Modern Software Engineering Methodologies for Mobile and Cloud Environments

António Miguel Rosado da Cruz
Instituto Politécnico de Viana do Castelo, Portugal

Sara Paiva
Instituto Politécnico de Viana do Castelo, Portugal

A volume in the Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) Book Series

Information Science REFERENCE
An Imprint of IGI Global

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA, USA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Names: Cruz, Antonio Miguel Rosado da, 1970- editor. | Paiva, Sara, 1979- editor.
Title: Modern software engineering methodologies for mobile and cloud environments / Antonio Miguel Rosado da Cruz and Sara Paiva, editors.
Description: Hershey, PA : Information Science Reference, 2016. | Includes bibliographical references and index.
Identifiers: LCCN 2015046896| ISBN 9781466699168 (hardcover) | ISBN 9781466699175 (ebook)
Subjects: LCSH: Cloud computing. | Mobile computing. | Software engineering.
Classification: LCC QA76.585 .M645 2106 | DDC 004.67/82--dc23 LC record available at http://lccn.loc.gov/2015046896

This book is published in the IGI Global book series Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) (ISSN: 2327-3453; eISSN: 2327-3461)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 4
Estimation for Mobile and Cloud Environments

Frank Vogelezang
Ordina, The Netherlands

Jayakumar Kamala Ramasubramani
Amitysoft Technologies, India

Srikanth Arvamudhan
Amitysoft Technologies, India

ABSTRACT

Estimation of Cost, Effort and Schedule is a very important aspect in commercial software development. Effort is usually the predominant cost driver in software development. The dominant determinant for effort is the size of the software to be developed. There are various ways to determine the size of software. The best option is to use a standardized measure for the functional size. In this chapter the COSMIC method for functional size is introduced. Due to its basic principles, the COSMIC method enables determination of the functional size of mobile and cloud-based applications. This chapter demonstrates how the COSMIC method provides a good basis for the estimation of Cost, Effort and Schedule in mobile and cloud environments.

INTRODUCTION

Estimation of Cost, Effort and Schedule is a very important aspect in commercial software development, since it is used to allocate resources, plan product releases and negotiate payments between suppliers and contractors. In the Software Engineering Body of Knowledge, Estimation is specified as a sub area of Project Planning for the Software Engineering Management knowledge area (IEEE Computer Society, 2014). In software development effort is usually the predominant cost driver. Estimation of Cost and Schedule, however, depends on the estimate of Effort. The dominant determinant for effort is the size of the software to be developed.

DOI: 10.4018/978-1-4666-9916-8.ch004

There are various ways to define size, which can be divided into technical and functional size measures. Technical size measures are easy to extract from the software itself, but have proven not to be very accurate predictors for effort estimation (Jones, 2013). In traditional software development a number of so called first generation functional size measurement methods have been developed such as Function Point Analysis, Feature Point Analysis and Use Case Points (Vogelezang, 2013a). These methods were developed empirically and were based on the paradigm of stand-alone applications with limited interactions with other software. This paradigm base leaves them not really suited for mobile and cloud software, with their layered architecture and multi-level interaction with other software components.

The COSMIC method is a second generation functional size measurement method that is based on the idea that software can exist in multiple layers and that moving data from one software component to another is the main predictor for the functional size of software (Vogelezang, 2013b). COSMIC is increasingly recognized for its applicability in estimating the development effort for mobile and cloud solutions involving multilayer and service oriented business applications.

BACKGROUND: FUNCTIONAL SIZE

Organizations engaged in software engineering have struggled for years in search of acceptable quantitative methods for measuring process efficiency and effectiveness, and for managing software costs, for the systems they acquire, develop, enhance or maintain. One critical, and particularly elusive, aspect of this measurement requirement has been the need to determine software size. In 1998, the concept of functional size was defined in an International Standard (ISO, 2007).

Functional size is a measure of the amount of functionality provided by the software, derived by quantifying the user practices and procedures that the software must perform to fulfill the users' needs, independent of any technical or quality considerations. The functional size is therefore a measure of what the software must do, not how it should work. This means that the functional size can be determined before the actual software has been built, and even before there is a final decision on what platform the software will be built and in which programming language.

The concept of functional size is therefore an ideal basis for comparison, either between software with similar characteristics or for cross-platform comparison. These different types of comparison can be useful for development or maintenance benchmarking or to support decisions on platform choices.

Functional Size Measurement

Functional Size Measurement deals with the measurement of functionality to be developed based on requirements for the software. Requirements describe the functionality to be built and certain characteristics of that functionality in order to deliver the software fit for the purpose. ISO has developed a meta standard for functional size measurement methods (ISO, 2007). All methods for functional size measurement must comply with this standard. ISO has recognized the COSMIC method, which is fully compliant with the meta-standard, as a standard for functional size measurement in 2003 (ISO, 2011).

Although the prime objective of the COSMIC method is to measure the functional size, a limited field trial on real-time applications was done to verify whether the COSMIC method was suitable for predicting Effort for specification, build and test, before submitting the method as an International Stan-

dard (Abran, 2001). More recently the International Software Benchmarking Standards Group analyzed performance data on 324 business application software projects, 40 real-time software projects and 22 projects that produced software components (ISBSG, 2012). Their conclusion was that the COSMIC benchmark data was self-consistent for all types of projects analyzed and well-suited for both external performance comparisons and for new project estimating. In comparing the data with IFPUG benchmark data the COSMIC method gives more differentiated figures for differing sets of project characteristics.

The COSMIC Method

The COSMIC method made a significant shift from the first generation methods by using a new paradigm and by making use of established software engineering principles. The COSMIC method is a second generation functional size measurement method that is based on the idea that software can exist in multiple layers and that moving data from one software component to another is the main predictor for the functional size of software. The new paradigm ensures that the method can be applied to all software where moving data is the main predictor for the functional size. This means that the COSMIC method can be applied to business application software and real-time software alike, making it a very powerful estimating resource in the mobile and cloud domain. The COSMIC method is maintained by the Common Software Measurement International Consortium.

THE COSMIC MODEL OF SOFTWARE

The COSMIC method regards software as a set of *functional processes* based on the functional user requirements. Users interacting with software across boundary through functional processes are referred as *functional users*. Functional processes do two things: they move *data groups* to and from the functional process and they manipulate data groups within the functional process. Data manipulation is assumed to be accounted for by the movement of the data groups associated with the manipulation.

The movement of a single unique data group to or from the functional process is called *data movement*. Based on the functionality of the process, data groups move between 'user and the software' as well as between the 'software and persistent storage'. The COSMIC method distinguishes four types of data movements (COSMIC, 2015a).

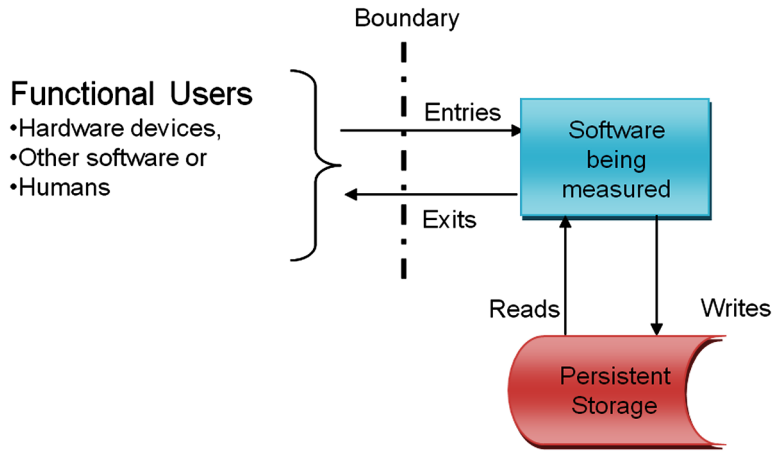
- *Entry* data movements move data into the software from functional users
- *Exit* data movements move data out of the software to functional users
- *Read* data movements move data from persistent storage to the software
- *Write* data movements move data from the software to persistent storage

This is depicted in figure 1.

COSMIC functional size measurement is based on the number of data movements related to functional processes executed by users (figure 2).

The sum of all data movements associated with all functional processes associated with the software is the size of software in *COSMIC Function Points* referred as *CFP.*

Figure 1. The four types of data movements



The COSMIC Measurement Process

In order to determine the functional size you need a repeatable process. The COSMIC method has a very well described measurement process (COSMIC, 2015a), consisting of three main steps:

First, it must be defined what will be measured. Therefore there must be agreement on the purpose and the scope of the measurement. This determines who or what are defined as functional users of the software. The next step is to transform the functional description into functional processes and data movements of the COSMIC generic software model.

The final step is the actual measurement. The size of the software is determined by identifying all the data movements (Entries, Exits, Reads and Writes) of each functional process and adding the number of data movements over all the functional processes.

Figure 2. COSMIC generic software model

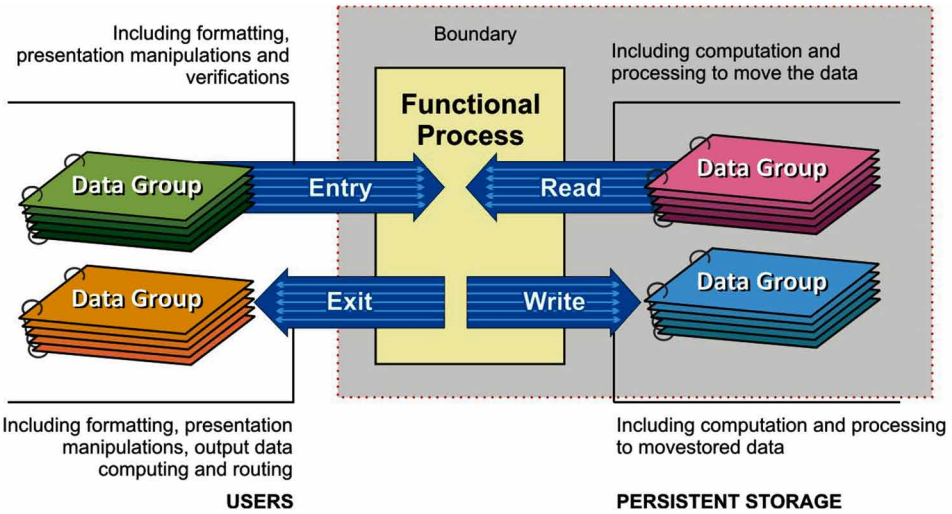


Figure 3. COSMIC measurement process

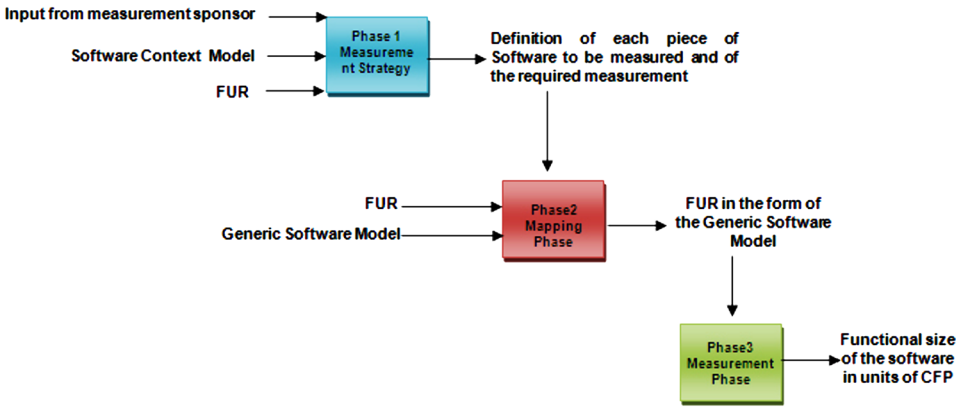


Figure 3 illustrates these steps of COSMIC Measurement process with inputs to and outputs from each of the stages.

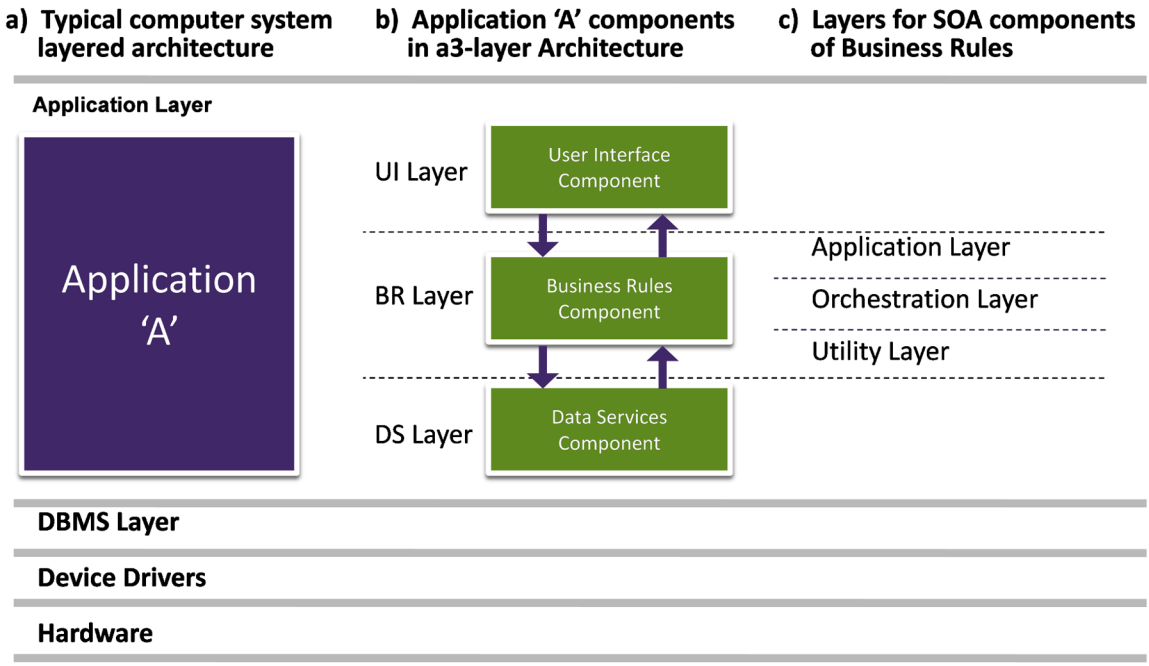
The COSMIC Concept of Functional User

One of the key concepts of the COSMIC method is the functional user (COSMIC, 2015a). Functional users can be human beings, software or engineered devices that interact with the software being measured. In general, functional users can be identified as the senders and/or intended recipients of data to and from the software. This can be human users, engineered devices, and peer components on the same platform or external applications on a different platform (figure 4).

Different types of users may require different functionality and therefore functional sizes will vary with the choice of functional users. This is why the measurement strategy is an important step in the measurement process. Thus, COSMIC measure is based on functional user requirements (FUR) as opposed to implementation artifacts or objects. Although COSMIC is the only method that describes such a process, this process is applicable to all kinds of measurement processes.

Figure 4. Functional Users of Software

Figure 5. Three views of Layers



The COSMIC Concept of Layer

COSMIC is the only functional size measurement method to recognize layers according to the standards of software architecture. Functional requirements can be mapped to any layer based on the nature of services provided. Figure 5 provides three views of layers as addressed by (COSMIC, 2015a).

In the first view the application is considered to be a monolith that contains all functionality, with the exception of the database management system, the device drivers and the operating system. This type of architecture is becoming very rare and mostly used for 'quick and trivial' small business software solutions, like temporary MS Access programs.

The most common architecture for software for the networked world using the Internet as back bone is the three layered architecture in which the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules on separate platforms. Layering is a software design pattern and is well-established software architecture (Janssen, 2014) for building complex software. This architecture allows any one of the layers to be upgraded or replaced independently.

- The user interface component is implemented on the user device and uses the standard graphical user interface of that device.
- The business logic component can be either on the user device (this is very common in mobile applications) or on an application server of a cloud solution or business administrative system.
- The data services component contains the computer data storage logic and is usually located on a data server in the cloud or as a separate part of a business administrative system.

In service oriented architecture, the business logic is split up into services that perform a specific task. The business logic layer itself is divided into multiple-layers. Service oriented architecture, adopted in cloud environment uses this paradigm (COSMIC, 2010).

- The application layer provides specific, limited operations as services. Such services may be regarded as a request/reply mechanism that feeds the user interface or orchestration layer with reusable functionality. Common examples in the application layer are services to create a booking or services to verify the authentication or authorization of the functional user of the user interface component.
- The orchestration layer calls and controls other services to implement a complete (business) process. It handles the communication with multiple application services and/or business services that process a part of the (business) process. Examples of an orchestration are a risk analysis in a mortgage application or a location-based travel advice from a public transportation app.
- The utility layer provides common functionality (business or non-business) independently of, but available to, the other layers or even other applications. Common examples of services in the utility layer are logging and a file parsing API.

Most of the current first generation functional size measurement methods can only deal with the first view presented in the above figure, where the application is considered to be a monolith that contains all functionality. This makes them less powerful for estimating mobile or cloud software. Cloud and mobile software are usually constructed with a design pattern from the second or third view. The different layers are usually developed and maintained as independent modules on separate platforms by separate teams, so they need to be estimated separately (Janssen, 2014).

In COSMIC, sizing and estimation can be performed specific to a layer and hence provides more accurate and useful measures. As the technology for implementing different layers can vary, concerned productivity factors can be used to measure the development effort accurately than using a single productivity factor for the entire application.

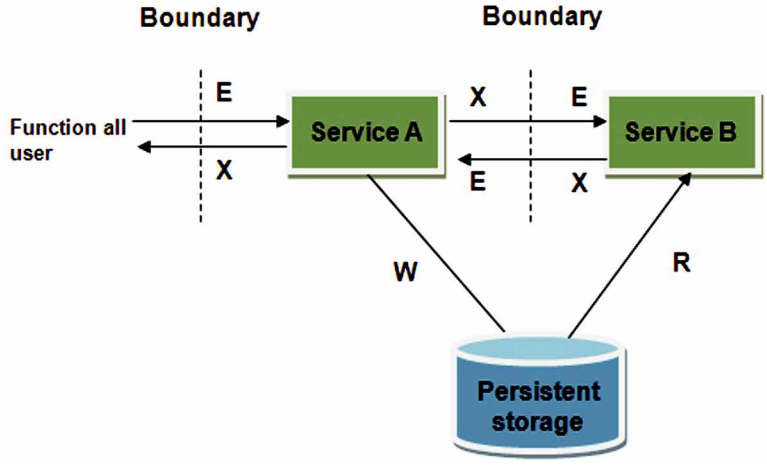
The COSMIC method contains a guide that provides a mapping of the COSMIC principles to service oriented architecture. It describes how services and their interactions can be measured to determine the functional size (COSMIC, 2010). The COSMIC method is equipped to measure functional processes that reside in different layers, but use the same stored data, as shown in the figure 6. This is a concept that is not present in any of the first generation functional sizing methods, but very common to cloud and mobile software.

APPLICATION OF THE COSMIC METHOD FOR MOBILE ENVIRONMENTS

Mobile Computing Characteristics

Software operating in Mobile environment can be classified as a hybrid of both real-time and business application types. Operating system and critical components are driven in real-time like any other real-time systems. Mobile Apps built on the top behave like business applications with characteristics specific to mobile environment. Although Mobile apps could be regarded as a modern type of client-

Figure 6. Mapping of COSMIC data movements in a service oriented architecture



server architecture, they are different from traditional applications, in a number of ways (van Heeringen & van Gorp, 2014).

- The user can interact with the apps in more ways than in traditional applications. For instance, the app can react to changing the position (toggling) of the mobile device, to shaking the mobile device and some apps can accept voice messages.
- Some apps respond to real-time events as well, like moving of the mobile device, switching from Wi-Fi to cellular and back or when the network is out of reach.
- Apps must have functionality that handles interruptions, like an incoming call.
- There are usually important non-functional requirements to which an app should comply, e.g. security, performance, minimum data traffic, space occupied on the device and consumption of battery power.

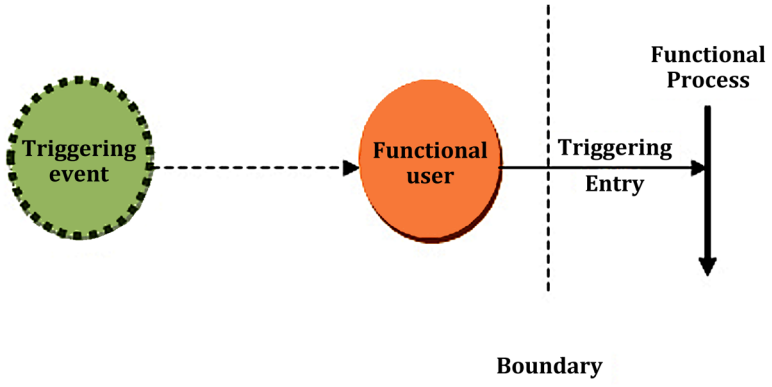
Mobile Architecture Context

The concept of functional user in COSMIC addresses interactions by users of all types that is present in a mobile environment. Functional Users can be human users, engineered devices, and peer components on the same platform or external applications on a different platform.

In the COSMIC method all functional processes respond to events in the real world. An event is sensed by a functional user. Functional user triggers a functional process. When a sensor detects a condition to which the software must respond sensor becomes the functional user and the triggering event is the condition that the sensor is designed to detect. The triggering entry data movement is the message from the sensor to the software announcing that the event has occurred. The message may also carry data about the triggering event. This is depicted in figure 7 (COSMIC, 2015a).

While the processes in business application software mostly deal with data inputs and outputs, there is significant amount of internal processing in case of mobile applications that is not directly related to the processing of the inputs and outputs, like visualizing in graphs or images or highlighting of certain data, based on processing rules. Any process that has a significant amount of internal processing but little external visibility will not be accredited for the full functionality it delivers, when measured using first generation methods such as IFPUG FPA.

Figure 7. The relationship between events, functional users and functional processes



Mobile applications do not have the same emphasis on stored and maintained data as business application software. They connect to cloud or back-office systems and retrieve only the results of the data processing. Internal processes often use real-time information to present the retrieved data in the right context. The data input to the process is not always permanently stored but retrieved externally and used during the concerned process requiring the data. Even when it comes to non real-time applications such as business apps, some data may be stored on the mobile device, but most of the data will be in the cloud or on back-office servers. The exact architecture may vary for each app, but as a general rule mobile apps do not have much emphasis on stored data.

The first generation functional size measurement methods heavily rely on functionality around stored and maintained data. It is not possible to map these first generation methods such as IFPUG function points to a mobile architecture in a natural way where the concept of internal and external files have to be explicitly identified. The COSMIC method directly fits into the needs for sizing and estimation in mobile environment for developing both real-time components and business apps. The COSMIC method only has to know that there is a data source somewhere to retrieve data from. This idea is illustrated in figure 8.

In a mobile app human users communicate with the app by means of Entry and Exit data movements. Typically the app will communicate real-time with peer components on the mobile device also via Entry and Exit data movements. If data needs to be stored on the device the app will communicate with the permanent storage via Read and Write data movements. The app can communicate with an administrative back-end system or with a cloud service via Entry and Exit data movements.

Example of a Banking App

To illustrate the COSMIC method we show a common function of a banking app (figure 9), the transaction overview. This is a basic view functional process:

E Triggering entry
X Request for transaction data to the Orchestrator
E Reception of transaction data
E Reception of Orchestrator error messages
X Show transactions
X Show application error message

Figure 8. Example of data movements in a mobile app

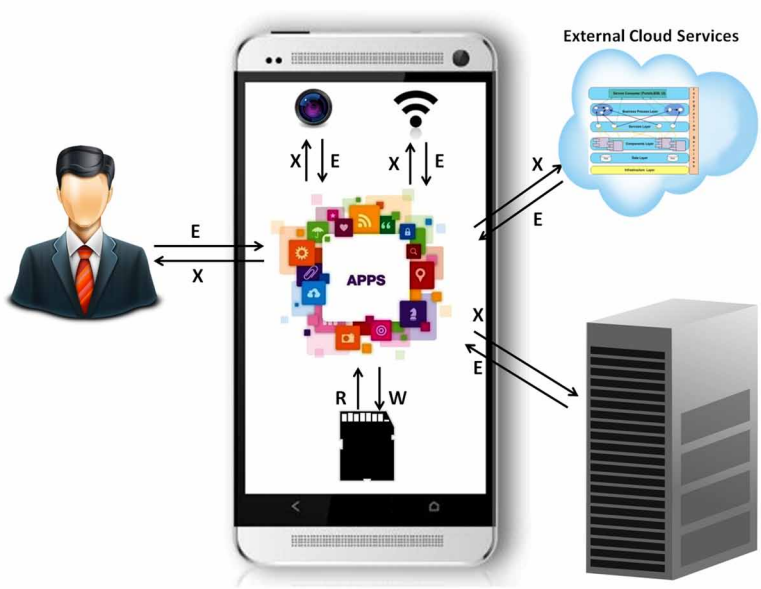
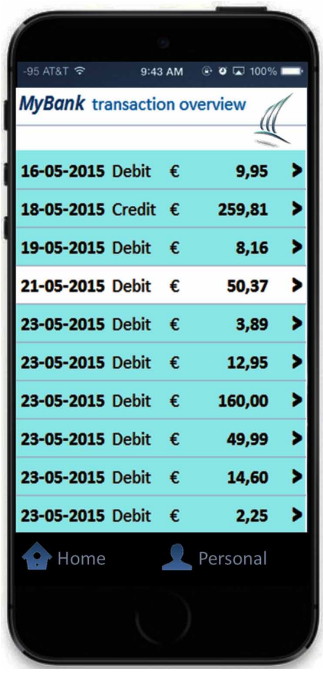


Figure 9. Banking app



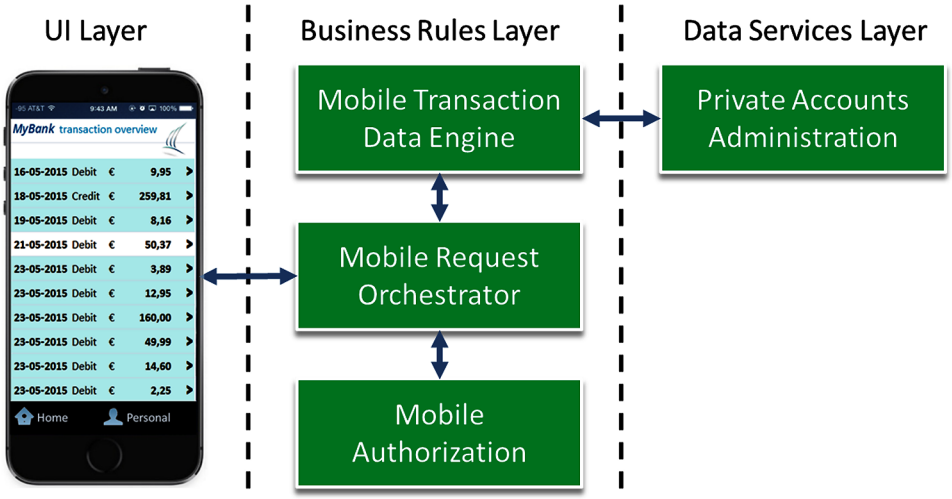
This functional process consists of six data movements and thus the functional size of the transaction overview is 6 CFP. This functional process includes the view of all data attributes of a data group that belongs to one object of interest. When tilting the mobile device, more data attributes are shown. This is considered to be the display of the same data group and does not lead to a new functional process. Tilting the device is considered to be a control command within this functional process that is not counted as a separate data movement. For the measurement of the functional size this is similar to the ability to change the view of the displayed data (e.g. by filtering or sorting the data) which also does not lead to the identification of additional data movements (van Heeringen & van Gorp, 2014).

The transaction overview is only the app functionality. To be able to present the user of this app with his or her financial transactions this is not the only functionality that needs to be developed to make the app work. As stated before, COSMIC functional size is counted per architecture layer. In this case the relevant part of the full architecture of the bank is shown in the figure below (Figure 10).

The app sends the request to the Mobile Request Orchestrator component in the Business Rules layer. The Orchestrator verifies with the Mobile Authorization component that the mobile device is authorized to receive financial transactions. When the device is authorized, the Orchestrator requests the transaction data from the Mobile Transaction Data Engine and sends them back to the App. When either the authorization fails, or there is no transaction data available. The Mobile Transaction Data Engine is an exact copy of the financial transactions in the Private Accounts Administration that only holds the data attributes that will be used on mobile devices. The Private Accounts Administration is not capable to service the number of requests from mobile devices, therefore the Data Engine component is introduced in the bank's architecture.

For the transaction overview, functional processes in all components shown in the figure above are necessary:

Figure 10. Essential elements of the banking architecture that support the Mobile App



Mobile Request Orchestrator

Generate Transaction Overview Functional Process

E Request for transaction data from the Transaction overview*(triggering Entry)*
X Authorization request to the Mobile Authorization component
E Reception of Mobile Authorization information
E Reception of Mobile Authorization error message
X Request for transaction data to the Mobile Transaction Data Engine
E Reception of transaction data from the Mobile Transaction Data Engine
E Reception of Mobile Transaction Data error message
X Send transaction data to the App
X Send error messages to the App
9 CFP

Mobile Authorization

Authorize Requesting Device Functional Process

E Authorization request from the Mobile Request Orchestrator *(triggering Entry)*
R Verifying whether the requesting device is authorized
W Storing the timestamp of the authorization request *(when authorized)*
X Send authorization message to the Mobile Request Orchestrator
X Send error message to the Mobile Request Orchestrator
5 CFP

Mobile Transaction Data Engine

Transaction Overview Functional Process

E Transaction Data request from the Mobile Request Orchestrator *(triggering Entry)*
R Retrieving transaction data from the Data Engine
W Storing the timestamp of the latest retrieved transaction
X Send transaction data to the Mobile Request Orchestrator
X Send error message to the Mobile Request Orchestrator
5 CFP

Mobile Transaction Data Engine

Transaction Data Update Functional Process

E Transaction data update file from the Data Service layer *(triggering Entry)*
W Storing transaction data to the Data Engine
W Storing the timestamp of the latest retrieved transaction

X Send completion message (error or OK) to the Data Service Layer
4 CFP

Private Accounts Administration

Transaction Data Extract Copy Functional Process

E Completion message from the Business Rules layer *(triggering Entry)*
R Reading the latest timestamp (on OK) or the previous (on error)
R Retrieving transactions after the latest timestamp for the extract copy
W Storing the timestamp of the latest retrieved transaction
X Send transaction data extract copy to the Business Rules layer
5 CFP

The 6 CFP functional process of the Transaction overview in the App, thus makes use of four functional processes in the Business Rules layer of 23 CFP in total and one functional process in the Data Services layer of 5 CFP. Functional size measures from different layers should never be combined (COSMIC, 2010).

This example illustrates that it is important to know whether the infrastructure to supply the required data for the App is already available. The App functionality is only 6 CFP, but the infrastructure in the Business Rules layer and Data Services layer is an additional 23 and 5 CFP, respectively. This does not only increase the functional size, but different layers usually require different skills to implement the functionality.

The example works with a 3-layer architecture (figure 5b) in order to be able to meet the requirements for the transaction volume. In case a SOA architecture would have been chosen (figure 5c) the *Generate Transaction overview* functional process would have been in the Orchestration layer, the *Transaction overview* functional process in the Application layer and the *Authorize requesting device* and the *Transaction data update* functional processes in the Utility layer.

This illustrates that the choice for a layer structure does not influence the total functional size, but only the distribution over the different layers. This is not only relevant for Mobile environments, but also for cloud environments, because when you look at the complete architecture, you can see that there is a large overlap between the Mobile environment shown in the example and a basic cloud architecture (see also figure 8).

APPLICATION OF THE COSMIC METHOD FOR CLOUD ENVIRONMENTS

Cloud Computing Characteristics

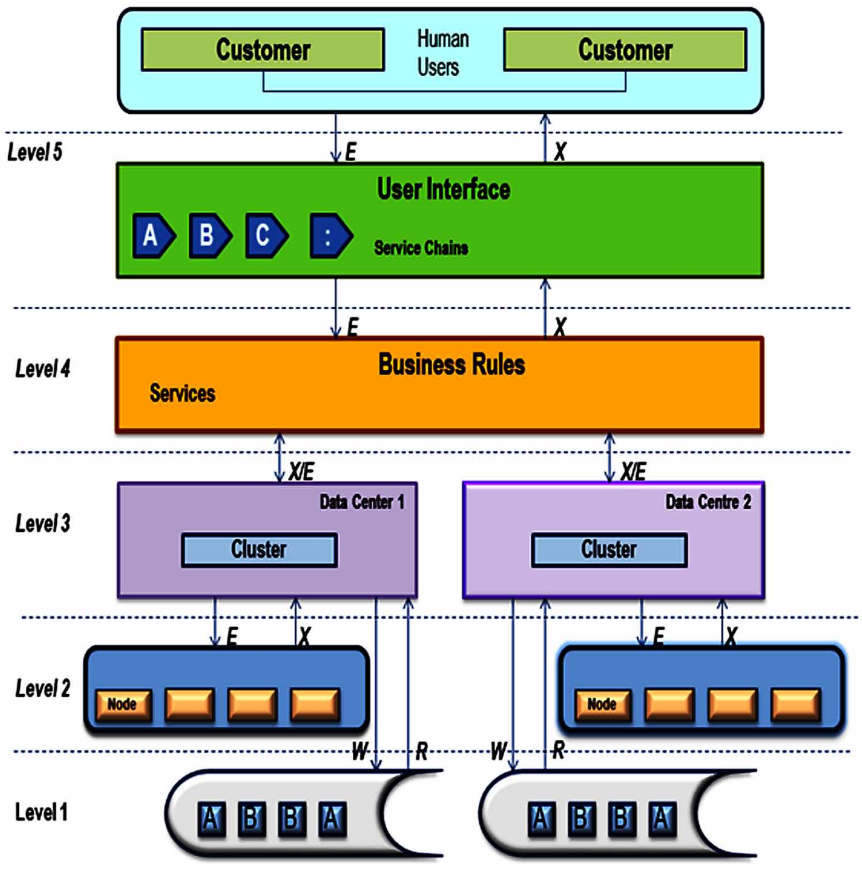
Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models. (NIST, 2011). Cloud resources are usually not only shared by multiple users but are also dynamically reallocated per demand. With cloud computing, multiple users can access a single server

to retrieve and update their data without purchasing licenses for different applications. Cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and enables IT to more rapidly adjust resources to meet fluctuating and unpredictable business demand. The present availability of high-capacity networks, low-cost computers and storage devices were the essential preconditions that have led to the emergence of cloud computing (figure 11).

The cloud environment is characterized by the following:

- High speed network access promoting use of heterogeneous platforms.
- Resource pooling building economies of scale.
- Multi-tenancy enables sharing of resources and costs across a large pool of users.
- On demand self-services provided without requiring human interaction.
- Elasticity allowing applications to scale rapidly, based on actual demand of service.
- Reliability by using multiple sites, providing business continuity and easy disaster recovery.
- Measured service enabling transparency in IT expenditure based on actual usage.
- Can be linked to large data warehouse solutions for big data applications.

Figure 11. COSMIC measurement patterns for cloud software



The computing stack of cloud software is made up of the following software paradigms:

- Virtualization, the main enabling technology to make cloud software independent of the actual hardware platforms behind.
- Service oriented architecture, to break the business processes into services that can be wired together and consumed according to the need.
- Programmable API's, to provide the ability to interact with the available cloud services, independent of the connecting device of the user (PC, tablet or mobile phone).
- Management layer, providing real-time insight in the actual use of services for management and billing.

In 2013 German researchers proposed an architecture to describe the different levels of functionality involved in cloud software (Schmietendorf, 2013). This Cloud Systems Architecture consists of five levels of functionality:

Level 1: represents the different machines (whether virtualized or not) producing a service. These machines have a priority corresponding to the criticality of the outage.

Level 2: maps the factor of clustering on a machine level while not all machines and role types are necessarily clusterable. On these both levels different severities of events and incidents (log, replication, capacity information etc.) can occur.

Level 3: consolidated the datacenter level representing the aspects of multiple redundancies on physical hosting level (such us twin or triple core datacenter approaches) where services are produced in full redundancy. Is one datacenter fully - or a service hosted there partially failing - a failover in real-time is initiated without any service and user impact. For these three levels providers set certain Operation Level Agreements to ensure their Service LevelAgreements.

Level 4: finally constitutes the Service Level over all complex alignment underneath where only a well-founded Customer Business Impact can be determined. In addition this determination will be automatically on a sound model and algorithm instead of manual interpretation.

Level 5: shows the layer of service chains which is an important level for customers if business is distributed through the cloud and different service providers for example.

From a user's point of view, there may not be much difference in functionality between on-premise traditional software and similar cloud functionality. Cloud applications function in the above ecosystem and their functionality needs to be designed and built accordingly, especially on the levels below the top level. Many functional requirements for the traditional computing models need to be examined and reformed for deployment on a cloud. Therefore, the Functional User Requirements of the cloud system can be characterized in the following general manner (Schmietendorf, 2013):

Level 1: The physical services constitute the functional processes by SOA architecture, including their Business Impact constraints (infrastructure (de-)provisioning request, withdraw of Virtual Machines, storage, network bandwidth / IO, CPU, memory)

Level 2: The clustered services constitute the functional processes by SOA architecture, including their Business Impact constraints (service (de-) provisioning and requests (CRUD (Create, Read, Update, Delete) and incident, problem, and change requests)

Level 3: The hosted services that build the functional processes by SOA architecture, including the constraints by the Operation Level Agreement to ensure their Service Level Agreements

Level 4: The provided services constitute the functional processes by SOA architecture including their Business Impact constraints (Sarbanes-Oxley Audit & Log Requests)

Level 5: The required services are the functional processes by users including their Service Level Agreement constraints of usability and kind of paying (Live, Realtime Information of fulfillment KPIs)

The following figure describes the generic COSMIC software model for the proposed Cloud Systems Architecture using the appropriate COSMIC patterns above.

Applications in the cloud range from simple personal applications, that can be accessed by mobile apps for instance, to very complex enterprise class applications with users across the globe. Applications often combine real-time and business applications data as generated by internet of people, things and machines thus requiring engineering methods to scope and model the functional data across domains and devices in a standardized manner.

Many types of cloud applications evolve very rapidly with the continuous discovery of new use cases based on usage patterns/feedback by functional users or often through social media across the globe. This requires scientific methods that can quickly map the scope and measure the changes in a consistent and practical manner which is acceptable to and understandable for all stakeholders, from customer super users and management sponsors to technical software engineering staff. This is where a functional size Measurement method as the COSMIC method can play a vital role.

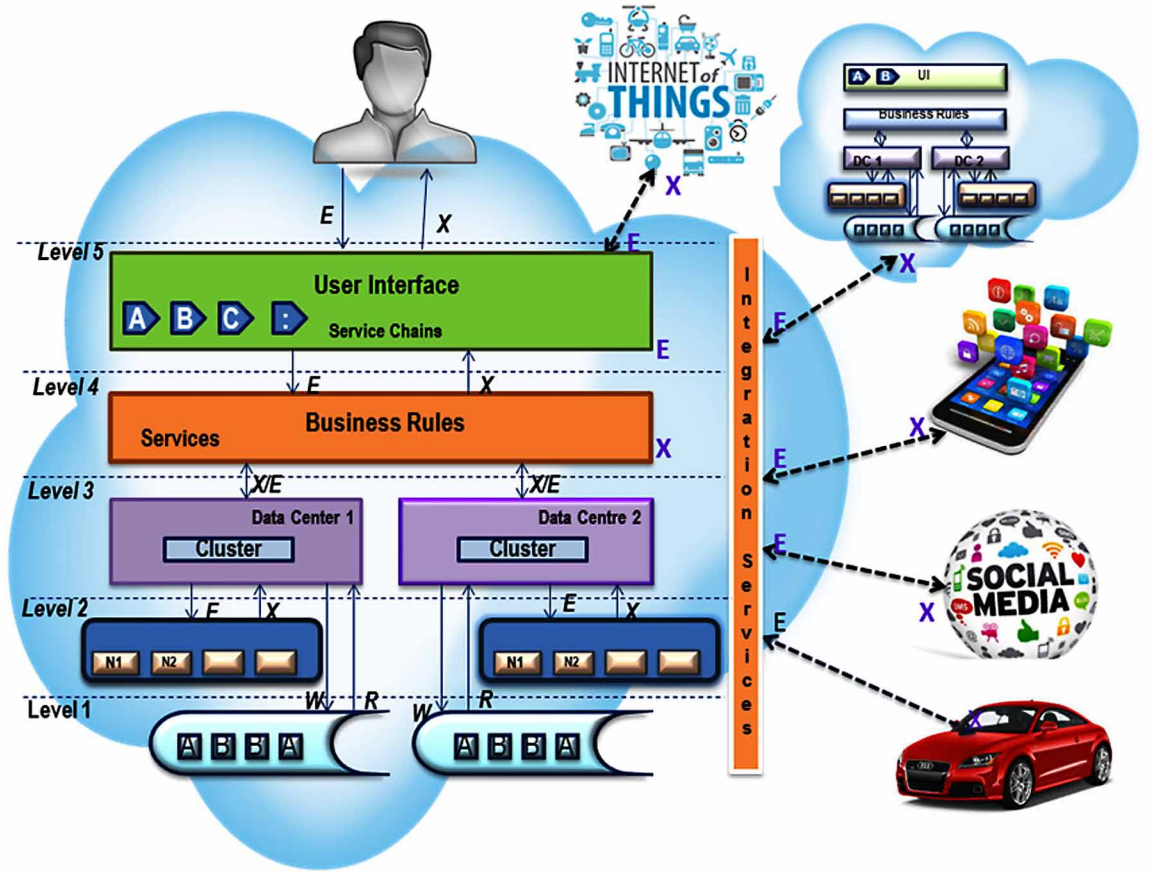
Figure 12 depicts a typical cloud application and its SOA ecosystem on the Internet. The COSMIC data movements E – Entry, R – Read, W – Write and X – eXit that contribute to "FUR" in different layers and interfaces are clearly identifiable using the Measurement Strategy and COSMIC Generic Software Model. It is important to recognize that these "data movements" are functional in nature and visible to the functional user in that scope of sizing.

Cloud Architecture Context

Cloud applications use distributed service-oriented architectures with functional requirements crossing one or more application boundaries to use web services in specific ways required by different users/applications. The architecture typically involves multiple components communicating with each other over a loose coupling mechanism such as a messaging queue. Because of the elastic provision, the architecture must contain intelligence in the coupling mechanisms. Theyalso use different data exchange formats across different layers of the solution. Software components process different data types and data entities at different layers. Multi-tenancy brings in software requirements that are unique to cloud-solutions.

Often cloud application development teams consist of multiple distributed sub-teams focusing on some layer, component or specific service. Productivity of different teams working on different layers/component using different technologies will vary and thus require estimation of efforts at any layer/level of components. Quick reliable estimation of small enhancements to very big enhancements across distributed service oriented architecture should be possible.

Figure 12. Context of cloud applications



Applying COSMIC in Cloud Usage Contexts

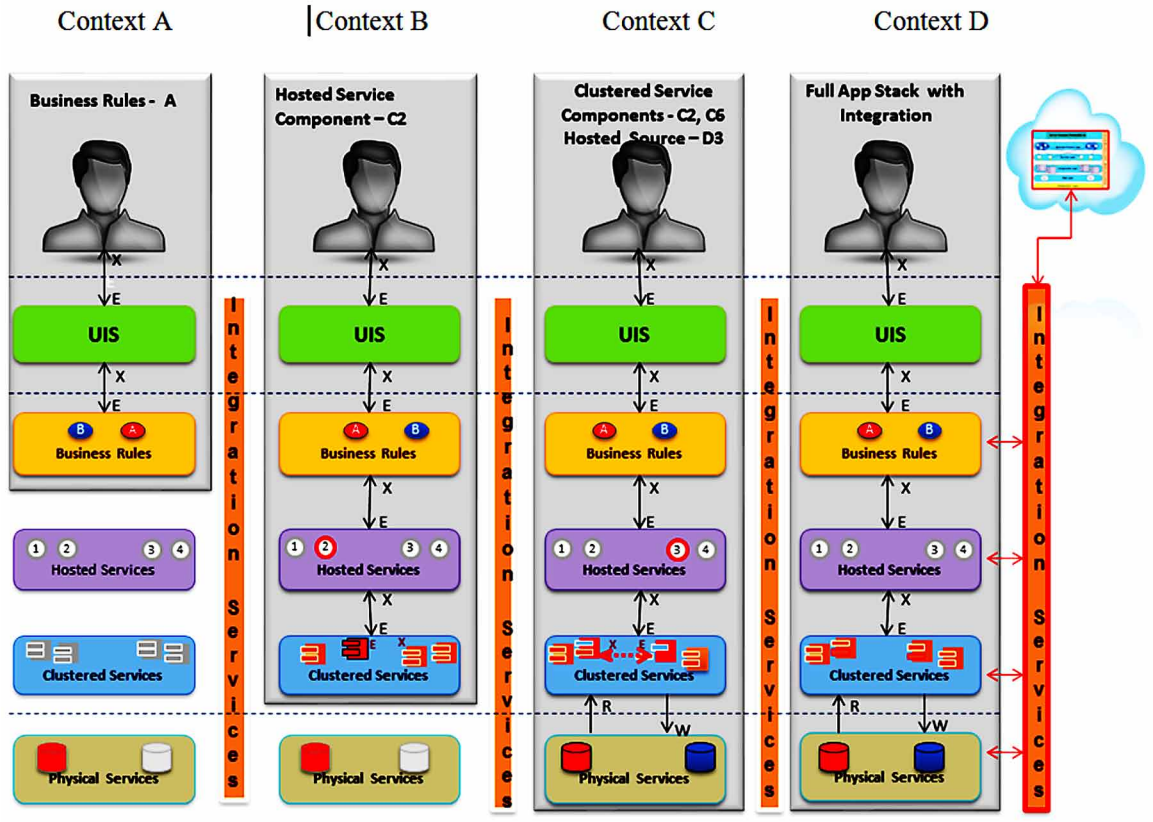
There can be multiple scenarios that can be visualized for cloud applications as illustrated in figure 13 and corresponding data movements can be measured using COSMIC for the purpose of sizing.

Context A: UR within the scope of business process A. This particular interaction does not require any functionality from any other layer or component. Though it may have its mini data-store with work-flow rules, parameters and settings.

Context B: FUR generated at the application user level involves implementing functional changes to hosted service "2" and clustered service CS 2 This enables the measurer(s) to clearly scope out the sizing instance applicable to their functional changes at their level in the cloud stack, and draw interaction boundaries to identify triggering events and functional users at different levels.

Context C: FUR includes intra component data-movements between CS2 and CS3 as well as the related Business Process B, Hosted Service 3 and Data Component D5. This illustrates the application of COSMIC to horizontal movements between components (on two nodes) within the same Cluster

Figure 13. Example of data movements in cloud software



CS. These cluster service components may be developed by different teams using different technologies at the same level of abstraction.

Context D: FUR involves the user using an external cloud application thru the Integration Service component I6 and the related data movements of the full application stack where each layer interacts independently with the integration layer. This illustrates the application of COSMIC in various hybrid-cloud Enterprise applications integration scenarios.

ESTIMATION OF COST, EFFORT AND SCHEDULE

As already mentioned, the estimation of cost, effort and schedule is a very important aspect in commercial software development projects, since it is used to allocate resources, plan product releases and negotiate payments between suppliers and contractors. In software development projects, effort is the most dominant parameter for determining cost and schedule. Therefore, estimation of cost, effort and schedule are very closely linked. The most significant input for the estimation of effort, and thus cost and schedule, is the size of the software to be developed.

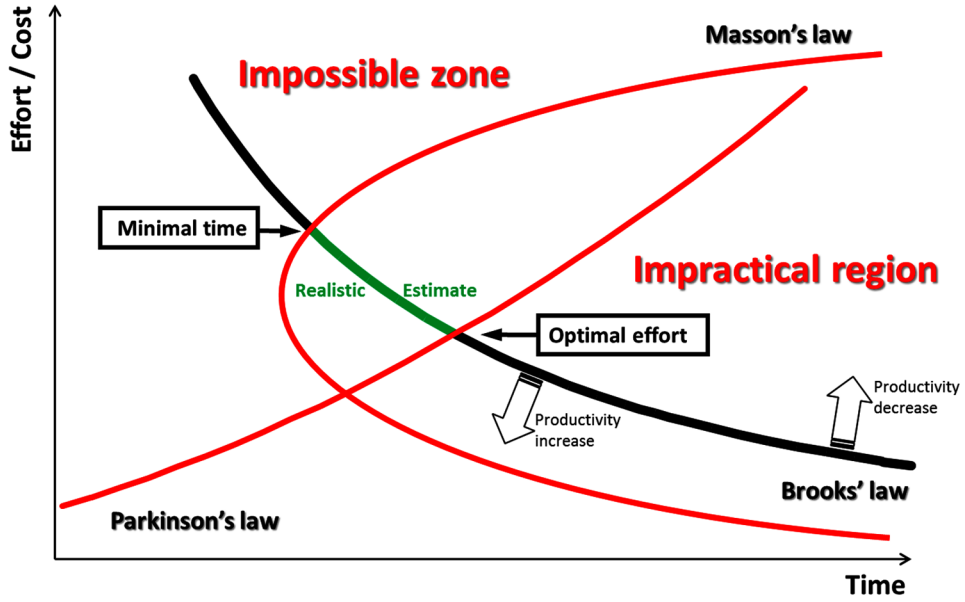
The relation between size and effort is the productivity. Productivity is the number of size units that can be produced per unit of time, for example CFP/person day. Productivity is dependent on a number

of factors, like the programming language, development platform, team experience and organization maturity. The factors that determine or influence productivity have been extensively studied and have led to the COCOMO family of models that predict productivity in various areas of software engineering (Boehm, 1981). Most of today's estimating tools use some descendent of the COCOMO family. Because of the more convenient numbers often the inverse of productivity, product delivery rate, is used to report on productivity figures. The most common unit for product delivery rate is hr/CFP. Note that a higher number of hours per COSMIC function point means worse productivity. For a given productivity, the relationship between size, effort and schedule is governed by Brooks' law (Brooks, 1995). Brooks was the first to describe that productivity was not a fixed figure, but in fact a function of time pressure. Brooks' law is therefore also known as the time/effort trade-off.

In figure 14, this trade-off is depicted from the top left, to the bottom right (Ross, 2005). Each point on the line represents a situation where all factors determining and influencing productivity are the same, except for the required schedule (Putnam & Myers, 2003). When the productivity increases, the trade-off will shift towards the lower left corner. When the productivity decreases, the trade-off will shift towards the upper right corner. Brooks' law explains that projects that are put under time pressure (moving to the left in the figure) require more effort, and hence cost more. This effect is caused by the fact that activities that ideally are executed by the same person in sequence now must be done by different people in parallel. This increases the amount of people needed to staff the project and increases the time to coordinate the parallel activities (Galorath & Evans, 2006).

Brooks did not define any boundaries to this effect. In theory that would mean that when we can wait long enough, projects would require almost no effort. In reverse, when we have enough resources, projects could be finished in a day. Paul Masson developed a function that determined for every project the minimum time in which a project can be finished (Jensen, 2006). The point where the Masson function intersects with Brooks' law is the minimal time in which a project of a given size with a given

Figure 14. Relations governing parametric estimation of cost, effort and schedule



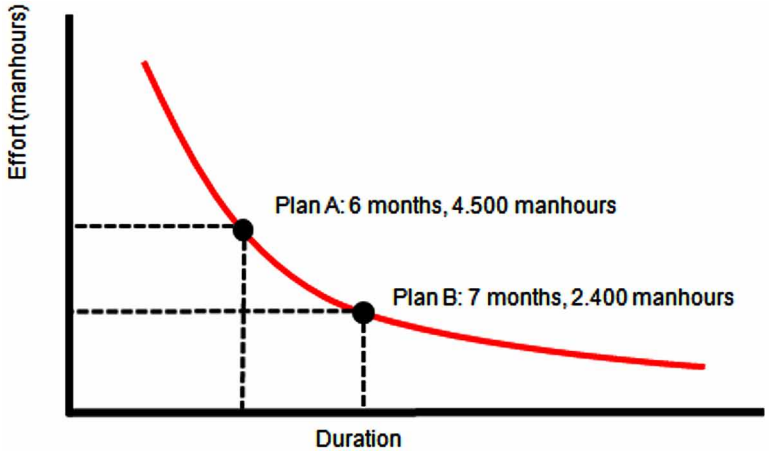
productivity setting can be realized. The area outside the Masson function is known as the impossible zone. This is consistent with the analysis done on 6,300 projects a few years earlier, where no project was found that was finished before the minimum development time (Putnam & Myers, 2003). Finishing the project earlier than that intersection is only possible when a higher productivity can be achieved, or when the scope of the project is decreased significantly.

There is also a maximum to the economically realistic schedule for a project to finish. Beyond a certain point a project will not require less effort. This point is governed by Parkinson's law. Parkinson's law was first stated by Cyril Northcote Parkinson as a part of a humorous essay published in The Economist in 1955: *Work expands so as to fill the time available for its completion*. A few years later the essay was expanded to a full book and the law entered the world of serious economic principles (Parkinson, 1957). Creating a schedule beyond the intersection of Brooks' law and Parkinson's law will not lead to less effort for the project to complete. So the intersection of these laws is the point where the project can be realized with the economically optimal amount of effort. Any point between the intersection of Brooks' law with the laws of Paul Masson and Parkinson is considered a realistic estimate.

The effect of this trade-off was demonstrated in an experience report in 2011 for a 500 CFP Java project (van Heeringen, 2011).The productivity to build this application can be anything between 0.11 CFP/hr and 0.20 CFP/hr, depending only on the chosen duration of the project, all other variables remaining constant (figure 15).

The size of software in COSMIC FP can be used to estimate the cost, effort and schedule by building parametric models based on the above rules. Best practice for building such parametric models is to build them based on internal data of the organization (Abran, 2015). As a quick start, benchmark reports based on COSMIC FP from the International Software Benchmarking and Standards Group can be used to populate the first version of an estimating model (ISBSG, 2012). COSMIC measures have been successfully used with estimation models based on Regression Analysis, Estimation by Analogy, COCOMO and Putnam Norden Rayleigh. COSMIC size based estimation models can be used to estimate efforts for full life cycle of software development or any major life cycle activities such as Testing (Kamala Ramasubramani & Abran, 2013).

Figure 15. Time-effort trade-off for a 500 CFP Java project



COSMIC can also be used in software development projects using Agile approaches (COSMIC, 2011). In 2014 PhD research in the remote surveillance industry showed that the inaccuracy of effort estimation for Agile projects could be reduced from 58%, based on the Planning Poker technique, to 16.5%, based on COSMIC (Commeyne, 2014). Similar results, although more qualitative in nature, have been reported by an Australian risk broker (Dekkers, 2014).

The most important application of the COSMIC method is (supplier) cost management for software development and maintenance. The COSMIC organization maintains a list of known users of the method (COSMIC, 2015b). Most of the companies on that list use the method for cost management. Only a few of them have reported their success in the use of the COSMIC method in detail, for example in the financial (Vogelezang & Lesterhuis, 2003) and automotive industry (Stern, 2010 & Oriou, 2014).

Another application of the COSMIC method is the estimation of software code size. This application is particularly useful for embedded mobile applications that must be fit on ROM memory into products with high cost pressure, like in-car software systems and electronic control units (Lind & Heldal, 2011). Researchers from the Swedish car industry showed that it is possible to estimate the code size within 15% accuracy. This makes it possible to implement the software into the smallest possible memory product.

FUTURE RESEARCH DIRECTIONS

The COSMIC method is getting wider recognition and its usage for various environments is evolving. The original basic principles of the COSMIC method have remained unchanged since they were first published in 1999. Due to various refinements, clarifications and additions the method has now progressed up to version 4.0.1 (COSMIC, 2015a). The documentation structure of the method is such that guides are issued to provide special guidance for specific topics. The guidance that may be needed to apply the COSMIC method on mobile and cloud software is currently addressed in guides on service oriented architecture, real-time software and agile software development. Currently a number of case studies are under development from the mobile community.

One of the subjects that requires further research is early size estimation. Especially mobile and cloud software is evolving rapidly. Estimates of cost, effort and schedule are needed faster and earlier and thus it must be possible to produce the underlying size estimates based on global specifications. A number of approximation techniques have already been developed and a guide how to apply them is expected in 2015. Some of these techniques need to be fully tested in practice, some have not been tested yet in the mobile and cloud domain.

Another major subject, not only in mobile and cloud software, but in software development and maintenance in general, is the impact of non-functional requirements. Experience of IBM's Event-driven Decision & Smart Mobile Platforms department has shown that non-functional requirements in spatio-temporal aware services represent more than 50% of the effort to produce services (IBM Research, 2014). Late 2015, a joint initiative of COSMIC and IFPUG led to a classification of different types of non-functional requirements and project constraints that can be used to estimate Cost, Effort and Schedule (COSMIC & IFPUG, 2015).

CONCLUSION

The COSMIC method has proven its applicability in mobile environments and is gaining acceptance for estimating software development in cloud environments. The method has been accepted as International Standard, recognized as a technology independent measure for the functional size of software. The COSMIC method can be used by procurement and contract managers to select a supplier and control supplier performance. COSMIC can be used by investors to decide on the cost/benefit analysis for new projects and for valuing software assets. The COSMIC method is an independent size measure that can be used by Project Managers for estimating, project scheduling and controlling scope creep in projects. Response from the users of COSMIC reveal that requirements become unambiguous, traceable and testable when COSMIC is used for sizing, resulting in improved customer satisfaction.

Major Advantages of the COSMIC Method Are:

- Early size approximation, as a basis for the estimation of cost, effort and schedule early in the development lifecycle.
- Applicable in layered and distributed architecture.
- Applicable to different types of software (components) in different architectural layers.
- Suitable for different types of applications (transactional, real-time, data warehouse).
- Technology and platform independent.
- Applicable for linear, iterative and agile software development methodologies.
- Open standard, conformant to the ISO meta standard for functional size measurement.
- Supported by a world-wide community of volunteers.

ABOUT COSMIC

COSMIC is a voluntary, not-for-profit organization of software metrics experts, representing both industry practitioners and academics, from over twenty countries around the world. COSMIC was founded in 1998. All its publications are 'open' and available for free distribution subject to copyright and acknowledgement restrictions.

The principles of the COSMIC method were laid down in 1999 after which field trials were conducted during 2000 – 2001 with several international companies and academic institutions before announcing the method. ISO adopted COSMIC as an International Standard and announced it as ISO/IEC 19761 in 2003. Currently the Measurement Manual v4.0.1 serves as the reference for COSMIC Measurement (COSMIC, 2015a). A number of additional documents to support the method have been published and can be downloaded free of charge (COSMIC, 2014).

REFERENCES

Abran, A. (2015). *Software Project Estimating: The fundamentals for providing high quality information to decision makers*. Hoboken, NJ: John Wiley & Sons.

Abran, A., Symons, C. R., & Oligny, S. (2001). An overview of COSMIC field trial results. In *Proceedings of the 12th European Software Control and Metrics conference*.

Boehm, B. W. (1981). *Software Engineering Economics*. Upper Saddle River, NJ: Prentice Hall PTR.

Brooks, F. P. Jr. (1995). *The Mythical Man-Month*. Boston, MA: Addison-Wesley Longman Publishing.

Commeyne, C. (2014). *Établissement d'un modèle d'estimation a posteriori de projets de maintenance de logiciels.* (PhD thesis). École de Technologie Supérieure, Montréal, Canada.

Common Software Measurement International Consortium. (2010). *Guideline for Sizing Service Oriented Architecture Software*. Montréal, Canada.

Common Software Measurement International Consortium. (2011). *Guideline for the use of COSMIC FSM to manage Agile projects*. Montréal, Canada.

Common Software Measurement International Consortium. (2012). *Guideline for Sizing Real-time Software*. Montréal, Canada.

Common Software Measurement International Consortium. (2014). *Publications*. Retrieved May 20, 2015, from http://cosmic-sizing.org/publications

Common Software Measurement International Consortium. (2015a). Measurement Manual (version 4.0.1). Montréal, Canada.

Common Software Measurement International Consortium. (2015b). *Usage of the COSMIC method.* Retrieved May 20, 2015, from http://cosmic-sizing.org/cosmic/usage/

Common Software Measurement International Consortium & International Function Point User Group. (2015). *Glossary of terms for Non-Functional Requirements and Project Requirements used in software project performance measurement, benchmarking and estimating*. Montréal, Canada: Author.

Dekkers, C. (2014). *Function Points and Agile Development... Considerations and Opportunities.* Retrieved May 20, 2015, from https://www.linkedin.com/grp/post/2758144-5904564005854285828

Galorath, D. D., & Evans, M. W. (2006). *Software Sizing, Estimation, and Risk Management*. Boca Raton, FL: Auerbach Publications. doi:10.1201/9781420013122

IBM Research. (2014). *SaS hub non-functional requirements*. Retrieved May 20, 2015, from https://www.research.ibm.com/haifa/projects/software/nfr/index.html

Institute of Electrical and Electronics Engineers Computer Society. (2014). *Guide to the Software Engineering Body of Knowledge (version 3.0)*. Washington, DC: IEEE.

International Organization for Standardization. (2007). *Information Technology – Software Measurement – Functional Size Measurement. International Standard ISO/IEC 14143:2007*. Geneva, Switzerland: ISO.

International Organization for Standardization. (2011). *Software Engineering – COSMIC – A Functional Size Measurement Method. International Standard ISO/IEC 19761:2011*. Geneva, Switzerland: ISO.

International Software Benchmarking and Standards Group. (2012). *The Performance of Real-time, Business and Component Software Projects – An analysis of COSMIC measured projects in the ISBSG database*. Balaclava, Australia: ISBSG.

Janssen, C. (2014). *Three-Tier Architecture.* Retrieved May 20, 2015, from http://www.techopedia.com/definition/24649/three-tier-architecture

Jensen, R. W., Putnam, L. H. Sr, & Roetzheim, W. (2006). Software estimating models: Three viewpoints. *Crosstalk*, *2*, 23–29.

Jones, C. (2013). *A short history of the Lines of Codes (LoC) metric*. Retrieved May 20, 2015, from http://namcookanalytics.com/a-short-history-of-the-lines-of-code-loc-metric

Kamala Ramasubramani, J., & Abran, A. (2013). A survey of Software Test Estimation Techniques. *Journal of Software Engineering and Applications*, *6*(10), 47–52. doi:10.4236/jsea.2013.610A006

Lind, K., & Heldal, R. (2011). A model-based and automated approach to size estimation of embedded software components. In *Proceedings of the 14th international conference on Model Driven Engineering Languages and Systems (MODELS)*. Wellington, New Zealand: Springer Verlag. doi:10.1007/978-3-642-24485-8_24

National Institute of Standards and Technology. (2011). *The NIST Definition of CloudComputing*. Retrieved May 20, 2015 from http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

Oriou, A., Bronca, E., Bouzid, B., Guetta, O., & Guillard, K. (2014). Manage the automotive embedded software development cost & productivity with the automation of a functional size measurement method (COSMIC). In *Proceedings of the Joint Conference of the 24th International Workshop on Software Measurement (IWSM) and the 9th International Conference on Software Process and Product Measurement (Mensura)*. Rotterdam, The Netherlands: IEEE Computer Society Conference Publishing Services.

Parkinson, C. N. (1957). Parkinson's law, or the pursuit of progress. Cutchogue, NY: Buccaneer Books.

Putnam, L. H., & Myers, W. (2003). *Five Core Metrics: The intelligence behind successful software management*. New York, NY: Dorset House Publishing.

Ross, M. A. (2005). *Parametric project monitoring and control: performance-based progress assessment and prediction*. Retrieved May 20, 2015, from http://www.dtic.mil/ndia/2005cmmi/wednesday/ross3.pdf

Schmietendorf, A., Fiegler, A., Neumann, R., Wille, C., & Dumke, R. R. (2013). COSMIC Functional Size Measurement of Cloud Systems. In *Proceedings of the Joint Conference of the 23rd International Workshop on Software Measurement (IWSM) and the 8th International Conference on Software Process and Product Measurement (Mensura)*. Ankara, Turkey: IEEE Computer Society Conference Publishing Services. doi:10.1109/IWSM-Mensura.2013.15

Stern, S., & Guetta, O. (2010). Manage the automotive embedded software development cost by using a Functional Size Measurement Method (COSMIC).In *Proceedings of the 4thEmbedded Real Time Software and Systems conference*. Toulouse, France: ERTS2.

van Heeringen, H. S. (2011). Estimate faster, cheaper... and better! In *Proceedings of the 8th Software Measurement European Forum*. Rome, Italy.

van Heeringen, H. S., & van Gorp, E. W. M. (2014). Measure the functional size of a mobile app using the COSMIC functional size measurement method. In *Proceedings of the Joint Conference of the 24th International Workshop on Software Measurement (IWSM) and the 9th International Conference on Software Process and Product Measurement (Mensura)*. Rotterdam, The Netherlands: IEEE Computer Society Conference Publishing Services. doi:10.1109/IWSM.Mensura.2014.8

Vogelezang, F. W. (2013a). *The first generation of Functional Size Measurement*. Retrieved May 20, 2015, from http://thePriceofIT.blogspot.nl/2013/01/the-first-generation-FSM.html

Vogelezang, F. W. (2013b). *What is a second generation FSM method*. Retrieved May 20, 2015 from http://thepriceofit.blogspot.nl/2013/02/second-generation-FSM.html

Vogelezang, F. W., & Lesterhuis, A. (2003). Applicability of COSMIC in an administrative environment - Experiences of an early adopter. In *Proceedings of the 13th International Workshop on Software Measurement*. Montréal, Canada: Shaker Verlag.

ADDITIONAL READING

Abran, A. (2015). *Software Project Estimating: The fundamentals for providing high quality information to decision makers*. Hoboken, NJ: John Wiley & Sons.

Dumke, R. R., & Abran, A. (2011). *COSMIC function points: theory and advanced practices*. Boca Raton, FL: Auerbach Publications.

Dumke, R. R., Schmietendorf, A., Seufert, M., & Wille, C. (2014). *Handbuch der Software Umfangsmessung und Aufwandschätzung*. Berlin, Germany: Logos Verlag.

Symons, C. R., & Lesterhuis, A. (2014). *Introduction to the COSMIC method of measuring software*. Montréal, Canada: COSMIC.

KEY TERMS AND DEFINITIONS

Data Movement: A base functional component which moves a single data group type. There are four types of data movement: Entry, Exit, Read and Write (COSMIC, 2015a).

Entry: An Entry is a data movement that moves a data group from a functional user across the boundary into the functional process where it is required. An Entry is considered to include certain associated data manipulations.

Exit: An Exit is a data movement that moves a data group from a functional process across the boundary to the functional user that requires it. An Exit is considered to include certain associated data manipulations.

Functional Process: A set of data movements representing an elementary part of the Functional User Requirements for the software being measured, that is unique within that Functional User Requirements and that can be defined independently of any other functional process in that Functional User Requirements. A functional process may have only one triggering Entry. Each functional process starts processing

on receipt of a data group moved by the triggering Entry data movement of the functional process. The set of all data movements of a functional process is the set that is needed to meet its Functional User Requirements for all the possible responses to its triggering Entry (COSMIC 2015a).

Functional User Requirements: A sub-set of the user requirements. Requirements that describe what the software shall do, in terms of tasks and services. Functional User Requirements relate to but are not limited to: data transfer (for example Input customer data; Send control signal) data transformation (for example Calculate bank interest; Derive average temperature) data storage (for example Store customer order; Record ambient temperature over time) data retrieval (for example List current employees; Retrieve latest aircraft position) Examples of user requirements that are not Functional User Requirements include but are not limited to: quality constraints (for example usability, reliability, efficiency and portability), organizational constraints (for example target hardware and compliance to standards), environmental constraints (for example interoperability, security, privacy and safety), implementation constraints (for example development language, delivery schedule).

Layer: A functional partition of software system architecture. Software in one layer provides a set of services that is cohesive according to some defined criterion, and that software in other layers can utilize without knowing how those services are implemented. The relationship between software in any two layers is defined by a 'correspondence rule' which may be either, 'hierarchical', i.e. software in layer A is allowed to use the services provided by software in layer B but not vice versa (where the hierarchical relationship may be up, down or sideways), or 'bi-directional', i.e. software in layer A is allowed to use software in layer B, and vice versa. Software in one layer exchanges data groups with software in another layer via their respective functional processes. Software in one layer does not necessarily use all the functional services supplied by software in another layer. One layer may be partitioned into other layers according to different defined software architecture (COSMIC, 2015a).

Non-Functional Requirements: Any requirement for or constraint on a hardware/software system or software product, or on a project to develop or maintain such a system or product, except a functional user requirement for software. Note that system or software requirements that are initially expressed as non-functional often evolve as a project progresses wholly or partly into Functional User Requirements for software (COSMIC, 2015a).

Persistent Storage: Storage which enables a functional process to store a data group beyond the life of the functional process and/or from which a functional process can retrieve a data group stored by another functional process, or stored by an earlier occurrence of the same functional process, or stored by some other process (COSMIC, 2015a).

Read: A Read is a data movement that moves a data group from persistent storage into the functional process which requires it. A Read is considered to include certain associated data manipulation.

Triggering Event: An event, recognized in the Functional User Requirements of the software being measured, that causes one or more functional users of the software to generate one or more data groups, each of which will subsequently be moved by a triggering Entry. A triggering event cannot be sub-divided and has either happened or not happened. Note that clock and timing events can be triggering events (COSMIC, 2015a).

User (Functional User): A (type of) user that is a sender and/or an intended recipient of data in the Functional User Requirements of a piece of software. The functional users of a piece of software to be measured shall be derived from the purpose of the measurement. When the purpose of a measurement of a piece of software is related to the effort to develop or modify the piece of software, then the functional users should be all the senders and/or intended recipients of data to/from the new or modified functionality, as required by its Functional User Requirements (COSMIC, 2015a).

Write: A Write is a data movement that moves a data group lying inside a functional process to persistent storage. A Write is considered to include certain associated data manipulation.