Software Measurement European Forum (SMEF) 2007, May, 9-11, 2007, Rome, Italy.

[6] Guideline for Early or Rapid COSMIC Functional Size Measurement.

[7] Guideline for Measurement Strategy Patterns.

[8] The performance of real-time, business application and component software projects: an analysis of COSMIC-measured projects in the ISBSG database, published by COSMIC and the International Software Benchmarking Standards Group, September 2009, www.isbsg.org.

GLOSSARY OF SOA TERMS

This glossary contains only terms and abbreviations that are used in this Guideline and that are specific to SOA and services. For terms and definitions of the COSMIC method, please refer to the Measurement Manual [1].

Application service. A piece of application software that offers functionality to other pieces of application software by means of a standard interface.

Enterprise Service Bus (ESB). A central communication infrastructure consisting of a message system and a variety of functionality for connecting applications.

Intermediary service. A service that enables service ‘requestors’ to communicate with service ‘providers’ when some translation of the request and the response is needed.

Message. Data passed between any applications, pieces of software or software services in a standard format. A message may consist of one or more data group types.

Orchestration service. A service that controls (‘orchestrates’) application services to implement a (business) process.

Service. A piece of software that offers functionality to other pieces of software by means of a standard interface.

Service Oriented Architecture (SOA). A software architecture that relies on service orientation as its fundamental design principle.

SOAP (Simple Object Access protocol). A protocol intended for exchanging structured information in a decentralized, distributed environment.

Utility service. A service that provides common functionality, enabling other software to function.

Web service. A service that can be called using Internet standards.

APPENDICES

A.1 ACKNOWLEDGEMENTS

Version 1.1.1 Authors and reviewers (alphabetical order)		
Arlan Lesterhuis* The Netherlands	Bruce Reynolds Tecolote Research United States	

Version 1.1 Authors and reviewers (alphabetical order)		
Dian Baklizky TI Metricas Brazil	Arlan Lesterhuis* The Netherlands	Baris Ozkan Atilim University Turkey
Charles Symons* United Kingdom	Frank Vogelezang Ordina The Netherlands	

* Authors and co-editors of this Guideline

Version 1.0 Authors and reviewers 2010 (alphabetical order)		
Peter Fagg, Pentad Ltd., UK	Samved Galegaonkar, India	Poonam Jain, India
Arlan Lesterhuis*, The Netherlands	Kumaravel Natarajan, India	Marie O’Neill, Software Management Methods, Ireland
Grant Rule, Software Measurement Services Ltd, UK	Luca Santillo*, Agile Metrics, Italy	Charles Symons*, UK
Gerhard Ungerer, TIAA-CREF, USA	Frank Vogelezang, Ordina, The Netherlands	

* Authors and co-editors of this Guideline

A.2 VERSION CONTROL

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

DATE	REVIEWER(S)	Modifications / Additions
April 2010	COSMIC Measurement Practices Committee	First version 1.0 issued
March 2015	COSMIC Measurement Practices Committee	Version 1.1 issued, which corresponds with the Measurement Manual versions 4.0 and 4.0.1
Jan. 2019	COSMIC Measurement Practices Committee	Version 1.1.1 issued, which complies with the Measurement Manual version 4.0.2

A.3 GUIDELINE CHANGE HISTORY

This appendix contains a summary of the principal changes made in the evolution of the COSMIC Service-Oriented Architecture guideline from version 1.0 to version 1.1 and from version 1.1 to the present version 1.1.1.

A ‘MUB’ is a Method Update Bulletin, published between major releases of the Measurement Manual to announce and explain proposed changes.

Changes from version 1.1 to version 1.1.1

V1.1.1 Ref	Change
-	A few editorial improvements added for ease of readability
Foreword	In the 3rd paragraph ‘Function Point Analysis’ replaced by ‘FSM methods’
Foreword	The summary text of Chapter 3 worded more precise
1.1.2	The previous definition of Non-Functional Requirement replaced by the current one
1.1.3	Example 1 removed, being not relevant
2	In the intro a reference to the Measurements Strategy Patterns Guideline added
3.1.1	In Figure 3.1 the correct symbols of ‘functional user’ applied
3.2.2	In Examples 1 and 3 the requirement to send the result of processing added
References	The versions and/or dates of COSMIC documents removed
Glossary	Irrelevant details of the description of the SOAP protocol removed

Changes from version 1.0 to version 1.1

V1.1 Ref	Change
-	Figure numbering is now sequenced per chapter, in conformity with the numbering in other guidelines (e.g. Figure 5 is now Figure 3.2).
-	The ambiguous term ‘middleware’ has been replaced by ‘message system’, where ‘message system’ is meant.
-	Several paragraphs have been moved so as to discuss a topic at one location.
-	Where needed, diagrams have been changed so as to comply with the diagram conventions of the Measurement Manual.
-	Where appropriate. we now distinguish between ‘FUR’ and ‘functional requirements’ or ‘software artefact’.
1.1.2	A new section and example have been added, explaining the concept of NFR and its relevance for functional size.
1.1.3	Example 1 has been re-written to make it clearer.
1.1.4	The text on the Enterprise Service Bus has been extended and a diagram added.
2.2.1	A short explanation of Layers has been added in line with the COSMIC method v4.0.2 and the examples have been changed in order to be consistent with the new definition of Layer.

2.4	A section on the level of decomposition has been added.
3.2.1	In SOA message ‘footers’ are not used, so this term has been removed.
3.2.1	Example 3 and its Figure 3.2 have been changed to be illustrate how message headers may be added and removed and to show the correct number of data movements (Entries and Exits) between the various pieces of software, including both headers and the body of each message.
3.2.3	A section has been added on error/confirmation and fault messages. The rules for accounting for fault elements in messages issued by SOA components differ from the rules for error/confirmation messages issued to human functional users.
References	This section has been updated.
Glossary	A clearer definition of ‘intermediary service’ is given.

A.4 CHANGE REQUESTS, COMMENTS, QUESTIONS

Where the reader believes there is a defect in the text, a need for clarification, or that some text needs enhancing, please send an email to: mpc-chair@cosmic-sizing.org

You can use the forum on cosmic-sizing.org/forums to post your questions and receive answers from our world-wide community. The quality of any answers will depend on the knowledge and experience of the community member that writes the answer; the MPC cannot guarantee the correctness. 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.