Automated COSMIC Function Points Measurement for C Program Using Regular Expressions

Donatien Koulla Moulla [1,2], Oumate [2], Ernest Mnkandla [1], Hassan Soubra[3] and Alain Abran [4]

[1] *University of South Africa, The Science Campus, Florida, 1710, South Africa*
[2] *University of Maroua, Maroua, P.O. Box 46, Cameroun*
[3] *The German University in Cairo, New Cairo, Egypt*
[4] *École de Technologie Supérieure, 1100, rue Notre-Dame Ouest, Montréal, Québec, H3C 1K3, Canada*

Abstract

Functional size measurement (FSM) is an important basis for measuring productivity and estimating the effort required for software activities. Automating FSM can be very valuable for organizations with a large number of projects to measure in a very short time, and there are several issues related to manual FSM, including measurement errors due to human measurers, the cost of measurement, measurers' subjective interpretations, and assumptions regarding the project scope. This paper presents an automated FSM for the C programming language, selected for being the most popular programming language from 1965 to 2020 and, as of January 1, 2022, in the second position after Python language. The FSM procedure is based on COSMIC ISO 19761, which is an FSM method designed based on metrology and software engineering principles. An automated measurement prototype tool was also introduced. The automated measurement tool can be useful for organizations and practitioners.

Keywords

Functional Size Measurement, COSMIC – ISO 19761, C Programming Language, Sizing Software Automation, Automation Tool

1. Introduction and background

In software engineering, as in other engineering disciplines, mastering and evaluating software development is a concern for both practitioners and industry, and through effective measurement, organizations can estimate, control, plan, and learn to perform software work more effectively [1, 2]. As Tom DeMarco said in [3]: "You cannot control what you cannot measure". Measurement is a fundamental engineering concept that helps managers to find effective strategies for improving software management. According to the principles of metrology defined in [4], the term of measurement is used in the context of "measurement method", "application of a measurement method", and "measurement results". There are several options for sizing software: lines of code, Use case Points, Object Points, Story Points, and functional size with Function Points.

Huijgens et al. [5] conducted a structured survey of 336 Functional size measurement (FSM) specialists who concurred that automated FSM from source code is important, but challenging. Most respondents think that automated FSM will help measurement specialists and decision-makers. In this survey, COSMIC was the preferred FSM method for automation, followed by IFPUG and NESMA. The respondents considered automated FSM to be most suitable for baseline, benchmarking, maintenance, and legacy purposes.

Compared with other options, the COSMIC functional size measurement (FSM) method is designed according to basic software engineering and metrology principles and is technology-

IWSM-Mensura, September 28–30, 2022, Izmir, Turkey
EMAIL: moulldk@unisa.ac.za; oumateb@gmail.com; mnkane@unisa.ac.za; hassan.soubra@guc.edu.eg; alain.abran@etsmtl.ca
ORCID: [orcid.org/0000-0001-6594-8378]


independent. Two main software artifacts are generally used to measure the functional size of a piece of software [6, 7]: requirement documents in the early development stages and lines of code once the code has been developed.

Applying FSM procedures manually is time-consuming, which is problematic for organizations with a large number of projects to measure in a very short time, either for project estimation purposes or productivity studies [8]. In addition, the manual application of FSM to a very large set of source code inputs requires specialized expertise when there are a variety of languages in which the source code has been implemented.

There are main ways to automate functional size measurement [7]:

1. Automate requirements analysis – with measurement based on key words and grammar patterns.

- Input is a set of textual or modeled requirements ;
- Output is the functional size measured ;
- An example is the automated COSMIC function points (CFP) size measurement from UML models in [9, 10, 11].

2. Automated code analysis:

- Input is source code;
- Output is the functional size measured;
- Here, lines of code measurement are usually automated against the standards and best practices in any specific language, rather than on the quality of a specific piece of code. An example is the automated COSMIC CFP size measurement from Java source code [12, 13, 14, 15].

In practice, there is a number of challenges to automated FSM:

- The requirements are almost always partial, incomplete, and ambiguous. The requirements do not describe the full scope of the functionality of the software with all the necessary functional details [16]. As a project progresses, the requirements will be detailed and changed as the project moves through the life cycle.
- While FSM methods are technology-independent, an automation design is required to handle the specifics of each programming language.

The most common reason for measuring the functional size of software from the source code is to develop a baseline of the relationship between size and effort in the form of an estimation model based on functional size [17]. Moreover, the size measured from the source code reflects what has been delivered with more accuracy than the incomplete and ambiguous requirements.

The C programming language was created by Dennis Ritchie at the former Bell Laboratories in 1972 [18]. The C programming language was the most popular programming language from 1965 to 2020 [19], and as of January 1, 2022, it is in second position after the Python language [20]. For example, Linux and Windows kernels are largely developed in the C programming language. In this study, we focused on the automated measurement of the COSMIC CFP from C programs using regular expressions.

This study provides the first automated FSM from the source code using regular expressions. The current study focuses on the design and implementation of an automated FSM for C programs based on COSMIC ISO 19761 method. The automated measurement tool can be useful for organizations and practitioners.

The remainder of this paper is organized as follows. Section 2 presents an overview of related work on the COSMIC FSM method and its measurement automation to date. Section 3 presents the proposed automated measurement procedure for sizing C programs using the COSMIC rules and regular expressions. Section 4 presents the automated measurement prototype. Section 5 concludes the paper with directions for future research.

2. Related work

2.1. Benefits for sizing software with COSMIC – ISO 19761

Over the past few decades, several FSM methods have been proposed, such as FPA [21], MkII [22], NESMA [23], FisMA [24], and COSMIC [25]. The most common reason for measuring software size is to estimate the effort or cost of development. Measuring software sizes can be valuable for many purposes other than project estimation, as discussed in Sneed [26] and Symons [27].

Several studies have shown the quality of COSMIC as an FSM method. For instance, it has been shown to perform better than Story Points when it comes to building effort estimation models and productivity within an agile context [28, 29]. Di Martino et al. [30] empirically investigated the performance of COSMIC and FPA methods for effort estimation of web applications: based on two empirical studies using 25 web applications, they found that COSMIC outperformed FPA for effort estimation. Symons et al. [31] demonstrated that these challenges for software project control and estimating upcoming activities have been achieved through the advances made by the COSMIC community over the years. The flexibility of the COSMIC method makes it possible to measure any type of software and is used around the world.

This motivated the current choice of COSMIC as a measurement method to design automated functional size measurements for C programs. The COSMIC method defines four types of data movement as follows:

1. ENTRY: A data movement that moves a data group from a functional user across the boundary into the functional process where it is required.
2. EXIT: A data movement that moves a data group from a functional process across the boundary to the functional user that requires it.
3. READ: A data movement that moves a data group from persistent storage to the functional process that requires it.
4. WRITE: Data movement that moves a data group from inside a functional process to persistent storage.

2.2. Functional size measurement automation in the literature

To address FSM automation issues, several papers have been published on COSMIC-based measurement automation, both from requirement and source code analyses. An example of the latter is the study by Soubra et al. [8] on an approach for a theoretical 'universal' tool based on COSMIC ISO 19761 for the automated measurement of software written in different programming languages. Their work included a prototype tool based on COSMIC and MIPS, with a small-scale validation, and was limited to a specific release of the MIPS architecture and a specific instruction set.

Ahmed et al. [6] proposed an FSM procedure based on COSMIC ISO 19761 to measure software artifacts expressed in the ARM's base 32-bit assembly code. They introduced an automated measurement tool prototype that can produce the functional size in the CFP of an ARM program, but did not include all ARM instructions.

From a comparison and lessons learned from their previous studies [12, 13, 14], Tarhan et al. [32] derived an operational scenario for automated FSM from software code and proposed a set of requirements that must be considered in automation. The authors replicated the study in [33] by developing a tool called "COSMIC Solver" for Java Business Applications (JBA), and their results indicated that CFPs measured manually and automatically converged at 77%.

Chamkha et al. [15] proposed a "JavaCFP" plugin tool for measuring the COSMIC functional size of Java source code being developed. Their measurement tool can be used to control the completeness of the implemented functionality against specified requirements, to identify deviations, and to generate progress reports on the implementation of new functions. Sahab et al. [17] developed a CFP4J library to automate COSMIC functional size measurements from Java web applications using the Spring Web MVC framework. Their contribution lies in defining mapping rules from code and a publicly available software library to automate the COSMIC functional size of Java web applications that use the Spring MVC framework.

Ungan et al. [34] presented ScopeMaster®, the first commercial tool to perform COSMIC measurement on a set of free-form textual requirements in English. ScopeMaster performs several successive steps of analysis, individually and collectively, on the textual requirements to detect

candidate COSMIC Objects of Interest, potential functional users, potential data movements, and potential defects. The details of how ScopeMaster® performs these techniques are proprietary and are protected by a pending patent application; however, the measurement results are fully transparent.

Bagriyanik et al. [35] proposed an ontology model for transforming requirements into COSMIC function point method concepts, and a method was developed to automatically measure the functional size of software using the designed ontology. To validate their method, they implemented a software application using requirement data from several real projects. Their findings indicated that manual and automated measurement results were in agreement.

Meiliana et al. [11] used the XML structure of the UML sequence diagram to automate software size measurements. Functional size was measured using the COSMIC method, while structural size was calculated based on the control structures in sequence diagrams.

Zaw et al. [36] proposed an automated software size measurement tool, including a generation model based on UML, SysML, and Petri nets. General mapping rules between the COSMIC FSM and a generation model to measure the size of the software have been proposed.

Sellami et al. [37] designed an extended sizing method by considering the structural aspects of a sequence diagram to quantify its size. These functional and structural sizes can then be used as distinct independent variables to improve the effort estimation models. Their findings showed that the size of sequence diagrams can be measured from two perspectives, both functional and structural, and at different levels of granularity with distinct measurement units. The authors refined their previous study through the assessment of the nested (multi-level) control structures in the sequence diagram and a web site case study "Digital-Training Center" were used to depict and apply the proposed measurement algorithms [38].

Abrahão et al. [39] defined a measurement procedure (OO-HCFP) for Object-Oriented Hypermedia method (OO-H) web applications based on COSMIC. They argued that the results obtained using their proposed approach were more accurate than those obtained using other measurement approaches based on function points and design measures.

De Vito et al. [40] provided a measurement procedure to derive the COSMIC functional size from UML software artifacts, and developed a prototype tool (J-UML COSMIC). To assess the measurement procedure, they carried out two case studies and compared the measurement results provided by the tool with those obtained by experts by applying the standard COSMIC method. They concluded that the tool allowed incremental accurate measurements to be obtained when new or existing models were considered.

In summary, a number of studies have proposed COSMIC FSM automation, but none yet have proposed, to the best of our knowledge, an automated measurement solution for C programs. This motivated the present study.

3. COSMIC FSM procedure for sizing C programs

This section presents an overview of C program structure and the proposed measurement approach.

3.1. Overview of C program structure

The general structure of the C program can be represented in a simplified form as follows (Figure 1):

```
[Preprocessor directives]
[Secondary functions]
int main(void){
Declaration of internal variables
Instructions
return 0;}
```

Figure 1: A simplified form of a C program

The preprocessor directives are used to include libraries, and the secondary functions represent subprograms, either procedures or functions. A subprogram is defined as a block of code that has a name and consists of a sequence of instructions that perform a specific task. The main program is a procedure that governs the operation of the program.

In this study, a C program was represented as a function. These functions consist of two main elements:

- Function header, where the name, type of input parameters, and type of output value are defined;
- The body of the function where variables are defined, and all instructions are executed.

These information will serve as guidelines for the mapping of COSMIC to C programs. The structure of the C function is represented in a simplified form as follows (Figure 2):

```
type function_name(arguments){
declarations of internal variables
instructions
}
```

Figure 2: A simplified form of a function

3.2. The proposed approach: mapping COSMIC to C program

The purpose of this procedure is to apply the COSMIC method to the source code of programs written in C, for either effort estimation or productivity studies. We considered the whole set of functional user requirements (FUR) expressed within a C program; therefore, the measurement scope is the whole program written in C. We used a detailed functional process at the granularity level.

Our COSMIC mapping process can be summarized by the identification of:

- Functional process: a C function is considered as a functional process.
- Data group: A data group consists of a unique set of data attributes that describe a single object of interest. We assumed that each variable could be considered as a data group.
- Data movements: There were four data movement types: ENTRY, EXIT, READ, and WRITE.

Table 1 lists the proposed mapping rules between the COSMIC concepts and the elements of a function. Any input parameter of a function should be mapped to COSMIC ENTRY (E) data movements and the output of a function should be mapped to COSMIC EXIT(X) data movements. For READ (R) and WRITE (W) COSMIC data movements, any assignment to a variable (respectively from a variable) should be mapped to WRITE (W),), (respectively a READ (R) and WRITE (W)) WRITE (R) COSMIC data movements can also be linked to the initializations of variables during their declaration.

Table 1

Rules for identifying data movements in a C program

COSMICS Concepts	Elements of C program
ENTRY data movement	Identify an ENTRY (E) for each input parameter of a function. The number of parameters of a function corresponds to the number of ENTRY data movements.
EXIT data movement	An EXIT (X) corresponds to the value returned by a function. If there are several functions in a program, each output of a function is an EXIT data movement.
READ data movement	Identify a READ (R) during operations. Each variable read to carry out a processing corresponds to the READ (R) data movement.
READ data movement	When the scanf function is called, it also corresponds to a READ (R).
WRITE data movement	Each printf function corresponds to a WRITE (W).

WRITE data movement	Identify a WRITE (W) during assignment operations. Each assignment is a WRITE (W).
WRITE data movement	The initialization of a variable is also considered as a WRITE (W) data movement. Therefore, each initialization corresponds to a WRITE (W).

As described in Section 3.1, a function consists of two main elements: the function header and the body of a function. Table 2 presents the mapping of COSMIC to the C program instructions and their corresponding regular expressions.

Table 2
Mapping COSMIC Data movements to C program instructions using regular expressions

Instructions	Data movements	Regular expressions
Definition of the function header	ENTRY data movements	(([a-z]+)\s+(\w+)\s*(\(((\s*\w*\s+\w*(\s*\,| \s*))*| \s*(\w*)\s*)\)))
Assignment with constant values	One WRITE (1W)	((\w*)\s*\(=)\s*([0-9.]+|[’\”][a-zA-Z]+[’\”])\s* \;)
Assignment with variable values	One READ (1R), one WRITE (W)	((\w*)\s*\(=)\s*([a-zA-Z]+)\s* \;)
Arithmetic instruction	Two READ (2R), one WRITE (W)	((\w*)\s*\(=)\s*(([a-zA-Z]+)\s*(\+|\-|*|\/|\%)\s*([a-zAZ]+))\s* \;)
Increment	One READ (1R), one WRITE (W)	((\w+)\s*(=)\s*(((\w+)\s*(\+|\-|*|\/)\s*([0-9.]+))|(([09.]+)\s*(\+||*|\/)\s*(\w+)))\s*\;)|((\w+)\s*((\+|\-|*|\/)\s*=)\s*([0-9.]+)\s*)
Logical expressions Type 1 (between variable and constant)	One READ (1R)	(([a-zA-Z]+\s*(\<|\> |\6| \> | \==|\!=)\s*[0-9.]+)| ([0-9.]+\s*(\<|\>|\6| \> | \==|\!=)\s*[a-zA-Z]+))
Logical expressions Type 2 (between variable and variable)	Two READ (2R)	(([a-zA-Z]+)\s*(\<|\> |\6| \> | \==|\!=)\s*([a-zA-Z]+))
Read	One READ (1R)	((scanf)\s*\([”\’][a-zA-Z0-9.%]+[”\’]\s*\,\s*(.*?)+\)\s*\;)
write	one WRITE (W)	((printf)\s*\([”\’](.*?)*[”\’](.*?)*\)\s*\;)

4. Automated measurement tool prototype

This section presents the approach used to build the measurement tool prototype based on the COSMIC C-mapping rules defined in Section 3.

4.1. Tool architecture and description of the algorithm

The tool prototype is aimed at automating the COSMIC FSM from a C program file, visualizing it, and saving the measurement results. The general architecture of the tool comprises three modules (Figure 3).

1. Graphical user interface (GUI) module: This module allows a user to select a C program file to measure, visualize, and save the measurement results.
2. Filtering and data extraction module: This module is organized into two parts. The first part extracts all the function headers defined in the program to identify the ENTRY and EXIT data movements. The second part groups instructions per category. Persistent storage exists in this module.
3. Measurement module: This module contains a list of the identified data movements. In addition, the numerical values are assigned to each data movement (one CFP per COSMIC data movement), and then the aggregation of all the identified CFP is made.

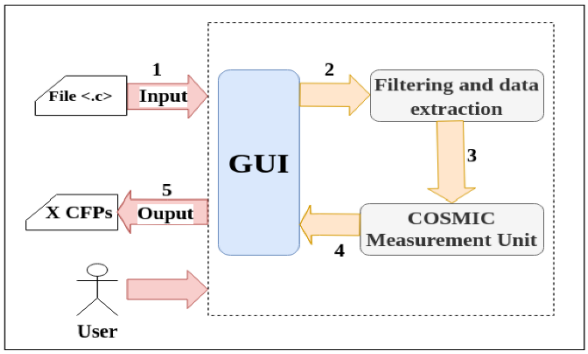

Figure 3: A simplified architecture of the COSMIC-C tool prototype

The algorithm describing the tool functions is shown in Figure 4.

Algorithm: Measurement process

Input: C program file
Outputs: size in CFP
Begin
{
1. Read file
2. Identify all functional processes
3. **For** Each functional process:
   For each output parameter:
      count 1 EXIT (X)
   For each input parameter:
      (**If** there are two separated words with at least one space **then** count 1 ENTRY (E)
       If there are comas, **then** ENTRY <– Count (number of comas) + 1)

   For each body of the function:
      If the instruction is the initialization, **then** count 1 WRITE (W)
      If the instruction is an arithmetic instruction, **then** count 2 READ (R) and 1 WRITE (W)
      If the instruction is an value assignment to a variable, **then** count 1 WRITE (W)
      If the instruction is an variable assignment to a variable, **then** count 1 READ (R) and 1 WRITE (W)
      If the instruction is an logical expressions Type 1 (between variable and constant), **then** Count 1 READ (R)
      If the instruction is an logical expressions Type 2 (between variable and variable), **then** Count 2 READ (R)
      If the instruction is an increment, **then** count 1 READ (R) and 1 WRITE (W)
      If the instruction is printf, **then** count 1 WRITE (W)
      If the instruction is the scanf, **then** count 1 READ (R)
4. Add ENTRIES, EXITS, READS and WRITES from step 3.
5. Aggregate the CFP for all functional processes
}
End

Figure 4: Algorithm of measurement process

4.2. Validation of the prototype

The automated measurement tool prototype was implemented using Python language. As a test case example, the source code of the C program, as shown in Figure 5, was used.

```c
#include<stdio.h>

int sum(int x, int y){
        int z = 0;
        z = x + y;
        return z;
}
void checkEvenOrOdd(int x){
        if(x%2==0)
                printf("%d is Even\n", x);
        else
                printf("%d is odd\n", x);
}
int square(int x){
        return x*x;
}
int maximum(int x, int y){
        int max = 0;

        if (x<y)
                max = y;
        else
                max = x;

        return max;
}

int main(){
        int x= 0, y = 0;
        printf("Give two integers numbers\n");
        scanf("%d%d", &x,&y);
        int z = 0, result;
        z = x + 2;
        result = z;
        int tmp = square(z);
        int su = sum(x, y);
        int max = maximum(x, y);
        result += 2;
        checkEvenOrOdd(result);
        printf("The sum of %d and %d is : %d\n", x, y, su);
        printf("The square of %d is : %d\n", z, tmp);
        printf("The maximum of %d and %d is : %d\n", x, y, max);

        return 0;
}
```

Figure 5: Content of the C program file used for testing the automation prototype

Figure 6 shows the automated measurement results for all COSMIC data movements from the C program file.

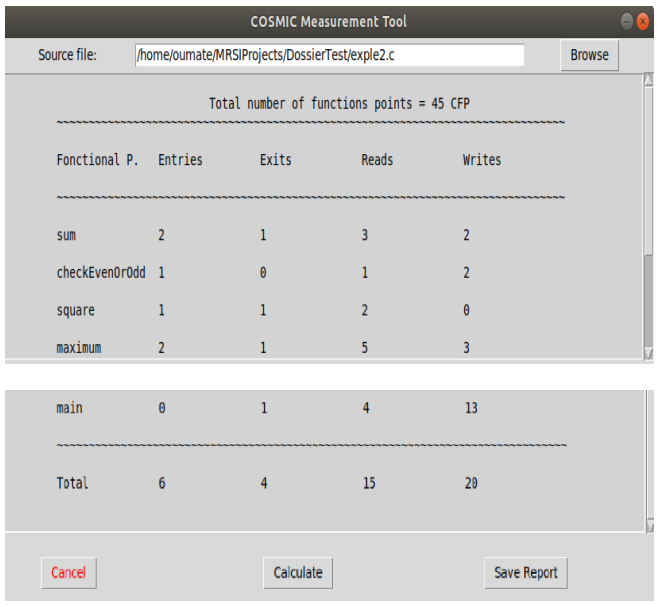

Figure 6: CFP automated measurement results

As shown in Figure 6, the tool identified 6 entries, 4 exits, 15 reads, and 20 writes from the C program file. The total number of COSMIC function points of the program was 45 CFP. Table 3 presents the corresponding manual measurement results for the same program using the COSMIC ISO 19761 method. Three people performed manual measurements, and one of them was a certified COSMIC measurer.

Table 3

CFP manual measurement results

N° of lines of code	COSMIC data movements	Value in CFP
3	1X, 2E	3
4	1W	1
5	2R, 1W	3
6	1R	1
8	1E	1
9	1R	1
10	1W	1
12	1W	1
14	1E, 1X	2
15	2R	2
17	1X, 2E	3
18	1W	1
20	2R	2
21	1R, 1W	2
23	1R, 1W (optional)	2
25	1R	1
28	1X	1
29	2W	2
30	1W	1
31	1R	1
32	1W	1
33	1R, 1W	2
34	1R, 1W	2
35	1W	1
36	1W	1
37	1W	1
38	1R, 1W	2
40	1W	1
41	1W	1
42	1W	1
30	**Total entries: 06; Total exits: 04; Total reads:15; Total writes: 20**	**Total CFP: 45**

From Table 3, the same number of COSMIC function points was obtained from both the automated and manual measurements.

5. Conclusion

Several studies have shown the importance of FSM automation for the software industry, and while there are a number of studies on COSMIC FSM automation for a variety of contexts and programming languages, none have tackled automated measurement solutions for C programs. In this

study, we proposed a measurement procedure for sizing C programs using the COSMIC rules, including relevant C regular expressions.

Next, an automated measurement tool prototype is introduced, with a test example as a case study. The measurement results for this automated measurement tool prototype were similar to those obtained manually. Such work on the COSMIC FSM automation of C programs can be useful for organizations' estimation purposes or benchmarking studies.

Limitations refer to influences or shortcomings that are beyond researchers' control and place restrictions on the methodology and analysis of research data [41]. The limitations of this study related to the research problem under investigation are as follows:

- The proposed prototype tool correctly identified the Main() function as a COSMIC functional process. However, this tool does not identify the triggering Entry. However, this limitation should be addressed in future studies.
- Our research relates exclusively to software written in C programming language.
- A test case example given in this study is relatively simple for both humans and the software to handle. More tests should be done on the different data structures.

This study was limited to software written in C programming language. We plan to extend our study to other languages with similar syntax in terms of regular expressions such as Java, JavaScript, and Python. We also plan to explore existing tools for lexical analysis to extract COSMIC function points from source code. To reveal any limitations or threats to validity of automated measurement from source code, we plan to compare the size measured based on the user requirement of the same piece of software and the source code.

6. Acknowledgment

We are grateful to anonymous reviewers.

7. References

[1] cosmic-sizing.org, Why measure software size?, 2022. URL: https://cosmic-sizing.org/cosmic-sizing/intro/why-measure-size/.

[2] M. Unterkalmsteiner, T. Gorschek, A.M.M. Islam, C.K. Cheng, R.B. Permadi, and R. Feldt, Evaluation and measurement of software process improvement: a systematic literature review, IEEE Transactions in Software Engineering, 38(2) (2012) 398–424. doi: 10.1109/TSE.2011.26

[3] T. DeMarco, Controlling Software Projects, New Jersey, Yourdon Press, 1982.

[4] VIM ISO/IEC Guide 99 International vocabulary of metrology — Basic and general concepts and associated terms (VIM), International Organization for Standardization — ISO, Geneva, 2007.

[5] H. Huijgens, M. Bruntink, A. van Deursen, T. van der Storm, and F. Vogelezang, An Exploratory Study on Functional Size Measurement Based on Code, in: Proceedings of IEEE/ACM International Conference on Software and System Processes (ICSSP), 2016, pp. 56-65. doi: 10.1145/2904354.2904360

[6] A. Darwish, and H. Soubra, COSMIC Functional Size of ARM Assembly Programs, in: Proceeding of the 30th IWSM-Mensura. Mexico city, Mexico, 2020.

[7] H.V. Heeringen, Automated Function Points, the Game Changer!, IT Conference – ISBSG, 2020.

[8] H. Soubra, Y. Abufrikha, and A. Abran, Towards Universal COSMIC Size Measurement Automation, in: Proceedings of the 30th IWSM-Mensura, Mexico city, Mexico, 2020.

[9] V. Bévo, G. Lévesque, and A. Abran, UML notation for functional size measurement method, in: Proceedings of the International Workshop in Software Measurement - IWSM, Lac-Supérieur, Canada, 1999.

[10] T.M. Fehlmann, and E. Kranich, COSMIC functional sizing based on UML sequence diagrams, in: Proceedings of the MetriKon, Kaiserslautern, Germany, 2011.

[11] Meiliana, S. Karim, S. Liawatimena, A. Trisetyarso, B. S. Abbas, and W. Suparta, Automating functional and structural software size measurement based on XML structure of UML sequence diagram, in: Proceedings of the IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom), IEEE, Phuket, Thailand, 2017, pp. 24–28. doi:10.1109/CYBERNETICSCOM.2017.8311709.
[12] A.A. Akca, and A. Tarhan, Run-time measurement of COSMIC functional size for java business applications: Initial results, in: Proceedings of the IWSM-Mensura, IEEE, Assisi, Italy, 2012, pp. 226–231. doi: 10.1109/IWSM-MENSURA.2012.40.
[13] R. Gonultas, and A. Tarhan, Run-time calculation of COSMIC functional size via automatic installment of measurement code into Java business applications, in: Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications. IEEE, Madeira, Portugal, 2015, pp.112–118. doi: 10.1109/SEAA.2015.30.
[14] M.A. Sag, and A. Tarhan, Measuring COSMIC software size from functional execution traces of Java business applications, in: Proceedings of the IWSM-Mensura 2014, IEEE, Rotterdam, Netherlands, 2014, pp. 272–281. doi: 10.1109/IWSM.Mensura.2014.29.
[15] N. Chamkha, A. Sellami, and A. Abran, Automated COSMIC Measurement of Java Swing Applications throughout their Development Life Cycle, in: Proceedings of the IWSM-Mensura, Beijing, China, 2018, pp. 20-33.
[16] A. Abran, Software Estimation: How to use ISBSG data for early software sizing?, IT Conference – ISBSG, 2020.
[17] A. Sahab, and S. Trudel, COSMIC Functional Size Automation of Java Web Applications Using the Spring MVC Framework, in: Proceedings of the 30th IWSM-Mensura, Mexico city, Mexico, 2020.
[18] Gartner, C (Programming Language), 2022. URL: https://www.gartner.com/en/information-technology/glossary/c-programming-language.
[19] Statistics and Data, The Most Popular Programming Languages – 1965/2022 – New Update, 2022. URL: https://statisticsanddata.org/data/the-most-popular-programming-languages-1965-2022-new-update/.
[20] TIOBE Software, 2022. URL: https://www.tiobe.com/tiobe-index/.
[21] ISO/IEC 20926: Software and systems engineering - Software measurement - IFPUG functional size measurement method 2009, 2nd ed., ISO, Geneva, 2009.
[22] ISO/IEC 20968: Software engineering - Mk II Function Point Analysis - Counting Practices Manual, ISO, Geneva, 2002.
[23] ISO/IEC 24570: Software engineering - NESMA functional size measurement method - Definitions and counting guidelines for the application of function point analysis, 2nd ed., ISO, Geneva, 2018.
[24] ISO/IEC 29881: Information technology - Systems and software engineering - FiSMA 1.1 functional size measurement method, ISO, Geneva, 2010.
[25] ISO/IEC 19761: Software engineering - COSMIC: a functional size measurement method, ISO, Geneva, (2011, reviewed and confirmed in 2019).
[26] H. Sneed, Purpose of Software Measurement, Software Measurement News, Volume 26, Number 1, March 2021.
[27] C. Symons, Why COSMIC functional Measurement?, Measurement News, Volume 25, Number 2, September 2020.
[28] C. Commeyne, A. Abran, and R. Djouab, Effort estimation with story points and COSMIC function points - an industry case study, Software Measurement News, Volume 21, Number 1, 2016.
[29] M. Salmanoglu, T. Hacaloglu, and O. Demirors, Effort estimation for agile software development: comparative case studies using COSMIC functional size measurement and story points, in: Proceedings of the 27th IWSM-Mensura, ACM, Göteborg, Sweden, 2017.
[30] S. Di Martino, F. Filomena, C. Gravino, and F. Sarro, Web Effort Estimation: Function Point Analysis vs. COSMIC, Information and Software Technology 72 (2016) 90-109. doi: 10.1145/3143434.3143450.

[31] C. Symons, A. Abran, C. Ebert, and F. Vogelezang, Measurement of Software Size: Advances Made by the COSMIC Community, in: Proceedings of the IWSM-Mensura, IEEE, Berlin, Germany, 2016, pp. 75-86. doi: 10.1109/IWSM-Mensura.2016.021.
[32] A. Tarhan, B. Özkan, and G.C. İçöz, A Proposal on Requirements for COSMIC FSM Automation from Source Code, in: Proceedings of the IWSM-Mensura, IEEE, Berlin, Germany, 2016, pp. 195-200. doi: 10.1109/IWSM-Mensura.2016.038.
[33] A. Tarhan, and M.A. Sag, COSMIC Solver: A Tool for Functional Sizing of Java Business Applications, Balkan Journal of Electrical & Computer Engineering 6(1) (2018).
[34] E. Ungan, C. Hammond, and A. Abran, Automated COSMIC Measurement and Requirement Quality Improvement Through ScopeMaster® Tool., in: Proceedings of the IWSM-Mensura, Beijing, China, 2018.
[35] S. Bagriyanik, and A. Karahoca, Automated COSMIC Function Point measurement using a requirements engineering ontology, Information and Software Technology 72 (2016) 189-203.
[36] T. Zaw, S.Z. Hlaing, M. M. Lwin, and K. Ochimizu, An Automated Software Size Measurement Tool based on Generation Model using COSMIC Function Size Measurement, in: Proceedings of the International Conference on Advanced Information Technologies (ICAIT), IEEE, Yangon, Myanmar, 2019, pp. 268-273.doi: 10.1109/AITC.2019.8920991.
[37] A. Sellami, H. Hakim, A. Abran, and H. Ben-Abdallah, A measurement method for sizing the structure of UML sequence diagrams, Information and Software Technology 59 (C) (2015) 222–232.
[38] H. Hakim, A. Sellami, H. Ben-Abdallah, and A. Abran, Improving the Structural Size Measurement Method Through the Assessment of Nested (Multi-Level) Control Structures in UML Sequence Diagram, in: Proceedings of the IWSM-Mensura, Mexico city, Mexico, 2020.
[39] S. Abrahão, L. De Marco, F. Ferrucci, J. Gomez, C. Gravino, and F. Sarro, Definition and evaluation of a COSMIC measurement procedure for sizing Web applications in a model-driven development environment, Information and Software Technology 14 (2018) 144-161.
[40] G. De Vito, F. Ferrucci, and C. Gravino, Design and automation of a COSMIC measurement procedure based on UML models, Software and Systems Modeling 19 (2020) 171–198.
[41] A. Filippova, E. Trainer, and J.D. Herbsleb, From diversity by numbers to diversity as process: Supporting inclusiveness in software development teams with brainstorming, in: Proceedings of the 39th International conference on software engineering, IEEE, Buenos Aires, Argentina, 2017, pp. 152–163.doi: 10.1109/ICSE.2017.22.