Development of COSMIC Scaling Factors Using Classification of Functional Requirements

Alain Abran[1], Shaghayegh Vedadi[1]

[1] École de Technologie Supérieure – ETS, University of Quebec (Montréal, Canada)
alain.abran@etsmtl.ca
shaghayegh.vedadi-moghaddam.1@ens.etsmtl.ca

Abstract. The focus of this paper is estimating COSMIC function points early in the software development lifecycle. The main input to function point sizing is the set of functional requirements for a piece of software. However, very early in the lifecycle it is unrealistic to expect this set of requirements to describe the full scope of functionality, including all the necessary functional details. The application of the size scaling factors developed within this context is illustrated with two COSMIC case studies. While the scaling factors are specific to the case studies used, the approximation technique presented can be used in most organizations provided that data on past projects can be collected and relevant classifications of functionalities identified. For the purpose of this paper, the ISO 19761 COSMIC function points standard is taken as reference for discussion, while the majority of concepts presented are generic to other similar ISO standards.

Keywords: Functional size measurement, COSMIC, size estimation, functional requirements quality, software estimation, ISO 19761

1 Introduction

Function point sizing (FPS) quantifies the functional size of software and is used for various purposes in software project management, including effort estimation, project planning, project monitoring, productivity studies and benchmarking [1, 2]. Compared to lines-of-code based measures, FPS stands out among software size metrics as it is based on the requirements themselves, which as soon as they become available, prior to any coding, can provide size information in the earlier stages of the software development life cycle (SDLC). This makes FPS a tool of choice for planning techniques that require an early view of the software to be developed. Despite being available earlier than other sizing methods, a precise application of FPS requires that the functional requirements of the software be detailed, and the architecture defined [3]. More often than not, this point in the lifecycle comes relatively late for project estimation needs. Therefore, several size estimation techniques have been proposed that can be used for these early management activities.

The main input of FPS is the set of functional requirements for a piece of software. However, very early in the SDLC it is unrealistic to expect this set of requirements to describe the full scope of functionality of the software as a whole, including all the

necessary functional details. There are obviously many functional unknowns early in the life cycle. Therefore, the size one can measure directly from a set of incomplete functional requirements and the size measured from a very detailed set of functional requirements fully developed and implemented may differ considerably. This can be well illustrated with the uncertainty curve of Boehm [4] where the uncertainty progressively decreases as the project progresses and additional more precise information becomes available where:

- at time = project closure, everything is known about the software. When all of the requirements information as implemented in the software code is available, the size can be measured with high accuracy;
- at time = feasibility study, the information available on the requirements is typically high level and without much detail. From a functional perspective many unknowns remain. With imprecise and incomplete inputs, there cannot be accurate measurement. Rather, the expected functional size at project closure can only be estimated and, as with any type of estimate, this comes with a range of uncertainty that will vary depending on the quality, completeness and stability of the set of requirements [5];
- at t = between feasibility and project closure, the completeness of the information and requirements will progressively improve, which will impact the measurement results until all requirements have been specified in detail.

Measurers and software engineers need a clear understanding of the level of information available at the time of measurement and how it impacts any measurement or estimation of size. More often than not, this point in the lifecycle is late for project estimation needs of the organization. Therefore, several size estimation techniques have been proposed that can be used as an input for early management activities, mostly based on statistical analyses [6-10]. However, these do not provide insight into the sources of the gaps, their timing and how such insights can help improve size estimation.

This paper focuses on estimating and approximating function points early in the lifecycle. Discussion of effort estimation and the relationship between functional size and effort estimation is out of scope for this study.

An exploratory analytical study by Abran et al. [6] identified the nature of the gaps between earlier sizing and final/delivered size, as well as the sources of these gaps. In this paper, we complement the analytical study using ISO-IEEE 21948 on requirements engineering [7] and report on empirical research carried out with two case studies of the COSMIC group, documenting both requirements at various levels of detail as well as the corresponding COSMIC function points measured on the basis of the most detailed requirements. From these observations, and comparison of the information available at various points in time, size scaling factors specific to these case studies were developed and discussed.

This paper is structured as follows. Section 2 presents related work. Section 3 presents a number of key requirement concepts and types discussed in ISO-IEEE 21948 relevant for the development of scaling factors across the lifecycle phases. Application of these concepts to two COSMIC case studies, including the relevant scaling factors

developed within this context are presented in section 4. Section 5 summarizes the work and offers suggestions for future work.

2 Related Work

Some COSMIC-based size estimation techniques have been proposed that can be used as an input for early management activities [8-14]. For instance, the COSMIC 'Guideline for Early or Rapid COSMIC Functional Size Measurement by using approximation approaches' [13] proposed the following seven approximation techniques, based mostly on statistical analyses that can be tailored by organizations collecting their own data from completed projects. This COSMIC guideline also mentions some emerging approaches based, for example, on informal text, fuzzy logic, etc. Most of these techniques are based on quantitative analysis of data at project completion and the development of indicators at that point in the lifecycle. However, these techniques do not refer back to the lifecycle to identify earlier missing information or the corresponding sources.

The analysis by Abran et al. [6] looked at the sources of the gap between the initial visible size of the functionality documented at the initiation of a project and the final size at project closure. It identified several factors that may lead the final functionality delivered by the software at project end to be different from that defined at the time the initial requirements were created. Such factors increase the amount of functionality in the software resulting in a gap between the initial and final size, so that as the project progresses it is expected that:

- initial requirements will be detailed progressively (added functional details to a functionality already identified);
- some initially undocumented functionality will be approved and documented;
- additional functionality will be captured as additional requirements;
- some functionality will be removed and/or changed.

So, as measurements are performed at further points in the lifecycle, the visible size will approach the final size. The study by Poulin et al. [15] illustrates how implied and/or hidden functionality can cause a large gap between the initial and final functional size of a piece of software. The set of requirements used as the input for the initial COSMIC measurement was one of the standard case studies published by COSMIC. Requirements were of high quality and did not include vague or incomplete requirements. For the selected set of requirements in the example, the final size turned out to be 236% greater than the size as measured in the initiation phase, due to security functionality that was documented as high-level system non-functional requirements at the time of the initial measurement and that later was allocated to the software as additional functionality [15].

To tackle these issues, an organizational repository-based approach was proposed [6] to record size data information related to the initial estimated size (e.g. t = project initiation phase) and size at the time of delivery (e.g. t = project closure) that would allow users to:

- for each piece of software, compare the initial requirements and the functionality at the time of delivery;
- classify additional size (hidden functionality, undocumented functionality, additional/modified functionality);
- identify the reasons for the size change in each dimension;
- use customized techniques for different dimensions of size change to adjust initial measurement results and better estimate the final size of software.

However, this proposed approach could not be evaluated at that time with case studies. The research presented here addresses two of the related issues:

- development of a standard-based approach for the identification of missing software functionalities at various levels of detail, and
- illustrating their application with the documented high-level as well as detailed requirements and corresponding COSMIC function points size of two publicly available COSMIC case studies.

3 Sources, types & timing of requirements in ISO-IEEE29148

The ISO-IEEE standard 29148 on requirements engineering presents a number of concepts related to the sources, types and levels of detail of the requirements throughout the system and software life cycle.

The initial set of requirements originates from two sets of sources, the business stakeholders and other stakeholders, which leads to the 'systems' requirements. From the system functional requirements, some will be allocated to software requirements (as well as to hardware requirements and at times to manual operational procedures). These sources provide the system contextual requirements, including the system purpose, system scope and system overview. From this contextual information, the following are then identified:

- system functional requirements,
- system non-functional quality

ISO-IEEE 29148 also notes that in addition to software functions explicitly identified, there may be interfaces identified, but not yet specified, as well as quality requirements, still at a high level.

At various points later in the lifecycle, these lists of software functions are detailed, programmed, tested and implemented; the same applies to system non-functional quality requirements, some of which may later be allocated to additional software functional requirements [16-20]. It is to be noted that a large number of these system non-functional quality requirements may be derived from stakeholders not identified upfront in the feasibility studies but who must be involved at some point in the operationalization and ongoing operation of the software developed.

At the end of development all of these functional details are known to the developers, even if not formally documented, and if the COSMIC function point rules are known, the software functionality implemented can be measured precisely. An outside measurer without access to all the documentation, or the undocumented functionality (such as that derived from system-NFR and implemented late in the testing phase), would

miss a number of software functions. Similarly, if measurement is done earlier in the lifecycle, a number of software functions which have been neither identified nor specified in detail would be 'invisible' to the outside early measurer.

From a measurement perspective, measurement has to be carried out throughout the lifecycle, from the situation on the left when initially only the visible, above-water components can be measured, progressively, to below the water line where the visibility increases and additional sizing can be carried out, up to complete measurement when full under-water visibility reveals the total view.

However, in software development there is no known ratio, as in physics, for the mass above to the mass under water for a floating iceberg. Across all software projects, functional visibility will vary across the development lifecycle, and hence software functional documentation across lifecycle phases will vary. Should such exist in an ideal work, there has not yet been research carried out to identify ratios across these phases.

What follows next is our reporting of the research addressing this challenge by working backwards from known detailed contexts documented and measured in a context of full visibility, full documentation and no uncertainty. This is precisely the context of the case studies documented and measured by the COSMIC group [21, 22].

These COSMIC case studies also document the software functionality at various levels of detail, prior to measurement with full functional details.

In the paper, the concepts from ISO 29148 were used, as per the iceberg analogy, to:

- identify types of requirements when they become visible, and
- develop (by working backwards) ratios to extrapolate in previous phases.

This approach is illustrated in practice with the following two COSMIC case studies:

— the course registration system (CRS) [21],
— the restaurant case study (RestoSys) [22].

4 COSMIC case studies: course registration system (CRS)

4.1 Course registration system (CRS) case study: Documentation levels

Over the years, the COSMIC group has published a number of versions of their course registration system (CRS) case study [21]. The June 2015 version was used in this research. In the documentation of the CRS case study, the description of the functionality is presented sequentially at three levels of increasing detail. We use, where relevant, the terminology from ISO-IEEE 29148:

- Level 1: (Business) functions (list of 'system' functions);
- Level 2: (Business) functions allocated to software functional processes (list of 'software functions');
- Level 3: Detailed functionality allocated to each software functional process (functional details allocated to software).

Level 1. Business functions = 'system' functions. The information at this level – Table 1 – is available at 'vision' time – or feasibility:

Level 2. Software functions. Next, a detailed list of functions is allocated to the software – often this is available early in the specification phase – Table 2.

Level 3. Detailed functionality for each functional process allocated to software. The CRS case study presents a level of detail that is ideally complete and verified at the end of the specification phase (as well as at the end of a project, provided that the functional documentation has been kept up to date across the lifecycle).

This is the point at which all of the functional details are available, and a detailed measurement of functional size can be done accurately and completely. However, in practice, a large number of these functional details are documented much later, and sometimes not at all.

Table 1. CRS case study: list of business functions (N=7)

NO	Business function
1	Maintain professor information (by the registrar)
2	Maintain student information (by the registrar)
3	Maintain courses to teach (by professor)
4	Maintain student schedule (by students)
5	Close registration (by the registrar)
6	Submit grades (by professor)
7	Enquire report card (by students)

Table 2. CRS case study: list of functional processes (N=21)

NO	Functional process
1	Add a professor
2	Modify a professor
3	Delete professor
4	Enquire on a professor
5	Add a student
6	Modify a student
7	Delete a student
8	Enquire on student
9	Add courses to teach
10	Modify a course to teach
11	Delete courses to teach
12	Enquire courses to teach
13	Enquire on course to teach details
14	Add courses
15	Modify a course
16	Delete a course
17	Enquire on courses
18	Enquire on course details
19	Close registration
20	Submit grades
21	Enquire on a report card

4.2 CRS case study: functional size at level 3

In this case study, the COSMIC measurement is carried out and presented at the most detailed level 3 for one of the 21 functional processes – see Table 3. In Table 3, the

'DM type' column refers to the four COSMIC types of data movements (DM): E for entry, X for exit, R for read and W for write data movements.

Overall, in this CRS case study, there were 21 functional processes with a total size 103 CFP, with the following descriptive statistics:

- minimum size of a functional process is 3 CFP and maximum size is 8 CFP
- mean size is 5 CFP
- median size is 4.85 CFP
- standard deviation 1.5 CFP.

Table 3. List of COSMIC data movements – Functional Processes (N=2)

FP NO/ Req.	Process name	Functional user/object of interest	Sub-process description	Data group	DM type	CFP	∑
1/ 1.2.1	Add a profes-sor	Registrar/ professor	Registrar enters information for the Professor	Profes-sor data	E	1	
			The system validates the entered data and checks if a professor of the same name exists already	Profes-sor data	R	1	
			The system creates a new professor	Profes-sor data	W	1	
		Registrar/ professor	Display the system generated pro-fessor ID number	Profes-sor ID	X	1	
		Registrar/ messages	Display error message	Mes-sages	X	1	
							5

4.3 CRS case study: sources of software functionality using ISO-IEEE 29148

The sources of these functional details were analyzed using the concepts from ISO-IEEE 29148. Each of the functional details within each functional process of the case study was classified into the following five categories with the following color-coding scheme:

- Functionality from business requirements– allocated to software functions - level 2,
- Functionality with more details from business requirements - level 3,
- Operational functionality for implementing in practice the business requirements functionality - level 3,
- Functionality derived from system requirements & allocated to software - level 3,
- Functionality related to an interface to other software applications - level 1 or 2.

Table 4 illustrates the application of this classification and color-coding scheme for the functional processes 'add a professor' and 'modify a professor'. In this example, the functions in yellow in Table 4 represent the 'system requirements' related to data and operational quality requirements allocated as software functions in the CRS case study. For ease of readability and traceability, this is re-labelled as 'quality functionality' in the subsequent tables.

Table 4. CRS case study: example of the classification of types of functionality within a functional process – color coded (N=2)

FP No/ Req.	Process name	Functional user/ object of interest	Sub-process description	Data group	DM type	CFP	Σ
1/ 1.2.1	Add a professor	Registrar/ professor	Registrar enters information for the professor	Professor data	E	1	
			The system validates the entered data and checks if a professor of the same name exists already	Professor data	R	1	
			The system creates a new professor	Professor data	W	1	
		Registrar/ professor	Display the system generated professor ID number	Professor ID	X	1	
		Registrar/ messages	Display error message	Messages	X	1	
							5

4.4 CRS: case study: functional distribution at level 3

Table 5 presents the results from the classification by origin of the functional details for all the functional processes of the course registration system in five different types of functionality (the types are color-coded) together with their corresponding COSMIC size in CFP units and percentages.

4.5 CRS case study: functional size distribution at level 2

Figure 1 illustrates, using the iceberg analogy, the usage of the above information, sized at the functional process level:

- 20 % of the functionality comes from the size of the functions listed from the systems-software requirements;
- 9 % comes from the functional details added later to the software requirements;
- 41% comes from the operational functionality that must be added to implement such functional requirements in an operational context (business, embedded software, etc.);
- 30% came from the implementation of quality derived functionality allocated to the software – here more specifically 'data integrity'.

Fig. 1. CRS case study - Scaling factors based of the size of functional processes

Table 5. CRS case study – functional classification & size at functional process level 3 (N=21)

ID	Functional process name	Direct business	Business details	Operational business	Quality functions	Interface	Total size in CFP	%	%	%	%
1	Add a professor	1	0	2	2	0	5	20	0	40	40
2	Modify a professor	1	0	1	1	0	3	33.3	0	33.3	33.3
3	Delete a professor	1	0	1	1	0	3	33.3	0	33.3	33.3
4	Enquire on a professor	1	0	2	1	0	4	25	0	50	25
							15				
……							…				
Sub-total in CFP		**21**	**9**	**42**	**30**	**1**	**102**	-	-	-	-
Percentage over TOTAL size		**20 %**	**9%**	**41%**	**30%**	-	**100 %**	-	-	-	-
Average size		**1**	**0.4**	**2**	**1.4**	-	-	-	-	-	-

It is to be observed that the above requirements were progressively identified, from earliest to latest, over the lifecycle.

Such information (i.e. the percentage per classification of requirements) can then be used as it becomes available in the early phases of a project as scaling factors to estimate the final size of the fully developed software, taking into account the functionality-type to be added across the project life.

Of course, the usual caveat applies similar types of applications, similar organizational contexts, etc.

As an example: if for a subsequent project, 33 functional processes are identified, this statistical information and scaling factors can be used to provide an estimate of the final size of the corresponding software:

- 33 functional processes at 1 CFP each would represent 21 % of the total functionality, that is, an estimated final size of 158 CFP.

These estimates are based on an average. In addition, an expected range of final size should be calculated using the standard deviation from the detailed descriptive statistics.

The above have all been classified and calculated on the basis of the functional processes allocated to software.

4.6 CRS case study: functional size distribution at level 1

A similar approach can be developed for earlier usage by using the list of 'system' requirements (e.g. level 1) instead of the list of functional processes (e.g. level 2) that become available later.

To develop a scaling factor for level 1, the size information available at levels 2 and 3 was rolled-up at level 1 (e.g., system requirements level in ISO-IEEE 29148) with the following system level functions from Table 1 – see results in Table 6.

As an example, if for a subsequent project, 10 additional system functions are identified, this statistical information and scaling factors can be used to estimate the final size of the corresponding software:

- 10 business functions at 3 CFP size at the feasibility study phase would represent 21% of the total expected total functionality (or 30 CFP), and would then scale up to an estimated final added functional size of 143 CFP.

Table 6. CRS case study – functional classification & size at system function level 1 (N=7)

ID	Functional process name	Business functions	Business details	Operational functions	Quality functions	Other	Total size	%	%	%	%
1	Maintain professor	4	0	6	5	0	15	27	0	40	33
2	Maintain student info	4	0	6	5	0	15	27	0	40	33
3	Maintain course	5	4	8	8	0	25	20	16	32	32
4	Maintain student schedule	5	3	14	7	1	29	18	11	49	25
	Sub-total maintain functions	**18**	**7**	**34**	**25**		**84**	**22**	**9**	**50**	**30**
	Average per maintain functions	**4.5**					**21**				
5	Close registration	1	1	2	3		7	15	15	29	43
6	Submit grades	1	1	4	1		7	15	15	55	15
7	Enquire report card	1	0	2	1		4	25	0	50	25
	Sub-total other functions	**3**	**2**	**6**	**5**		**18**	**17**	**12**	**36**	**28**
	Average per other functions	**1**					**6**				
	TOTALS & %	21	9	42	30	1	102	20	9	41	30
	Average from TOTAL	**3**	**1.3**	**6**	**4.3**		**14.6**				

4.7 RestoSys case study: documentation levels

The version of the restaurant management system case study (RestoSys) used in this research is the February 2019 version [22]. In the documentation of this CRS case study, the description of the functionality is also presented sequentially at three levels of increasing detail, where the RestoSys is composed of two parts: a mobile app and a web application:

- RestoSys ensures communication between the smartphone client (the waiter) and the web client (the administrator);
- web client maintains the database (which is included in the DB server).

The RestoSys application includes the following functionality:
- smartphone client receives the waiter's username and password;
- web server retrieves data from the DB and provides the required data to the smartphone client;
- smartphone client maintains the customer order (by adding or modifying an order);
- web client receives the administrator's username and password;
- web server retrieves data from the DB and provides the required data to the web client;
- web client maintains the required data.

The RestoSys includes the following tasks:
- order management allows the waiter to add, and/or modify an order via his smartphone. It also allows the administrator to delete an order. During working hours, the waiters (smartphone) and the administrator (web client) are continuously connected;
- account management, involves user management, and enables access to the application with a username and password;
- restaurant menu management allows the management of item(e.g. a dish and a beverage) families and the classification of items into item families.

Note that the users of RestoSys (waiter and administrator) must be logged on before executing one of the previous tasks (order management, account management, and restaurant menu management).

4.8 RestoSys case study: functional size at functional process level

Table 8 presents the functional processes of RestoSys classified into the five different types of functionality, including security-NFR.

Table 7. RestoSys case study - use case identification (N=10)

Actor	Global use cases	Detailed use cases
Waiter	Logon	FUR1: Logon
	Maintain order	FUR2: Maintain order
Administrator	Logon	FUR3: Logon
	Maintain data	FUR4: Maintain user FUR5: Maintain item FUR6: Maintain item family FUR7: Maintain table FUR8: Maintain restaurant menu
	Maintain order	FUR9: View the list of orders FUR10: Delete customer order

Table 8. RestoSys case study - list of functional processes and their sizes (N=33)

ID	Functional process name	Direct business	Business details	Operational functions	Quality functions	Security functions	Total size CFP	%	%	%	%	%
1	Log on to mobile app.	0	0	0	0	5	5	0	0	0	0	100
2	Add an order	4	0	7	5	0	16	25	0	44	31	0
3	Modify an order	3	0	6	3	0	12	25	0	50	25	0
Total functional size of mobile application							**33**					
4	Create new order	4	1	7	2	0	14	29	7	50	14	0
5	Modify existing order	3	2	3	1	0	9	33.3	22	33.3	11.1	0
6	Log on as administrator	0	0	0	0	4	4	0	0	0	0	100
7	Add user in web app.	1	0	2	1	0	4	25	0	50	25	0
8	View the user list	1	0	2	1	0	4	25	0	50	25	0
..	…….											
Total functional size of web application							**118**					
Total in CFP		**40**	**3**	**63**	**36**	**9**	**151**	-	-	-	-	-
Percentage over TOTAL size		**26 %**	**2%**	**42 %**	**24%**	**6 %**	**100 %**	-	-	-	-	-
Average size		**1.21**	**0.09**	**1.9**	**1.09**	**0.27**	-	-	-	-	-	-

4.9 RestoSys case study: functional size at level 2

The previous information and sizes at level 3 can be rollup-up (e.g., aggregated) at level 2 of the software functions - that is, the 10 use cases from Table7. The results of this consolidation are presented in Table 9 where:

- 27 % of the functionality comes from the size of the functions listed from the systems-software requirements;
- only 2 % of the functionality comes from the functional details added later to the software requirements;
- 42% comes from the operational functionality that must be added to implement such functional requirements in an operational context (here, a business application);
- 30% came from the implementation of quality derived functionality allocated to the software – here more specifically:
  - 24 % as ‘data integrity’,
  - 6% as security through the 2 login simple functions.

Table 9. Resto case study – list of the use cases and their sizes – level 2 (N=10)

ID	Use case name	Direct business	Business details	Operational functions	Quality functions	Security	Total size in CFP	%	%	%	%	%
1	Log on to mobile app.					5	5					100
2	Maintain order	7	0	13	8	0	28	25	0	46	29	
Total functional size of mobile application							**33**					
2	Maintain order	7	3	10	3	0	23					
	Sub-total FUR 2 + NFR	14	3	23	11	5	56	25	5	41	20	9
3	Log on as administrator					4	4	0	0		0	100
4	Maintain user	5	0	8	4	0	17	29	0	47	23	
5	Maintain item	5	0	7	5	0	17	29	0	47	29	
6	Add an item family	4	0	8	5	0	17	23	0	47	29	
7	Maintain table	5	0	7	7	0	19	26	0	37	37	
8	Maintain menu	4	0	6	4	0	14	29	0	43	29	
9	View list of orders	1	0	2	1	0	4	25	0	50	25	
10	Delete an order	1	0	1	0	0	2	50	0	50	0	
Total functional size of web application							**118**					
Total in CFP		**39**	**3**	**62**	**37**	**9**	**151**	-	-	-	-	
Percentage over TOTAL size		**26 %**	**2 %**	**42 %**	**24 %**	**6 %**	**100%**	-	-	-	-	
Average size in CFP for 8 FUR		**5**	**0.4**	**8**	**4.6**	**4.5**	**15 CFP (or 4.5 & 19)**	-	-	-	-	

5 Summary and future works

The main input of function point sizing is the set of functional requirements for a piece of software. However, very early in the software development lifecycle it is unrealistic to expect this set of requirements describes the full scope of functionality of the software as a whole, including all the necessary functional details. Early in the development life cycle, there are obviously many functional unknowns. Therefore, the size one can measure directly from a set of incomplete functional requirements early in the lifecycle and the size measured much later from a very detailed set of functional requirements fully developed and implemented in a piece of software will differ.

A number of size estimation techniques have already been proposed but are based mostly on statistical analyses [6-10] and do not provide insight into the sources of the gaps, timing and how such insights may improve size estimation.

Following the exploratory analytical study by Abran et al. [6] that identified the nature of the gaps between earlier sizing and final/delivered size and their sources, this research used the ISO-IEEE 21948 standard on requirements engineering [7] to identify

sources of requirements and related levels of documentation for system and software functionality. ISO-IEEE 29148 presents a number of concepts related to the sources, types and levels of detail of the requirements throughout the system and software life cycle. In summary, the initial set of requirements originates from two sets of sources, the business stakeholders and other stakeholders, which led to the 'systems' requirements. From the system functional requirements, some were allocated to software requirements (as well as to hardware requirements and at times to manual operational procedures). ISO-IEEE 29148 also notes that in addition to software functions explicitly identified, there may be interfaces identified, but not yet specified, as well as quality requirements, still at a high level.

At various points in the lifecycle, these lists of software functions are detailed, programmed, tested and implemented. Similarly, for system non-functional quality requirements, some of which may be allocated later to additional software functional requirements. However, an outside measurer without access to all the documentation, or undocumented functionality (such as those derived from system-NFR and implemented late in the testing phase), would miss a number of software functions. Similarly, measurement done earlier in the lifecycle when a number of software functions are neither identified nor specified in detail, would be 'invisible' to the outside and early measurer.

To develop the functionality-based approximation technique presented in this paper, we used two case studies of the COSMIC group, which documented requirements at various levels of detail as well as the corresponding COSMIC function points measured on the basis of the most detailed requirements [20, 21]. From these observations, and comparison of the information available and described using concepts from ISO-IEEE 29148, at various points in time size scaling factors specific to these case studies and levels of documentation and sizing were developed. The challenge of designing functionality scaling factors was worked out from known detailed documented requirements measured in a context of full visibility, full documentation and no uncertainty. More specifically, requirements were positioned at three levels of documentation, from the initial high-level system level down to the most detailed functional levels where all the requirements allocated to the software from the business functions to the operational functions as well as quality functions allocated to software were known.

Such scaling ratios, with successive levels of documentation, can be used in future projects such as project progress through the lifecycle, and documentation of levels of completeness.

While the scaling factors derived from the two case studies are specific to these case studies, this functionality-based approximation technique can be used in most organizations provided data on past projects can be collected and relevant classification of functionalities identified. For the purpose of this paper, the ISO 19761 COSMIC function point standard was taken as reference for discussion, while the majority of concepts presented are generic to other similar ISO standards.

Additional empirical research work is required to consolidate the insights developed in the research reported here. In particular, additional case studies from other domains may provide additional types and sources of functionality that could be considered for scaling purposes. Also, lessons learned from the research work in [15-19] on measuring software functionality derived from system–NFR must be investigated. Finally, access

to measurements of large software in operations contexts would be useful to explore the robustness of this proposed functionality-based approximation technique.

References

1. Abran, A., Dumke, R. (Eds.): COSMIC Function Points Theory and Advanced Practices, CRC Press. ISBN 978-1-4398-4486-1 (2011).
2. Abran, A.: Software Metrics and Software Metrology, John Wiley & Sons and IEEE-CS Press, New Jersey, p. 328, ISBN:978-0-470-59720-0 (2010).
3. COSMIC Group: The COSMIC Functional Size Measurement Method Measurement Manual, version 4.0.2, available on: https://cosmic-sizing.org/publications/measurement-manual-v4-0-2/ (2017).
4. Boehm, B., Abst C.: Software Cost Estimation with COCOMO II, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (2000).
5. Yılmaz G., Ungan, E., Demirörs, O.: The Effect of the Quality of Software Requirements Document on the Functional Size Measurement, United Kingdom Software Metrics Association International Conference on Software Metrics and Estimating, London, UK (2011).
6. Ungan, E., Trudel, S., Abran, A.: Analysis of the Gap between the Initial Estimate Size and the Final (True) Size of the Software', 28th IWSM-MENSURA Conference, Vol. 2207, pp. 123-137, Beijing, China, ISSN 1613-0073 (2018).
7. ISO 29148 Systems and Software Engineering -Life cycle processes- Requirements Engineering, International Organizations for Standardization (ISO), Geneva (2011).
8. Desharnais J.M., Abran, A.: Approximation techniques for measuring Function Points, 13th International Workshop on Software Measurement (IWSM), pp. 270-286. Springer Verlag, Montréal, Canada (2003).
9. Santillo, L.: Early FP Estimation and the Analytic Hierarchy Process, ESCOM-SCOPE Conference, Munich, Germany (2000).
10. Vogelezang F.W., Prins, T.G.: Approximate size measurement with the COSMIC method: Factors of influence, SMEF Conference, Rome, Italy (2007).
11. Almakadmeh, K., Abran, A.: Experimental evaluation of an industrial technique for the approximation of software functional size. International Journal of Computers and Technology. vol. 10, no. 3, pp. 1459-1474, ISSN 22773061 (2013).
12. Almakadmeh, K.: Development of a scaling factors framework to improve the approximation of software functional size with COSMIC – ISO 19761, Doctoral Thesis, École de technologie superieure, University of Québec, Canada (2013).
13. COSMIC Group: Guideline for Early or Rapid COSMIC Functional Size Measurement by using approximation approaches, available on: https://cosmic-sizing.org/publications/guideline-for-early-or-rapid-cosmic-fsm/ (2015).
14. Lavazza, L., Morasca, S.: Empirical evaluation and proposals for bands-based COSMIC early estimation methods, Journal of Information and Software Technology. vol. 109, pp. 108-125 (2019).
15. Ungan, E., Trudel, S., Poulin, L.: Using FSM patterns to size security non-functional requirements with COSMIC, 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement (IWSM Mensura '17) ACM, pp. 64-76, New York, USA (2017).
16. Al-Sarayreh, K.T.: Identification, specification and measurement, using international standards, of the system non-functional requirements allocated to real-time embedded software, Doctoral Thesis, Ecole de technologie supérieure, University of Québec, Canada (2011).

17. Abran, A., Al-Sarayreh, K.T., Cuadrado Gallego, J.: A Standard based Reference Framework for System Portability Requirements, Computer Standards & Interfaces Journal. Elsevier, vol 35, pp. 380-395 (2013).
18. Al-Sarayreh, K.T., Trudel, S., Meridji, K., Abran, A.: System Security Requirements: A Framework for Early Identification, Specification and Measurement of Software Requirements, Computer Standards & Interfaces Journal (2019).
19. Al-Sarayreh, K.T., Abran, A., Cuadrado Gallego, J.: A Standard based Model of System Maintainability Requirements, Journal of Software Evolution and Process. vol. 25, no. 5, pp. 459-505 (2013).
20. Al-Sarayreh, K.T., Abran, A.: Software Specifications Framework for System Operations Requirements, International Journal of Computer and Information Sciences. vol. 10, no. 3 (2010).
21. COSMIC Group: Course Registration System case study, available on: https://cosmic-sizing.org/publications/course-registration-c-reg-system-case-study-v2-0-1/ (2015).
22. COSMIC Group: Case Study Sizing Natural Language/UML Use Cases for Web and Mobile Applications using COSMIC FSM, available on: https://cosmic-sizing.org/publications/restosys-case/ (2019).