Using COSMIC for the Functional Size Measurement of Distributed Applications in Cloud Environments

Chapter · April 2017

DOI: 10.1007/978-3-319-54325-3_3

3 authors:

Filomena Ferrucci
Università degli Studi di Salerno
171 PUBLICATIONS **1,690** CITATIONS
SEE PROFILE

Carmine Gravino
Università degli Studi di Salerno
150 PUBLICATIONS **1,924** CITATIONS
SEE PROFILE

Pasquale Salza
University of Zurich
32 PUBLICATIONS **364** CITATIONS
SEE PROFILE

Chapter 3
Using COSMIC for the Functional Size Measurement of Distributed Applications in Cloud Environments

Filomena Ferrucci, Carmine Gravino, and Pasquale Salza

3.1 Introduction

Software sizing is a crucial management activity since it supports several other software project management tasks, such as effort/cost estimation, project planning, productivity benchmarking, and quality control. It is widely recognised that the competitiveness of software companies greatly depends on the ability of their project managers to carry out a reliable and accurate software size estimation. To this aim, functional size measurement (FSM) methods have been extensively investigated in software engineering research and are also widely applied in industry. The success of those approaches is mainly due to the fact that sizing is based on the functionality provided to the users, i.e., the Functional User Requirements (FURs), rather than on other software artefacts (e.g., code) that are not available in the early phases of software life cycle when size estimation is especially important [1].

The Function Point Analysis (FPA) was the first FSM method to be introduced in 1979 [2]. Since then, several variants have been proposed (known as first-generation FSM methods) to improve the size measurement or extend its application domain. COSMIC [3] is a second-generation FSM method, being the first to comply to the standard ISO/IEC14143/1 [1]. It is based on fundamental principles of software engineering and measurement theory.

The COSMIC size is essentially obtained by counting the data movements Entry, Exit, Read, and Write [3]. Although the method was conceived for business, real-time, and infrastructure software (or hybrids of these), COSMIC turns out to be applicable to other kinds of software systems (except for those characterised by complex algorithms) provided that suitable guidelines are provided to ensure that

F. Ferrucci (✉) • C. Gravino • P. Salza
University of Salerno, Fisciano, Italy
e-mail: fferrucci@unisa.it; gravino@unisa.it; psalza@unisa.it

Z. Mahmood (ed.), *Software Project Management for Distributed Computing*,
Computer Communications and Networks, DOI 10.1007/978-3-319-54325-3_3

the specific characteristics of those kinds of software are appropriately captured by the measurer. As a matter of fact, recent empirical studies have been conducted in order to verify the capability of COSMIC size to predict the effort needed to develop web applications. Di Martino et al. [4] reported a better predictive capability of COSMIC against first FSM methods, namely, FPA. Moreover, recent studies have investigated the applicability of COSMIC to mobile applications [5, 6]. This domain is rapidly growing, and new software engineering processes, including functional size measurement and estimation methods [7], are required to improve the quality of these applications. Similarly, the explosive growth of cloud computing requires suitable software size measurement approaches able to support project managers in planning, management and control of software suitable for distributed environments.

In this chapter, we analyse various aspects of the use of the COSMIC method to measure distributed applications in cloud environments. The analysis considers the three distinct provision models of the cloud computing stack: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Furthermore, it deals with specific concepts of cloud computing such as orchestrations and load balancing of several components that act together to realise the required functionality and to ensure critical non-functional requirements (e.g., scalability, reliability).

The rest of the paper is organised as follows. We first give an overview of the FSM and COSMIC methods in Section 3.2. In Sect. 3.3, we analyse the different approaches and guidelines present in the literature about functional size measurement of distributed applications in cloud environments. Section 3.4 contains some final remarks and future work.

3.2 Software Size and COSMIC Measurement

COSMIC is a second-generation function size measurement (FSM) method, and it has some main characteristics in contrast to traditional function point methods. In this section, we give an overview of the history of COSMIC with its main features and usage.

3.2.1 FSM Methods

The measures proposed in the literature to size software can be grouped into two main families: the functional and dimensional ones. A functional size measure is defined as ‘a size of software derived by quantifying the Functional User Requirements (FURs)’ [1]. Thus, measures obtained by applying FSMs are particularly suitable for the early phases of the software development process, when only FURs are available. They can be then exploited for tasks such as estimating a project development effort. Moreover, they can be employed for performing comparisons among projects developed with different platforms, solutions and so on, since they are independent of the adopted technologies. Differently, dimensional size measures basically allow counting structural properties of a software artefact, such as LOCs, the number of web pages and so on. As a consequence, they are strongly dependent on the adopted technological solutions, they can be employed only after the artefact has been developed and often a standard counting procedure is missing [8, 9].

Function Point Analysis (FPA) is considered the first FSM method proposed in the literature. It was introduced by Albrecht in 1979 [10] to have a measure (the function points) able to overcome the limitations of LOCs, by sizing the 'functionality' provided by a software, and from the point of view of end users. FPA is considered a structured method to perform a functional decomposition of the system whose size is the (weighted) sum of unitary elements (its FURs). The idea is that unitary elements can be measured more easily than the whole system. FPA has evolved in many ways. The original formulation was successively extended by Albrecht and Gaffney [10]. Since 1986 FPA is managed by the International Function Point Users Group (IFPUG) and it is named IFPUG FPA (IFPUG, for short), which has been standardised by ISO as ISO/IEC 20926:2009. Nevertheless, since FPA was originally designed from the experience gained by Albrecht on the development of management information systems, some researchers have analysed the applicability of FPA to other software domains [11, 12]. As a consequence of this debate on the application of FPA, many variants of the method have been defined for specific domains, such as MkII function point for data-rich business applications, or full function point (FFP) method for embedded and control systems [13]. All these variants are known and indicated as first-generation FSM methods since they are all based on the original formulation by Albrecht.

In the middle of the 1990s, important issues in the foundations of FPA against the measurement theory were highlighted in different researches. In particular, an improper use of different types of scales was highlighted in many steps of the FPA process. Moreover, the debate has interested how the 'weights' were defined and used in the method [14, 15].

At the end of the 1990s, a group of experienced software measurers formed the Common Software Measurement International Consortium (COSMIC) with the aim of addressing and overcoming the issues highlighted about the application of FPA and for defining a broader measurement framework able to tackle new IT challenges. The result of this investigation was the COSMIC-FFP method, which is considered the first 'second-generation FSM method'. To highlight this concept, the first version of the method was the 2.0. Successively, in 2007 the version 3.0 was characterised by many refinements and standardised as ISO/IEC 19761:2011. The method was named simply COSMIC. The current version of COSMIC is 4.0.1 [3], introduced in April 2015.

3.2.2 The COSMIC Method

The idea underlying the COSMIC method is that, for many types of software, most of the development efforts is devoted to handling data movements from/to persistent storage and users. Thus, a meaningful sight of the system size can be obtained by considering the number of these data movements [3]. As a functional size measurement method, the functional size is defined as the size of the software derived by quantifying the Functional User Requirements (FURs) [3]. FURs describe what the software is expected to do for its users. The standardised measure defined by COSMIC allows quantifying the functional size of the software in terms of COSMIC function point (CFP) units.

One of the main concepts underlying COSMIC is the functional process, which is defined as a set of data movements representing an elementary part of the FURs. A functional user is defined as a (type of) user that is a sender and/or an intended recipient of data in the FURs. Thus, a human or, for instance, an external device as well can play the role of a functional user. Another important concept that allows determining data movements is the boundary, which can be seen as a conceptual interface between the software being measured and its functional users. With these definitions of functional users and boundary, four different data movement types can be considered: an Entry (E) moves data from a functional user to a functional process; an Exit (X) moves data from a functional process to a functional user; a Write (W) moves data from a functional process to persistent storage; a Read (R) moves data from persistent storage to a functional process. One CFP unit is counted per each data movement, and the size of a software can be obtained by summing all the identified data movements.

The measurement process of COSMIC [3] is defined in terms of three phases: the ‘Measurement Strategy Phase’, the ‘Mapping Phase’ and the ‘Measurement Phase’ as depicted in Fig. 3.1.

3.2.2.1 Measurement Strategy Phase

The concept of Measurement Strategy Phase has been introduced in the last current version 4.0 of COSMIC [3], and it is meant to set the key parameters of the measurement: the purpose of measurement, the scope, the functional users and the level of granularity. The purpose defines what the measurement result will be used for; the scope specifies which pieces of software (in terms of FURs) have to be measured; the level of granularity describes how much detailed the documentation about the software is (e.g., in terms of the requirements description or also the structure description). The complete list of parameters can be found in the COSMIC Context Software Model, and it is necessary to carefully define them.

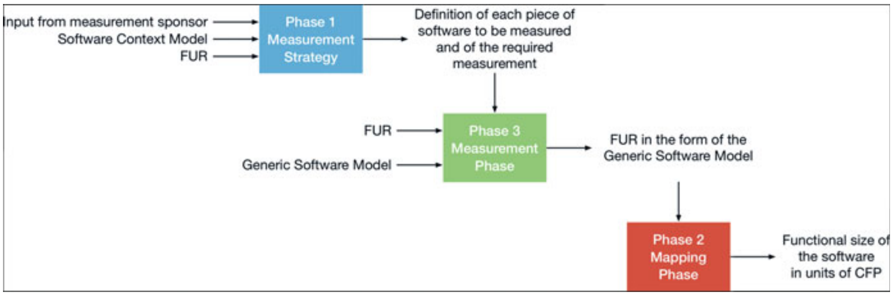
Input from measurement sponsor
Software Context Model
FUR
Phase 1 Measurement Strategy
Definition of each piece of software to be measured and of the required measurement
FUR
Generic Software Model
Phase 3 Measurement Phase
FUR in the form of the Generic Software Model
Phase 2 Mapping Phase
Functional size of the software in units of CFP

Fig. 3.1 The COSMIC method measurement process

3.2.2.2 Mapping Phase

This phase allows measurers to extrapolate the functional processes from the available FURs of the software being measured. In particular, a technical work has to be performed during Mapping Phase, carefully following the principles and, above all, the rules of the COSMIC method reported in the COSMIC Generic Software Model [3]. The potential functional processes inside the FURs can be identified by measurers looking at each functional process that is started by a triggering Entry and comprises at least two data movements: an Entry plus either an Exit or a Write. Indeed, the Entry of the functional user that starts the functional process can be identified as a triggering Entry. Other three crucial concepts are subprocess, data group, and data attribute. A subprocess may be either a data movement or a data manipulation. The COSMIC manual clearly suggests that the data manipulations inside a functional process are not counted as CFP. They are considered associated with the corresponding data movements. A data group is a distinct, non-empty and non-ordered set of data attributes, where each attribute describes a complementary aspect of the same object of interest. A data attribute is the smallest piece of information, within an identified data group, carrying a meaning from the perspective of the interested FUR. The object of interest is defined as any 'thing' that is identified from the point of view of the FURs. Thus, an object of interest may be any physical thing, as well as any conceptual object or part of a conceptual object in the world of the functional user about which the software is required to process and/or store data.

Four types of data movements are defined: each Entry, Exit, Read or Write is a movement of the data group of a single object of interest. An Entry moves a data group from a functional user across the boundary into the functional process where it is required; an Exit moves a data group from a functional process across the boundary to the functional user that requires it; a Read moves a data group from persistent storage within each of the functional processes that require it; and a Write moves a data group lying inside a functional process to persistent storage. There are only two exceptions: the triggering Entry which can start a functional process without data movement; e.g. in specific enquiry for a list of items, the error/ confirmation message that is defined as an Exit for the attention of a human user that either confirms only that entered data is accepted or only that there is an error in the entered data.

3.2.2.3 Measurement Phase

This Phase defines how to count data movements and it consists in associating a CFP to each data movement. The functional value of the measurement is obtained by considering the amount of all data movements. It is worth noting that the Measurement Phase may become more complex in cases (differently from our work) of aggregating measurement sizes (software stratified into different layers) or when measuring the size of software changes [3].

It is important to mention that the COSMIC community has also proposed approaches for counting the size of software in terms of COSMIC by exploiting approximate counts. We can highlight a couple of situations when there is the need of measuring a functional size approximately [3]: it can happen either early in the life of a project before the FURs have been specified in detail ('early sizing') or when a measurement is needed, but there is insufficient time or resources to apply the standard detailed method ('rapid sizing'). These motivations are not mutually exclusive and contribute to reaching a trade-off between a correct measurement and time and budget available.

3.3 Measuring Distributed Applications in Cloud Environments with COSMIC

As for any kind of software, a project manager that intends to provide a functional size of a project involving cloud environments must first define from which perspective the distributed application needs to be considered. There are three provision models of cloud computing, as shown in Fig. 3.2, in which the customers and the cloud vendors play a different role and have different responsibilities in the management of various aspects: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

In IaaS, the cloud vendor owns the hardware and network, and it is responsible for housing, running, and maintenance aspects. The customers can use virtual infrastructures (i.e., cloud instances), which run on physical resources but that can be created, reconfigured, resized and removed in a few moments based on the customers' or distributed applications' needs. The cloud instances consist of virtual machines for which the customer has full control of the environment. Before deploying a distributed application, the customer needs to install an operating system (OS) and the software stack. The main targets of IaaS are the system admins.

In PaaS, a platform is provided to customers that can run their applications and business in a distributed environment, without having to deal with lower-level

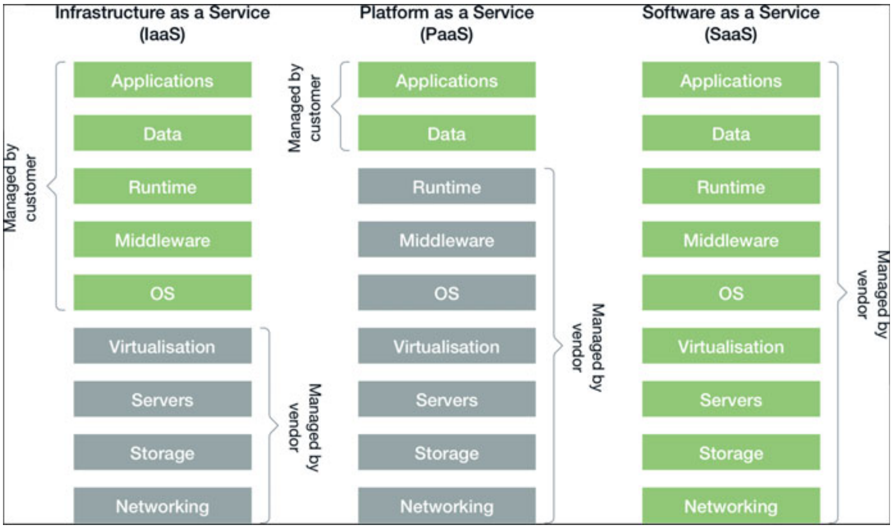

Fig. 3.2 Cloud provision models

requirements such as configuration, security, and management aspects. PaaS abstracts away all the aspects related to hardware decisions. Some examples of PaaS are database, development tools, and web server services. The target of PaaS is the developer, who writes codes that can run on the provided platforms.

SaaS is the top layer of cloud computing. The cloud vendor supplies software to customers in the form of a service such as e-mail clients, virtual desktop and communication services. The target user of SaaS is the end user, and it generally works on a subscription model, which means that the customer pays for the duration of time she/he uses the services. From an architectural point of view, distributed applications at PaaS and SaaS levels can be considered as service-oriented architecture (SOA) services, and therefore every existent contribute for SOA is also valid for distributed applications in the cloud environments.

The literature offers different approaches and guidelines [16–18] related to two main possible interactions: between cloud vendor components and customers and between SOA services. In the following, we collect and describe these literature contributes [16–18], giving specific considerations for cloud environments.

3.3.1 Measuring the Interaction Between Cloud Customers and Cloud Vendor Components

Schmietendorf et al. adapted the COSMIC method for cloud system size measurement [16]. In particular, they focused on the interaction between a cloud customer and the system provided by a cloud vendor.

For the authors, the COSMIC measurement strategy is defined considering the size of a chosen part of the developed cloud system for the purpose of measurement. They provided a rich list of possible size aspects such as the size of the involved services, the interaction between them, the customers, the resources, etc.

The scope of measurement is identified as the application of a service in the cloud system based on chosen characteristics (e.g., scalability, low cost), whereas the decomposition and granularity are established at five different levels, described below. The data movement-based interactions in the different cloud system levels correspond to the functional processes.

The FURs in the scope of measurement are based on the following characteristics:

- The on-demand services that are instantiated on the cloud system are controlled by triggering events and special object of interest like costs and resources.
- The network is the channel with which services communicate as separate layers, measured by means of COSMIC Entry and Exit data movements.
- Instead, Reads and Writes measure data movements from/to persistent storage with which the resource pooling is performed.
- The elasticity factor involves functional processes of service scaling on data groups like 'storage size' and 'location' requirements.
- The measured service itself supports the functional processes based on the object of interest like billing and service level agreement (SLAs) terms.

Thus, they investigated the cloud systems defining the following five levels of functionality (see also Fig. 3.3) and their interactions [16]:

1. The base level of virtualisation, representing the different virtual instances producing a service
2. The level of instances clustering on machine level
3. The level of multi-data centres, representing the aspects of multiple redundancies of machines on differently located data centres
4. The service level of the SOA, where the distributed application is run and the measurement is performed in terms of service interactions
5. The service chain levels if the application is distributed through different cloud vendors

Another important contribution is provided by Vogelezang et al. [17]. The authors overviewed the application of the COSMIC method to modern software such as mobile and cloud applications. They also established four possible contexts in which different kinds of FURs can be identified, considering and measuring different data movements (see Fig. 3.4). Each context is an extension of the previous one, increasing the deepness in the cloud infrastructure:

1. The interactions are those between the functional user and the user interface service (UIS) running in the cloud environment and between user interface and some business processes.

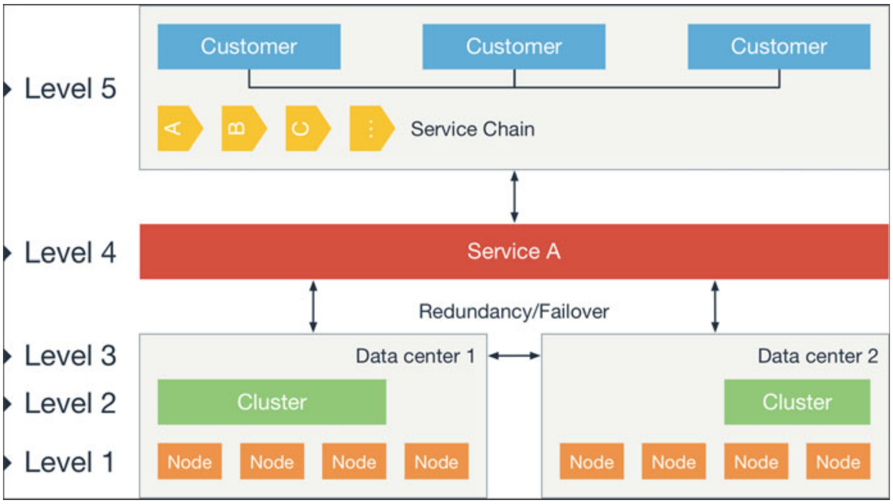

Fig. 3.3 Considered cloud system architecture

2. The FURs generated involve functional changes to hosted services on different machines in the same cluster.
3. The FURs include intra-component data movements between hosted services on different clusters.
4. An integration service allows the direct interaction between all the services in the application stack, using the integration service.

3.3.2 Measuring the Interaction Between SOA Services in the Cloud Environments

Distributed applications in cloud environments can be considered as a particular form of SOA software in which cloud-specific services can occur. Thus, the official guideline for sizing SOA with COSMIC can be applied [18].

An SOA-based software is designed following a specific pattern in which application components provide services to other components, exchanging data through a communication protocol (e.g., messaging queue), typically over a network. Even though sizing service-oriented software with Function Point Analysis fails when reconstructing or mapping the Functional User Requirements, the COSMIC method defines the concept of 'layers' that perfectly matches the SOA-based software sizing, without needing to adapt the method in any particular way.

The organisation for the Advancement of Structured Information Standards (OASIS) defines SOA as 'a paradigm for organising and utilising distributed

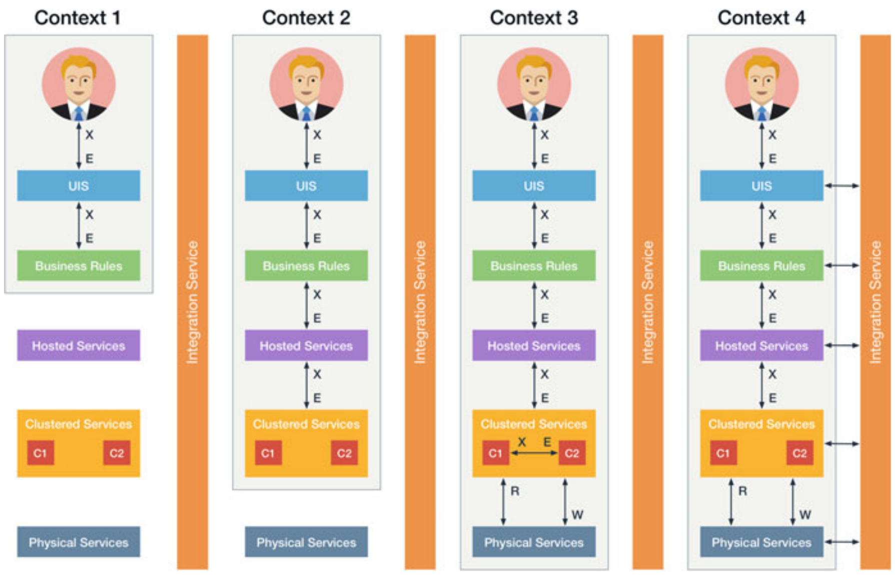

Fig. 3.4 Example of data movements in cloud software

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' [19].

Services are accessible to each other by means of a public interface, in the form of application program interfaces (APIs) often through simple Create/Read/Update/Delete (CRUD) operations. Not only do the APIs allow the services to communicate with each other but also they make the service independent in terms of development and execution. Indeed, the calling software does not have or need to know anything about how a service actually performs its tasks. This eases the independent work of parallel development teams and the inclusions of third-party components in SOAs. APIs are also useful for COSMIC sizing since they are independent of both involved technologies and implementations. Usually, APIs strictly follow the definition of FURs of the distributed application, being available from the early stages of the project.

Table 3.1 shows the classification given by the COSMIC guideline for SOA software [19].

The 'application services' provide specific business operations. They implement the features that characterise the distributed application; thus the most of the FURs are related to them. Each functionality can be invoked using an API in the form of network messages.

The role of 'orchestration services' is to call and control other services, often in an automatic way. An example of cloud orchestration service is the load balancer: it probes other services in the cloud system and checks the network and resource

Table 3.1 COSMIC guideline classification of service

Term used	Alternative terms	Description
Application service	Business service, entity service	Provides business functionality of an application
Orchestration service	Process service, task service	Controls ('orchestrate') application services to implement a (business) process
Intermediary service	Internal service, mediation service	Ensures independence of service requestors and service vendors
Utility service	Public service, software service	Provides common functionality (business or non-business) independent of, but made available to, any other applications

usage. It can scale the cloud system, by means of service instance replicas, to guarantee a balancing of load between the services and their consumers' satisfaction. Usually, the load balancer communicates with other services and, at the same time, with the cloud vendor, through the exposed API, being able to request the allocation of new resources or destruction of useless ones.

The communication management is delegated to the 'intermediary services' (i.e., message brokers) that interconnect a requestor's message with one or more application services. The intermediary services oversee controlling request and response messages, translating the language of messages and dealing with exception situations. Besides enabling service 'requestors' to communicate with service 'providers', another purpose of intermediary services is to ensure the independence between the two actors. This ensures flexibility allowing service vendor to change without needing to change service requestors and vice versa. This allows a high level of modularisation of the distributed application. A good example of intermediary service is a message queuing protocol: it ensures that the sent messages reach their destination, and it maintains a copy of messages until the requestors receive a confirmation or in the case of problems.

The 'utility services' provide functionality independently from other applications or services. For instance, a log service can be employed to monitor the application status to measure the service level agreement conditions. Also, they can be used to monitor the performance of cloud instance for statistics purposes. Another common example, especially in the cloud environment, is the discovery service: it allows services to reach and be reachable from other services. It is responsible for registering new services so that other services can query it and retrieve information data.

3.3.2.1 Measurement Strategy Phase

As for any target of a COSMIC measurement, for the SOA it is needed to specify the following elements.

In the case of measurement at the service level, usually the purpose of measurement is the estimation of the effort needed for the development or modification of the system. The measurement can be performed with different scopes. The guideline for SOA [18] distinguishes between the cases where the purpose is to measure a piece of software 'using' multiple SOA services and when the purpose is to measure a 'collection' of multiple SOA services. For the first cases, the scope should be defined without considering internal data movements. In the latter cases, the size of the distributed application is equal to the sum of the sizes of single services.

What makes the COSMIC method particularly suitable for SOA services is the fact that it allows for the size measurement of multilayer applications. A layer, as defined by COSMIC, provides a set of services, which can be utilised by the software in other layers and can be part of a structure either hierarchical or bidirectional. In particular, in SOA the orchestration services can call application services but not vice versa establishing a hierarchical relationship. Intermediary services can be called by application service and vice versa, and both orchestration and application services can call utility services.

If the services in different layers need to be measured, the measurement scopes must be different. Figure 3.5 shows that the definition of what is considered as 'layer' depends on the 'view' of the software architecture. If the purpose is to measure the size of application A 'as a whole' as in (a), the measurement scope is the whole of application A as a single layer. If application A has been built according to the 'three-layer' architecture, the purpose is to measure the three components separately in view (b). In the case of SOA, the measurement scope must be defined separately for each SOA component.

Figure 3.6 shows an example of relationships between some SOA services [3]. The possible data movements between services are in the form of COSMIC Entry and Exit movements. COSMIC boundaries are set between a service and another. In the cloud context, also a persistent storage can be expected if the measurement scope includes the interaction between the service and the data storage on the same virtual instance where the service is running, thus within the boundary of the calling service. In this case, the data movements are expressed in terms of COSMIC Write and Read movements. Otherwise, if the storage is provided as a separate data service (e.g., a database service), another boundary involving Entry and Exit movements is needed.

The functional users of SOA services depend on the scope of the measurement and the purpose of the measurement. For instance, the functional user of an orchestration service is the application service that calls it. In the case of application services, its functional users can be the end users or other kinds of services that call it. For an intermediary service, the functional users are the application services that call it and the application and utility services that are called by it.

Let us assume to have a distributed application, defined with an SOA and composed of several services: the application itself, a log utility, a database and a load balance services. From the application service perspective, its functional users are all the other services. The log utility service interacts only with the application service in the same way for the database service. The load balance service, instead, can interact both with all the services of the SOA architecture but also with the

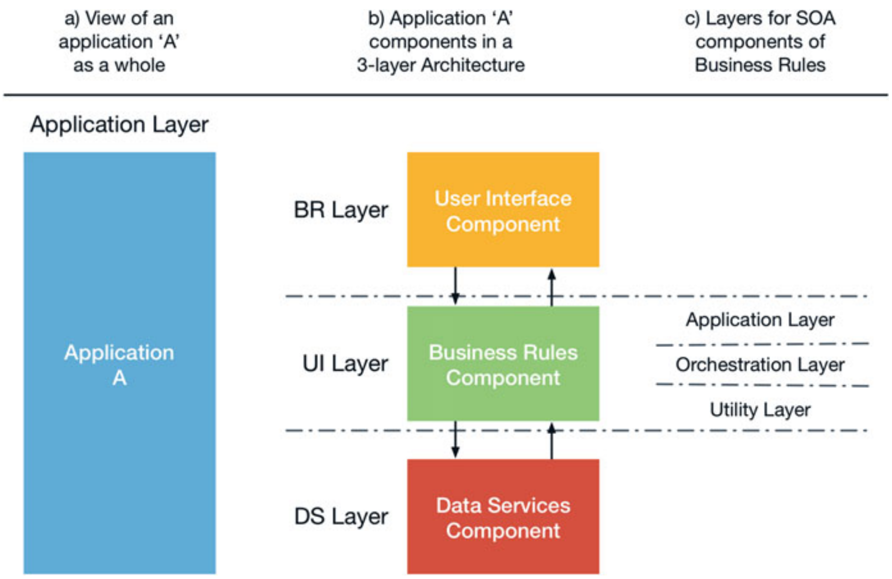

Fig. 3.5 Three views of the layers of an application

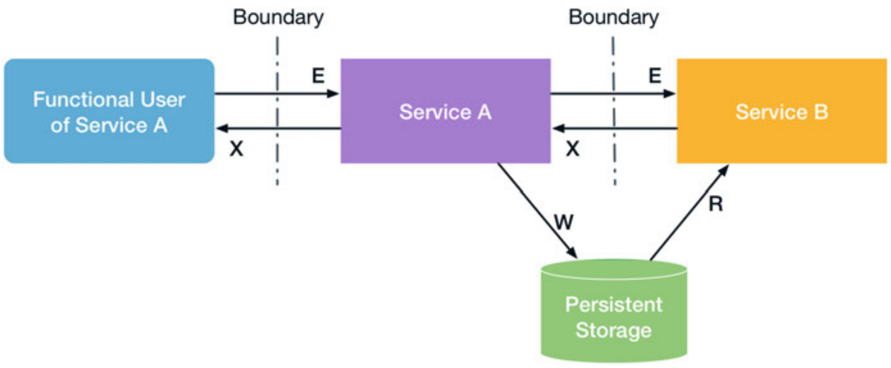

Fig. 3.6 Data movements between SOA services

cloud vendor service aiming, at the same time, to monitor the status of the resources in the system and demanding for creating/destroying resources to the cloud vendor.

3.3.2.2 Mapping and Measurement Phase

An FUR of a service may be restricted to define the 'capability' it provides for any service requestor, 'how' to request the capability and the 'form and content' of the request and reply messages. Some requirements that usually are considered as non-functional, in the case of SOA services, may be implemented directly as software and therefore are part of the software. For instance, the security requirements need to follow precise protocols (e.g., OAuth) and the protocols being implemented in the software itself. If the protocol is in the scope of the measurement, it requires being measured as well.

The COSMIC method considers that unique events give rise to one or more functional processes whose role is to respond to the events. The first step for the measurement is to identify these functional processes and events. In the field of SOA, there are no standards about considering whether the concepts of ‘service’ and ‘functional process’ coincide; thus it is not excluded that one service may lead to multiple functional processes.

As mentioned before, the communication between services consists of messages exchanged, and therefore developing a service always involves developing the request/reply mechanism. The exchange between components may be ‘synchronous’ if the requesting service waits for the response before continuing its task. As an example of cloud service, it may be a web server that queries a database service to retrieve information. It can also be ‘asynchronous’ if the requestor functional process does not wait for the response message. For instance, an application service may ask for a time-consuming task to another service, without blocking its task. Once the latter finishes its job, the result can go back to the original requestor. In terms of COSMIC data movements, the main difference between the two kinds of services is that with the asynchronous mode, the arrival of a response message needs to be considered as another event triggering a separate functional process in the requesting software.

Another important thing that a COSMIC measure must deal with when measuring SOA services is the error message management. Technically speaking, in case a requestor calls another service and any issue occurs, the response message is replaced by the error data itself. In terms of COSMIC data movements, an Exit only is considered in any case. Nevertheless, if a confirmation/error message is notified to a human functional user or another service, an additional Exit must be considered.

3.4 Conclusions and Future Research Directions

It is widely recognised that the competitiveness of software companies greatly depends on the ability of their project managers to carry out a reliable and accurate software size estimation. Among the approaches proposed to size software FSM methods are widely applied in the industry since size estimation can be obtained early, based on the functionality provided to the users. In this chapter, we have analysed and discussed the main aspects of the use of the COSMIC method to measure distributed applications in cloud environments. In the discussion, we have considered the three distinct provision models of the cloud computing stack: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Furthermore, we have analysed specific concepts for distributed applications such as orchestrations and load balancing of several components that act together to realise the required functionality and to ensure critical non-functional requirements (e.g., scalability, reliability).

It is clear that the effort for a software project and its related cost depend on both functional and non-functional aspects. On the basis of this consideration, in the last years, some approaches have been defined to measure the non-functional requirements of software systems to complement the information given by FSM methods. One of the most interesting examples is the Software Non-functional Assessment Process (SNAP) [20] devised by the IFPUG community. The aim of IFPUG is to capture functional aspects through the use of FPA and the non-functional ones with SNAP. The SNAP model consists of four categories and 14 subcategories to measure the non-functional requirements. Non-functional requirements are mapped to the relevant subcategories, and each subcategory is sized, and the size of a requirement is the sum of the sizes of its subcategories. These sizes are then summed to give the measure of the non-functional size of the software application. At the present, no empirical study demonstrating the effectiveness of SNAP is reported in the literature. Moreover, one of the main challenges is to understand which non-functional requirements do not give rise to functional components that are measured by FSM methods. In particular, there is the need to understand these aspects for the cloud environment also providing a specific SNAP approach. It is our intention to fill this gap as future work, in order to give a measure of the non-functional requirements of distributed applications.

Furthermore, empirical studies, possibly in the context of software companies, should be carried out measuring distributed applications, applying both COSMIC and SNAP and assessing the predictive accuracy of the built effort estimation models.

References

1. ISO (2007) ISO/IEC 14143–1:2007: information technology – software measurement – functional size measurement – part 1: definition of concepts
2. Albrecht AJ (1979) Measuring application development productivity. In: Joint SHAREGUIDEIBM application development symposium. pp 83–92
3. Abran A, Baklizsky D, Desharnais J-M, Fagg P, Gencel C, Symons C, Ramasubramani JK, Lesterhuis A, Londeix B, Nagano S-I, Santillo L, Soubra H, Trudel S, Villavicencio M, Vogelezang F, Woodward C (2015) The COSMIC functional size measurement method, measurement manual
4. Di Martino S, Ferrucci F, Gravino C, Sarro F (2016) Web effort estimation: function point analysis vs. COSMIC. Inf Softw Technol 72:90–109. doi:10.1016/j.infsof.2015.12.001
5. van Heeringen H, van Gorp E (2014) Measure the functional size of a mobile App: using the COSMIC functional size measurement method. In: Joint conference international workshop software measurement and the international conference software process and product measurement. IWSM-MENSURA. IEEE, pp 11–16

6. Ferrucci F, Gravino C,Salza P, Sarro F (2015) Investigating functional and code size measures for mobile applications: a replicated study. In: International conference prod.-Focus. Software process improvement. PROFES. pp 271–287
7. Nitze A, Schmietendorf A (2014) An analogy-based effort estimation approach for mobile application development projects. In: Joint conference international workshop software measurement and the international conference software process and product measurement. IWSM-MENSURA, pp 99–103
8. Zuse H (1997) A framework of software measurement. Walter de Gruyter & Co., Berlin
9. Trendowicz A, Jeferry R (2014) Software project effort estimation. Foundations and best practice guidelines for success. Springer, Cham
10. Albrecht AJ, Gaffney JE (1983) Software function, source lines of code, and development effort prediction: a software science validation. IEEE Trans Softw Eng 9:639–648
11. Conte SD, Dunsmore HE, Shen VY (1986) Software engineering metrics and models. Benjamin-Cummings Publishing Co., Inc., Menlo Park
12. Fenton NE (1991) Software metrics: a rigorous approach. Chapman and Hall, London
13. Gencel Ç, Demirörs O (2008) Functional size measurement revisited. ACM Trans Softw Eng Methodol. doi:10.1145/1363102.1363106
14. Abran A, Robillard PN (1994) Function points: a study of their measurement processes and scale transformations. J Syst Softw 25:171–184. doi:10.1016/0164-1212(94)90004-3
15. Kitchenham B (1997) Counterpoint: the problem with function points. IEEE Softw 14:29–31. doi:10.1109/MS.1997.582972
16. Schmietendorf A, Fiegler A, Wille C, Dumke RR, Neumann R (2013) COSMIC functional size measurement of cloud systems. In: Joint conference international workshop software measurement and the international conference software process and product measurement. IWSM-MENSURA. IEEE, pp 33–37
17. Vogelezang F, Ramasubramani JK, Arvamudhan S (2016) Estimation for mobile and cloud environments. Mod Softw Eng Methodol Mob Cloud Environ
18. Baklizky D, Lesterhuis A, Ozkan B, Symons C, Frank V (2015) Guideline for sizing service-oriented architecture software:1–31
19. Santillo L (2007) Seizing and sizing SOA applications with cosmic function points. Software Measurement European Forum SMEF
20. International Function Point Users Group (IFPUG) (2015) Software non-functional assessment process (SNAP), Assessment practices manual