



The COSMIC Functional Size Measurement Method

Version 3.0.1

**Guideline for Sizing
Service-Oriented Architecture Software**

VERSION 1.0a

April 2010

ACKNOWLEDGEMENTS

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 Voegelzang, Sogeti, The Netherlands	

* Authors and co-editors of this Guideline

Copyright 2010. All Rights Reserved. The Common Software Measurement International Consortium (COSMIC). Permission to copy all or part of this material is granted provided that the copies are not made or distributed for commercial advantage and that the title of the publication, its version number, and its date are cited and notice is given that copying is by permission of the Common Software Measurement International Consortium (COSMIC). To copy otherwise requires specific permission.

A public domain version of the COSMIC documentation and other technical reports, including translations into other languages can be found on the Web at www.cosmicon.com.

VERSION CONTROL

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

DATE	REVIEWER(S)	Modifications / Additions
7-4-2010	COSMIC Measurement Practices Committee	First version 1.0 issued
16-4-2010	A. Lesterhuis, C Symons	Version 1.0a issued with minor editorial corrections, mainly : Fig. 2.2 and associated text made clearer Text accompanying Fig. 3.1 made clearer In section 3.2, Figs. 3.1 and 3.2, are re-numbered as 3.2.1 and 3.2.2 respectively

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 v3.0.1 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') Function Point Analysis 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, peer 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 [8], 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 3.0.1 [1], and with the Business Applications Guideline version 1.1 [2]. 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 a SOA architecture is being widely adopted in the domain of business application software. All examples in this Guideline relate to that domain. Some examples of middleware functionality are also given, where 'middleware' may be loosely defined for this Guideline as 'utility software that supports SOA software, e.g. 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 relies on SOA or any other structure. This Guideline is only concerned with sizing the various types of components of an SOA architecture.

Introduction to the contents of the Guideline

This Guideline focuses on examples that illustrate the principles and rules of the Measurement Manual and that help interpret them when sizing services. For definitions of the terms of the COSMIC method in general, please refer to the glossary in the Documentation Overview and Glossary [4]. Terms specific for the Service Oriented Architecture are defined in the glossary at the end of this Guideline.

We note that literature on information technology uses a number of terms that are used 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 Functional User Requirements (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 four key parameters of the measurement that must be addressed, namely the purpose and scope of the measurement, the identification of functional users 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, functional processes, objects of interest and data groups. In the next Measurement phase, the individual data movements of the functional processes must be identified and counted, as the basis for measuring 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.....	7
1.1	The Service Oriented Architecture	7
1.1.1	<i>Services</i>	7
1.1.2	<i>Service oriented architectures in practice</i>	8
1.1.3	<i>SOA, business processes and the Enterprise Service Bus</i>	9
1.2	Characteristics of services in the Service Oriented Architecture	9
1.2.1	<i>Orchestration services</i>	10
1.2.2	<i>Application services</i>	11
1.2.3	<i>Intermediary services</i>	11
1.2.4	<i>Utility services</i>	12
2	THE MEASUREMENT STRATEGY PHASE.....	13
2.1	The purpose of the measurement	13
2.2	The scope of the measurement	13
2.2.1	<i>Layers and components</i>	13
2.2.2	<i>Data movements of data exchanges between components</i>	15
2.3	The functional users of services.....	16
2.4	The level of granularity of the measurement.....	16
2.4.1	<i>FUR at the conceptual level of granularity</i>	16
2.4.2	<i>FUR at the detailed level of granularity</i>	17
3	THE MAPPING AND MEASUREMENT PHASES	18
3.1	Identifying functional processes	18
3.1.1	<i>The impact of communications requirements on functional processes</i>	18
3.1.2	<i>The called functional process(es) of the services</i>	19
3.2	Identification of objects of interest, data groups and data movements.....	20
3.2.1	<i>Messages with headers and footers</i>	20
3.2.2	<i>Business-related services</i>	21
3.3	Measurement of the size of functional changes to services	23
4	PERFORMANCE MEASUREMENT AND PROJECT ESTIMATING WITH SOA SOFTWARE.	24
4.1	Productivity of developing services	24
4.2	Estimating.....	24
4.2.1	<i>Sizing the whole, or at component level</i>	24
4.2.2	<i>Creating and modifying services and estimating effort</i>	25
	REFERENCES	26
	APPENDIX A - COSMIC CHANGE REQUEST AND COMMENT PROCEDURE	27
	GLOSSARY	28

SOA AND THE COSMIC GENERIC SOFTWARE MODEL

1.1 The Service Oriented Architecture

1.1.1 Services

A 'software architecture' has been defined as 'the structure or structures of a program or system, which comprise software components, the 'externally visible' properties of those components, and the relationships among them' [6]. In fact, 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' [7]. The software 'capabilities' referred to above will be called 'services' in this Guideline¹.

In SOA, any application requiring commonly-used information from another application sends a request to the service of the application that can handle the request. Requests and replies are often implemented as standardized 'messages' (which term will be used in this Guideline to indicate data in standardized format that is exchanged between software services). Upon receipt of a request message the service processes it, i.e. obtains and/or computes the required information and returns the results to the requesting software in the form of a reply message. An application may call its own services; such calls are also called '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.

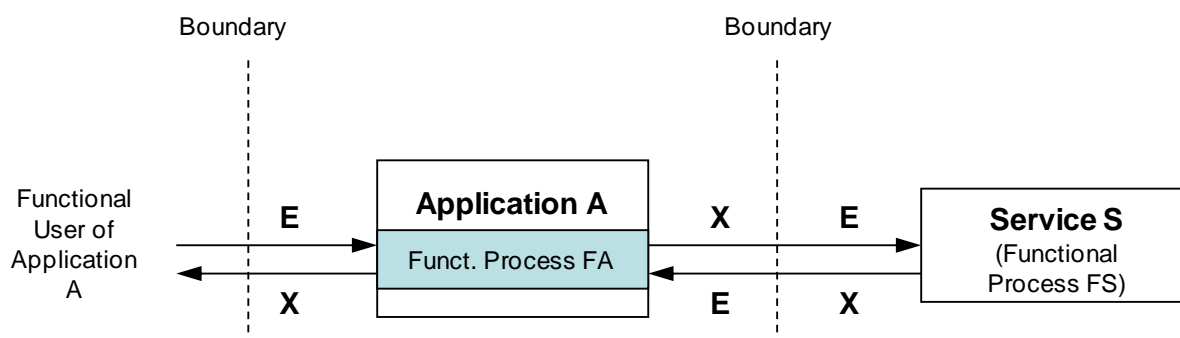


Figure 1.1 – The interactions between an application and a service

¹ Sometimes the interface to, rather than the business logic proper, is also called a 'webservice'. (see for instance [11])

An application A requires some data from a service S. 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 (which we assume consists of a single functional process FS) as a triggering Entry. The functional process FS replies to FA with a message containing the requested data or an error message. 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). Application A is a functional user of the service S, and vice versa.

Note that the service S could be independent, or part of another 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) or from persistent storage (requiring a Read) or from another application or service (needing another Exit/Entry pair), or by a combination of any of these three possibilities.

A message may contain data about one or more objects of interest. Therefore, as per the standard COSMIC method, one Exit/Entry pair (or one Entry/Exit pair) must be identified for each object of interest about which data is conveyed in any message – see more in section 3.2. In SOA, standards are defined that govern what data is to be exchanged in each message type and how it should be formatted, etc.

In SOA terminology, the requesting functional process FA is sometimes referred to as the 'pull' service and the supplying functional process as the 'push' service. Often, if the supplying software is a single service, it will consist of a single functional process.

The Functional User Requirements (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 calling message.

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

Services are 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. The request and reply messages between applications and services are commonly physically handled by means of 'middleware' (see more in section 3.2.1). 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 service), and without the calling software having or needing knowledge of how the service actually performs its tasks or of the support from any middleware or any other enabling software.

1.1.2 *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 (see Example 2). This Guideline thus intends 'to show the wood by means of its trees'².

EXAMPLE 1. In this service architecture, a number of services have been developed to perform financial computations (annuities, loan burden, etc.) and standard applicant tests ('has the applicant for the loan any other debts?', 'is the applicant

² Bertrand Russell.

on a black list', etc.). These services are collected in a separate 'Financial Services Component'. The services of this component are called by the applications that need their use, without intermediary services (see Ex. 2). Besides simplifying maintenance (which is required for one service only, rather than for a number of applications each containing the same computation), this architectural solution has the advantage that all applications that have to perform the same computation or apply the same test produce the same result.

Some communications with or between services may take place via 'intermediary' services (not to be confused with 'middleware'). 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 '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 2. 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 these service(s) the application calls an 'intermediary service' which in turn calls the required application service(s). The intermediary service ensures the flexibility referred to above.

1.1.3 SOA, business processes and the Enterprise Service Bus

In SOA, applications and services may mutually interact by sending messages via an 'Enterprise Service Bus' (ESB). An ESB is the central communication infrastructure consisting of middleware and a variety of functionality for connecting applications, especially work flow management (definition and execution of business process flows). In the SOA this is indicated by *orchestration* and enables business processes to be implemented by calling application services in the desired order. The business process flow is now independent of other application functionality. This architecture thus enables business process flows to be changed without changing the underlying applications and services.

1.2 Characteristics of services in the Service Oriented Architecture

In this Guideline services are classified as shown in Table 1.2.1. This classification is one of many possibilities, see for instance the classification in [8], and is introduced simply to illustrate SOA measurement principles. Some alternative commonly-used terms for these same characteristics are also listed. In this Guideline the term 'service' will be used when services in general are meant.

Software services are distinguished by the following characteristics:

- A service offers functionality to its users (normally other software), using a standard interface ('messages').
- Physically, a service may use a synchronous or an asynchronous communication method. A synchronous service fulfils its task(s) and notifies the result to the requestor immediately, i.e. the requestor waits for this result. An asynchronous service fulfils its task(s) and does not notify the result to the requestor immediately, i.e. the requestor does not wait for the result or doesn't get a result at all ('fire and forget'). The COSMIC method can model the FUR of services for both types of communications without needing to consider the physical method used.
- Many services are present in one layer, as defined in the COSMIC method, namely the 'application layer' ('business layer' in SOA jargon) and are peer pieces of software (see more in Chapter 2). General 'utility services', perhaps provided by externally supplied 'middleware' may also be present. Depending on their design, some of the utility services are present in the application layer, some may be in a lower 'utility layer'. For more details, see section 2.2.1.
- 'Web services' are services that can be called using internet protocol standards. Therefore, web services enable coupling of applications via the world-wide web infrastructure. Services of these

external 'web-enabled applications' (which can be called via a www address) may thus be called in the same way as local application services.

Term used	Alternative terms	Description
Orchestration service	Process service, task service	Controls ('orchestrates') application services to implement a (business) process
Application service	Business service, entity service	Provides business functionality of an application
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

Table 1.2.1 Terms used in this Guideline and alternative industry terms.

1.2.1 Orchestration services

Orchestration services call and control other services when the latter are needed to implement a (business) process. An orchestration service thus coordinates or conducts an 'orchestra' of services, and hence its name.

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 1.2. 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. It is an orchestration service of the mortgage application process that controls the sequence of the message calls.

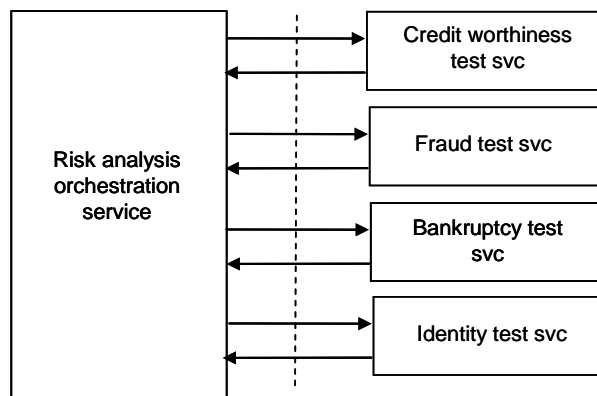


Figure 1.2 – Risk analysis orchestration service

However, although almost all organisations have business processes supported by software, often in practice an organisation that adopted SOA will not have implemented its software support as orchestration services. The reason is 'legacy', i.e. in practice processes will be already supported by specific applications which, besides their specific functionality, also include the needed workflow management. Processes may also have been implemented by dedicated work flow application software. In such cases, therefore, orchestration services are absent.

1.2.2 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. When defined as a service, the functionality of the application can be easily 're-used', for instance via orchestration services, and needs therefore to be developed only once.

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.

1.2.3 Intermediary services

As discussed in Example 2 of section 1.1.2, 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. 1.3

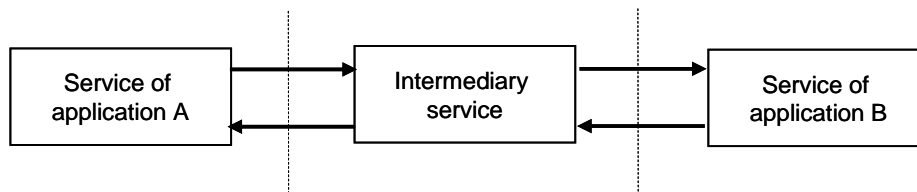


Figure 1.3 – Application services and interconnecting intermediary service

When a functional process of an application service in application A requires data that is available via a service in application B, the former application service calls a functional process of 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:

- Control the handling of the request received from (a service of) application A.
- 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.
- Call a functional process of the application service of application B by means of the translated message.
- Receive the reply message from the functional process of application B (and possibly reply messages from applications C, D, ...).
- Translate the results into a message in the language of application A.
- Send the reply message to (the functional process of the service of) application A.
- Finalize exception situations.
- Log data about 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 another application that processes the payment transactions. When a bank customer cancels his/her account, the 'Cancel account' function (application service) calls the intermediary service, which calls the application service of the payment transactions application that ends the account. Also, the intermediary service calls the application service of the agreements application which ends 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. As will be shown in the next chapter, it depends on its design whether a utility service resides in the application layer or in a lower 'utility layer'.

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 3rd line support (maintenance).

THE MEASUREMENT STRATEGY PHASE

The Measurement Strategy considers four key parameters of the measurement that must be addressed, namely the purpose of the measurement, its scope, the identification of the functional users and the level of granularity of the FUR.

2.1 The purpose of the measurement

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

- a) Comparing the estimated realization cost with the purchase price, when the choice is between buying or making services ('make-or-buy').
- b) Using the size of the services portfolio (i.e. the total of sizes of all services) for estimating application management support.
- c) Using the size of the services portfolio for pricing the outsourced development and maintenance of the application services portfolio.
- d) Benchmarking, i.e. comparing the productivity of the services development department of an organization with the productivity of other comparable organizations.

EXAMPLE. The purpose is to measure the size of a set of services realized in all projects in some year, for all four purposes. It might be sufficiently accurate to measure sizes using an approximation size variant of the COSMIC method to speed up this task [3]. For instance, in an organization it may have been observed that 8 CFP is an acceptable working estimate for an average 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 productivity.

2.2 The scope of the measurement

The overall scope of a measurement consists of the FUR of all pieces of software to be measured. This overall scope may have to be divided into a number of partial scopes, each of which must be measured separately, depending on the purpose. Examples would be when the system to be developed consists of different kinds of software, with different development environments, different operating platforms, with different expected productivities. There would then be no point in adding the sizes of these different types of software. When the software is developed over different layers or components, the estimator must also take into account that different productivities may apply.

EXAMPLE. If the purpose of the measurement is to support estimating and/or to determine the software project productivity (or other performance parameter), and the software to be measured consists of some application services of one application, then the scope will be the set of these services. If the software to be measured consists of application services of different applications, then separating the overall scope into a distinct scope per application may be considered.

2.2.1 Layers and components

Two pieces of software that exchange data will be in different layers if they satisfy all four Principles for distinguishing layers – see the Measurement Manual v3.0.1, section 2.2.4. For our purposes, the two most important of these principles are

- Hierarchy Principle: there is a hierarchical relationship between the two pieces (i.e. the software in the sending layer X relies on the services of software in the receiving layer Y to function, but the software in Y can function without the services of software in X) and
- Interpretation Principle: the data that is exchanged between software in the two layers X and Y is defined and interpreted differently in the respective FUR of the two pieces of software - although there must also exist one or more commonly defined data attributes or sub-groups to enable the software in the receiving layer to interpret the request that has been passed by the software in the sending layer.

A service may be designed:

- to be part of a specific application. When the application software has been decomposed the service is in one of the application's components, or sub-components, etc, all in the application layer.
- independently of a specific application. In this case the service may be in the application layer or in a lower layer depending on the two principles cited above.

In this Guideline four types of services are distinguished: orchestration services, application services, intermediary services and utility services. Business functionality resides in the application layer. In which layer do the services reside? From the above and the description of the types of services in section 1.2 it follows that caution is needed to allocate a service to a layer: one service may be in the application layer, whereas another service providing effectively the same functionality may be in a lower layer.

Generally speaking application services are normally designed as part of an application and are therefore in the application layer. The same holds for the intermediary services because they use services of another application and must therefore be in the same layer as the application services.

Orchestration services, if present, and when designed as part of an application would be in the application layer. Even if designed independently of any particular application, orchestration services are still in the application layer, because the data interchanged between the orchestration service and its application services will be business data interpreted identically by both the orchestration service and its application services.

However, as a service may be in the same layer as the 'calling' application or service, or in a lower layer, the measurer must check whether or not both the Hierarchy Principle and the Interpretation Principle are satisfied, if not they are peer pieces of software. 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. 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 and the parsing service interpret the exchanged data identically, so they must be peers, in the same application layer.

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. The logging service can operate without needing the service of any specific application, but not vice versa – once an application service is set up to use logging, it cannot execute without the logging service. So there is a hierarchical relationship between them. It then depends on the utility's FUR whether the utility is in the application layer or in a lower layer. If the data attributes to be logged must be interpreted identically by both the application and the utility, then the utility is in the application layer.

Alternatively, if the utility does not need to recognize the attributes to be logged the utility is in a lower (utility) layer.

EXAMPLE 3. A general 'logon and access' utility service authenticates users and determines which human users can access which applications, and which users can access which transactions within each application. Data exchanged between the 'logon and access' service and each application is viewed identically. Therefore applications and the utility service must be regarded as in the same application layer.

EXAMPLE 4. 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. However, the calling function specifies the ID's and locations of the source and destination documents to the PDF service and it also provides the content of the document to be converted. These must obviously be defined identically in the FUR of the PDF service. The PDF service is therefore in the same layer to the application, the application layer.

2.2.2 Data movements of data exchanges between components

Figure 2.1 below shows the possible flows of data movements between components in the same layer, i.e. between peer components (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 (per object of interest).

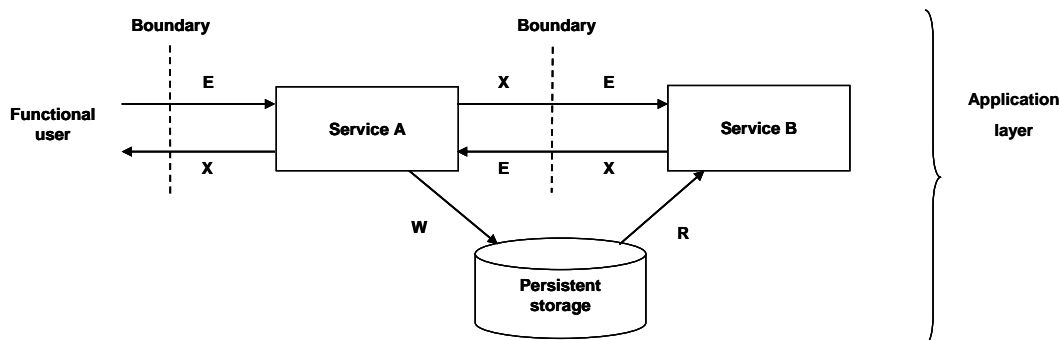


Figure 2.1 Direct and indirect exchange of data between services in peer components

Figure 2.2 shows the COSMIC models for data movements when an application service A must write data which is handled by a logging service L in a lower layer. The latter adds, e.g. a date and time stamp, but is not interested in the attributes of the business data to be logged. The logging service receives the data to be written as an Entry, writes the data to persistent storage and reports back the success or otherwise as an Exit. (The probable role of the operating system and a device driver is not shown.) This diagram corresponds to the 'alternative' of Example 2 in section 2.2.1.

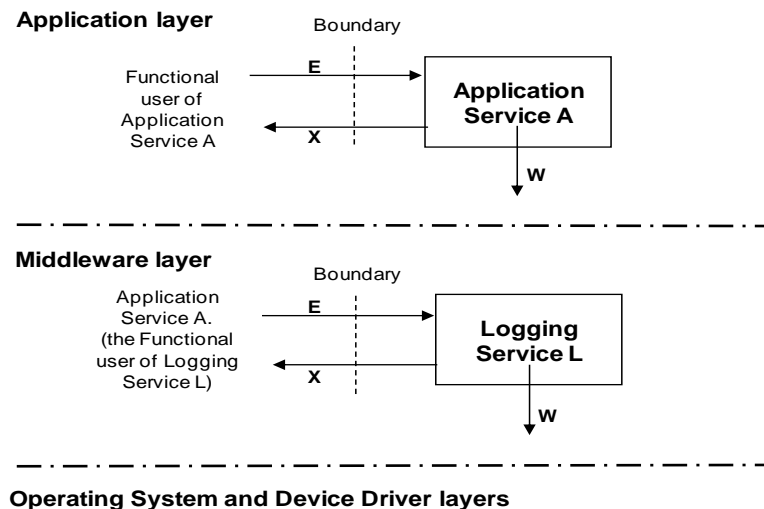


Figure 2.2 Making data persistent between services in a hierarchy of layers

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 functional users of an application service are the application to which it belongs and with which it interacts; any intermediary service and utility service (as far as they are in the application layer) that it calls; and any other service or application that calls the application service.
- The functional users of an intermediary service are the application services that call the intermediary service and the application service(s) and utility services (as far as they are in the application layer) 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 in the application layer may be any pieces of software (applications, application services, intermediary services) that call the service.
- When a service U in an upper layer Writes data to or Reads data from persistent storage which is handled by a service L in a lower layer, then the service U is the functional user of L (as in Fig. 2.2).

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 granularity of the measurement

When analyzing a service contract, a measurer must first separate the FUR from the non-functional requirements. Next, as requirements may be at varying levels of granularity, the measurer must check the level or levels at which the FUR exist, as the development project proceeds.

The FUR of business services are often in practice produced at several levels of granularity, ranging from a 'conceptual' level to a 'detailed' level. The former level is used for early measurement which most often is required for high-level estimates. During the design phase architects will group reusable functions into services. When designs are complete and at the detailed level all the information is available for precise functional size measurements.

2.4.1 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. If an estimate of functional size is needed, it is possible to use approximation variants of the COSMIC method. See [3] for details.

EXAMPLE. Early in the 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.

2.4.2 *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 detailed COSMIC 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.

THE MAPPING AND 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 to events and functional processes, to objects of interest and to data groups. In the Measurement Phase, knowing the functional processes and data groups, the individual data movements of the functional processes can be identified, as the basis for measuring functional size. As the examples treat both the functional processes and data movements together, both Mapping and Measurement phases are considered in this one chapter.

3.1 Identifying functional processes

Identifying the functional processes of a component 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 component. Each unique event-type gives rise to one³ functional process that must respond to the event.

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' [2], section 1.2.3).

All the models of the exchanges between components shown in this Guideline up till now, such as in Figure 1.1 which shows the exchanges between an Application A and a service S, assume that the communications are 'synchronous'. This means that the requesting (or 'pull') functional process FA waits for the response from the ('push') functional process FS of the service S before it can continue its task of responding to its triggering event.

If, however, there is a non-functional requirement for 'asynchronous' communications, this will impact the unique event-types that the software must respond to and hence the functional model and probably also the functional size. In asynchronous communications, the requesting functional process does NOT have to wait for the response before it can continue its task. The response (if any) will be an event that triggers another functional process in the component that made the original request.

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. There is a (non-functional) requirement that this checking is carried out asynchronously, i.e. the 'Make a booking' functional process does not need to wait for a response from the credit card system before accepting the reservation (because the credit card supplier system might be temporarily unavailable or over-loaded.). The model for this set of requirements is shown in Figure 3.1. (Compare this with Fig. 1.1 which assumes synchronous communications.)

³ In principle an event can give rise to one or more functional processes. In exchanges between SOA components of business application software, an event usually gives rise to a single functional process

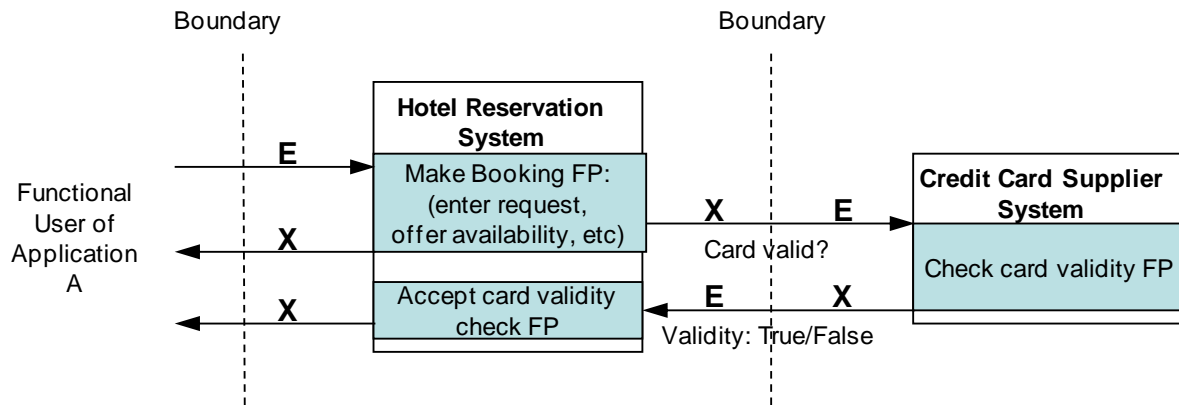


Fig. 3.1 Asynchronous communications between a Hotel Reservation System and a Credit Card Supplier System

Figure 3.1 shows that, due to the requirement for asynchronous communications, the 'Make Booking' functional process continues its processing after it has sent a validity check request to the credit card system without waiting for a reply. Meanwhile, the credit card system makes its check and replies to the reservation system informing whether the credit card is valid or not. The creation of the reply message represents an event generated by the credit card system (which is a functional user of the reservation system). The arrival of the reply message is a triggering Entry for a separate 'Accept card validity check' functional process of the reservation system.

The model shows that the number of data movements of the communications between the two systems is the same, and hence the functional size of the communications is the same, irrespective of whether the communications should take place synchronously or asynchronously. 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 to handle the passing of data between different functional processes – this aspect is not examined further in this Example.

EXAMPLE 2. If a service issues a message of the type 'fire and forget', where it does not expect a reply or does not wait for a reply, this is an asynchronous communication. 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, in the business software domain, services may also be developed which involve multiple functional processes.

An example is a service to support batch processing software. Such a service may consist of a number of processing steps (maybe under the control of orchestration services) that resemble the batch steps in classical batch processing. A batch processing service is invoked by sending a message with the required data, such as requestor name and password. To identify the functional process(es) of a batch service, first identify functional users (as in section 2.3) and then the separate events in the world of these users that the batch service must respond to. See the Business Application Guideline [2], section 4.1 and, for a more detailed discussion of processing in batch mode, section 4.4.3.

EXAMPLE. A services component contains a batch service that controls a number of application services each producing one report. The batch service is triggered each month by a scheduler and starts the report services. Each report service provides feedback to the calling batch service whether its report was successfully generated or not. Depending on the purpose of the measurement, the batch service may be analyzed as a single functional process or as a number of separate functional processes. For a discussion of whether each report service should be identified as a separate functional process, or not, see the Business Application Guideline reference above.

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 services are the same as the objects of interest that are of interest to the applications. The exchange between any application (or service) and any other application (or service) is made via one 'Exit/Entry pair for each object of interest involved in the communication.

The reply to a request for data from a service is normally a single data movement for each object of interest containing either the requested data or an error message, i.e. there is no error message to the requestor separate from the reply in such an exchange. (Exceptionally, besides the normal/error reply to the requestor, the called service may be required to issue a separate error message to some application management software, to inform it that an error situation occurred. In that case an additional Exit from the service-provider software is identified for this error message.)

An important aspect of services is identifying the Entry data movements in the input message(s) and the Exit data movements in the output message(s), which may concern more than one object of interest.

3.2.1 Messages with headers and footers

Application or production management may require to monitor message traffic to or from a service or to monitor the performance of services against a service level agreement. To meet these requirements data about messages must be processed. For this purpose messages may be provided with one or more headers⁴, and sometimes also footers, to contain this 'system' data, in addition to the business data.(referred to in IBM jargon as the 'payload').

A message header and/or footer contain data about an object of interest that may be called 'message'. The general form of a message with a header and footer is then:

Message Header: For instance: MessageID, ServiceRequestorID, ServiceProviderID, ServiceVersion, MessageTimestamp

Message Body: Business data⁵

Message Footer: Other message related data, for instance: End of Message, Count of business data records (if multiple occurrences)

Such a message contains one data movement for the header and footer data, and one or more data movements for the business data.

EXAMPLE A message with a header and footer, containing a file of several multi-item orders for batch processing would consist of three data movements, one for the header/footer, one for the order header and one for order-items.

As headers and footers (if any) can be added, processed (including 'monitored') and removed in many ways, the measurer must examine the FUR carefully to decide (a) what functionality is within the scope of the measurement and (b) which functional processes actually handle the headers, in addition to or as well as, the business data.

On question (a), SOA architects generally do not advise to have a service process both the header data and the business data, Therefore in some implementations, headers are handled by 'middleware'. The measurer must then decide whether the middleware is within the scope of the

⁴ According to the SOAP protocol, it is possible for a message to have two or more headers. The measurer must determine if these different headers contain data that describes different objects of interest (and therefore more than one data movement must be identified) or whether the headers all contain data that describes the same object of interest (so only one data movement must be identified).

⁵ Maybe multiple occurrences of the business data if the message contains a file for batch processing.

software to be developed and measured or (for example if it is pre-existing and standard for all services) whether it need not be measured.

The functionality of middleware which includes header-handling is a peer component of the services components⁶ of the applications involved.

On question (b), if header-handling functionality is within the measurement scope, the measurer must decide which functional processes in which applications, services or middleware create, process and remove (or do not pass on) the headers and footers.

EXAMPLE Figure 3.2.1 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. The communication is enabled by middleware on each processor.

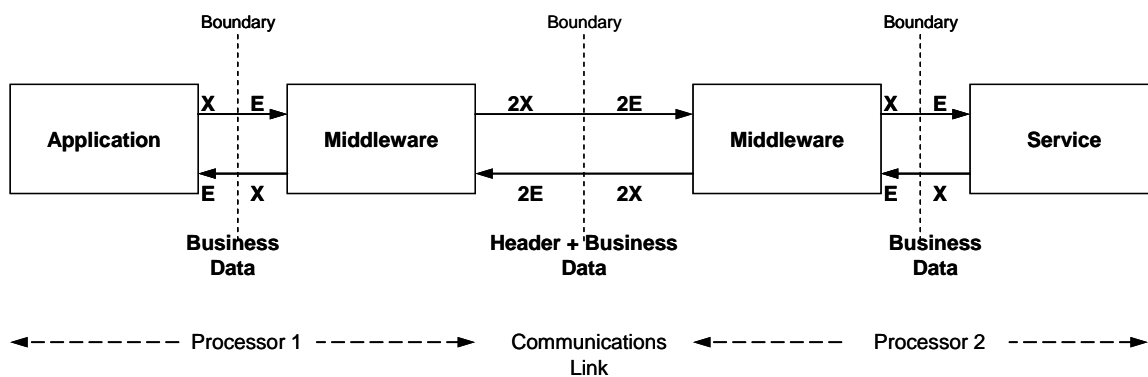


Figure 3.2.1 Communication between an application and a service on different processors enabled by middleware

As far as the functional processes of the application and the service are concerned, they exchange a single data group of business data. However, this data group is handled by the middleware on processor 1, which adds a header. The middleware then transmits a message consisting of two data movements (header plus business data) to processor 2 where it is received by the middleware which processes the header. The middleware on processor 2 sends only the business data to the service. The return reply of business data from the service to the application is handled similarly by the middleware adding and removing a header during the return.

The functional process of the middleware on processor 1 that receives the message from the application, adds a header, forwards it to its peer middleware, receives the reply and passes it back to the application has at least 6 data movements (3 Entries and 3 Exits). It may have further data movements for e.g. logging. Similarly for the middleware on processor 2.

3.2.2 Business-related services

Ignoring any need for, and processing of, a header, 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 application service (functional process) has the following data movements:

E	Customer data	Incoming message triggering the service
R	Customer data	Customer exists already?

⁶ Note that other parts of the middleware may reside in a lower layer

W	Customer data	Make customer data persistent
X	Result	Result of processing (for instance 'OK' or error code)

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 dictate that the Agreements application must register the corresponding agreements data. 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.

A functional process of the service in the Accounts application issues the data to be registered as an Exit which is received as a triggering Entry by a functional process of the Intermediary service. The latter passes on the data movement as an Exit, which is received as a triggering Entry by a functional process of the Agreements application service. This functional process stores the agreement details and then returns the result of the storage to a functional process of 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. Fig. 3.2.2 shows the exchanges between the three services.



Figure 3.2.2 – Application services and Intermediary service of Example 2

The functional process of the application service of the Agreements application is measured as follows:

E	Account number, Customer data	Request to store agreement details from intermediary service
R	Agreements	Agreement details already exist?
W	Store account number, Customer data	Store agreement details in Agreements application
X	Result of storage	Result of storage to Intermediary service

The size of the Agreements application service functional process is 4 CFP

The Intermediary service functional process that handles the Agreements application service is measured as follows. It is assumed that it functions synchronously and that there is no need to read or write data, nor to consult other pieces of software:

E	Account number, Customer data	Request to store agreement details from Account appl. service
X	Account number, Customer data	Request to store agreement details in Agreement application
E	Result of storage	Result of storage from service of Agreement application.
X	Result of storage	Result of storage to Accounts application service/Error message

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

EXAMPLE 3 Business data is logged by a separate utility service in a lower layer (see Service B in Layer B of Figure 2.2) via a functional process with the following data movements:

E	Business data	To be logged
W	Log data	Make data persistent (part of utility software, no result message)
X	Result	Result of processing to requestor/error message

The size of this utility service functional process is 3 CFP

Note that when a functional process of any application or service in the application layer must log data, an extra W data movement per object of interest must be identified for this functional process, regardless of whether or not the logging is handled by a separate logging service in a lower layer.

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 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 put beside 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).

E*	Customer info	Extended message body data
R	Customer data	Customer exists already?
W*	Customer data	Extended customer data made persistent
X	Result	Result of processing (for instance 'OK' or error code)

The size of the change is 2 CFP.

EXAMPLE 2. An existing service is implemented again with different behaviour, to correspond to a new event type. The size of the new delivered software service will be the same as the existing service. The size of the 'change' will be only 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.

PERFORMANCE MEASUREMENT AND PROJECT ESTIMATING WITH SOA SOFTWARE

4.1 Productivity of developing services

ISBSG Benchmark data show [9] that the productivity of developing small, individual service components 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

- adding sizes of service components and sizes of pieces of a whole application to get an overall size. (The total size of the delivered software application as seen by its external functional users must exclude all internal exchanges of Exit/Entry pairs - see section 4.1.8 in the MM)
- comparing the productivity of developing service components against the productivity of developing whole business applications. (It is imperative not to mix performance results of projects developing pieces of software at different levels of decomposition.)
- estimating for the development of a mix of conventional application software with new service components.

4.2 Estimating

4.2.1 Sizing the whole, or at component level

In SOA, a software architect will for instance 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 components visible. The FUR of the application to be sized would define the application as a whole, as seen by its external functional users, and ignoring any internal structure. With no decomposition the software may look like Figure. 4.1 (a),



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

When it comes to estimating the effort to develop services and the development effort differs per component, the component structure must be made visible. The FUR of the software at the lower level of decomposition shown in Fig. 4.1 (b), where the colouring indicates the service part of Application A, must then have sufficient detail that the services component and the remainder of the application may be sized separately. The size of the whole application in Fig. 4.1 (a) will be smaller than the sum of the sizes of the two components of the decomposed application in Fig. 4.1 (b), because the latter sum includes the inter-component data movements that are not visible at the level of the whole application. As the numbers of data movements between the two cases differ, it is important to take this into

account when developing and applying estimation models for the application as a whole and as decomposed into separate components.

EXAMPLE. If the purpose of the measurement is to estimate the effort for developing an application as accurately as possible, where the application consists of an application component and an application services component, then the overall scope will be divided into a scope for the application component and a scope for the application services component. Separate productivity factors must be applied to the size of each component to obtain the total effort.

4.2.2 Creating and modifying services and estimating effort

A service may either be newly built or specific functionality of existing applications may be modified to form services. Quantifying (estimating) the effort of changing services is the same as for newly built services, i.e. a productivity for changing services must be established, which, together with the functional size of the changes, produces the effort.

If an application provides API's (application-programming interfaces), 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 functionality, converting existing software to services and modifying existing services.

REFERENCES

REFERENCES

- [1] COSMIC, 'COSMIC FSM Method v3.0.1 – Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2003)', May 2009. URL: <http://www.cosmicon.com/>.
- [2] COSMIC, 'COSMIC FSM Method v3.0 – Guideline for Sizing Business Application Software, v1.1', May 2008. URL: <http://www.cosmicon.com/>.
- [3] COSMIC, 'COSMIC FSM Method v3.0 Advanced & Related Topics', December 2007. URL: <http://www.cosmicon.com/>.
- [4] COSMIC, 'COSMIC FSM Method v3.0.1 – Documentation Overview and Glossary of Terms', May 2009. URL: <http://www.cosmicon.com/>.
- [5] Clemens, P.C., 'Software Architecture Documentation in Practice', SEI Symposium, Pittsburgh, 2000.
- [6] OASIS SOA Reference Model Technical Committee, 'Reference Model for Service Oriented Architecture 1.0, Committee Specification 1', 2 August 2006, Organization for the Advancement of Structured Information Standards. URL: <http://www.oasis-open.org/>.
- [7] Santillo, L., 'Seizing and Sizing SOA Applications with COSMIC Function Points', Procs. Software Measurement European Forum (SMEF) 2007, May, 9-11, 2007, Rome, Italy.
- [8] Erl, T., 'Principles of Service Design'. Prentice Hall, 2008. ISBN 0-13-234482-3.
- [9] 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
- [10] IBM Redbooks on the SOA Foundation, www.redbooks.ibm.com
- [11] Wall, L, Lader, A. 'Building Web Services and .NET Applications', McGraw-Hill, 2002

APPENDIX A - 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@cosmicon.com

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 an error 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.cosmicon.com site.

The decision of the COSMIC MPC on the outcome of a CR review and, if accepted, on which version of the business application Guideline 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 organisations exist that can provide training and consultancy or tool support for the method. Please consult the www.cosmicon.com web-site for further detail.

GLOSSARY

This glossary contains only terms and abbreviations that are used in this Guideline and that are specific to SOA and services. For most terms and definitions of the COSMIC method, please refer to the COSMIC Documentation Overview and Glossary (References [4]).

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 middleware and a variety of functionality for connecting applications.

Intermediary service

A service that passes messages between two pieces of software or software service(s).

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 service is 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.

Utility service

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

Web service

An application service that can be called using Internet standards.