Functional Size Measurement for X86 Assembly Programs

Donatien Koulla Moulla [1,2], Abdel Aziz Kitikil [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] *Ecole Centrale d'Electronique (ECE), Lyon, France*
[4] *École de Technologie Supérieure, 1100, rue Notre-Dame Ouest, Montréal, Québec, H3C 1K3, Canada*

Abstract

Functional size measurement (FSM) provides a reliable and objective way to measure productivity and estimate the effort required for software activities. FSM automation, compared to manual measurement, enables to reduce errors, increase consistency, and improve the efficiency of the measurement process. This paper presents a COSMIC-based automated FSM for X86 assembly programs widely used in personal computers and servers. The proposed approach applies the COSMIC-ISO 19761 rules for sizing Functional User Requirements (FUR) of an X86 assembly program by identifying functional processes, data groups and data movements. An automated measurement prototype tool is also presented including its architecture, its measurement algorithm, the verification protocol used and its measurement accuracy on the measurement of two assembly programs. The prototype tool can be useful to organizations and practitioners in the embedded systems industry.

Keywords

Functional size measurement, COSMIC – ISO 19761, X86 assembly programs, X86 architectures, Software measurement automation, Automation tool

1. Introduction

In the field of software engineering, as in other engineering domains, mastering and evaluating software development is a concern for both practitioners and industry and, through effective measurement, organizations can learn to perform software work more effectively with better estimation, planning, monitoring and controls [1, 2, 3]. The concept of measurement, in its metrology sense in [4], includes a "measurement method," "application of a measurement method," and "measurement results".

Sizing software is an important basis for measuring productivity and estimating the effort required for software activities as part of good management. There are several options for sizing software such as lines of code, Use case Points, Object Points, Story Points, and functional size with functional size measurement (FSM) method, including COSMIC Function Points – ISO 19761. Although there is a large body of literature on software measurement, sizing software projects remains a challenging activity.

Many automatable COSMIC FSM procedures have been proposed in the literature for different domains e.g.: IoT [5], aeronautical [6, 7], automotive [8, 9], and quantum computing [10].

Two main software artifacts are generally used to measure the functional size of a piece of software [11]: requirement documents in the early development stages and lines of code once the code has been developed. Automating FSM can help organizations with large software projects to generate measurement results quickly. Compared to manual measurement, automation enables to reduce errors, increase consistency, and improve the efficiency of the measurement process.

IWSM-Mensura, September 14–15, 2023, Rome, Italy
EMAIL: moulldk@unisa.ac.za; kitikilabdelaziz@gmail.com; mnkane@unisa.ac.za; hsoubra@ece.fr; alain.abran@etsmtl.ca
ORCID: [orcid.org/0000-0001-6594-8378]


There are two main approaches to automate FSM [11]:

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 [12, 13, 14].

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 a 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 [15, 16, 17, 18].

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

- The requirements are often incomplete and ambiguous. The requirements do not describe the full scope of the functionality of the software with all the necessary functional details [19]. As a project progresses, the requirements will be detailed and changed as the project moves through the life cycle.
- FSM methods are technology-independent and an automation design is required to handle the specifics of each programming language.

To address the above challenges, this study provides the first automated FSM from the X86 assembly programs, selected for being widely used in personal computers and servers [20]. The current study focuses on the design and implementation of an automated FSM for X86 assembly 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. Section 3 presents the proposed automated measurement approach for sizing X86 assembly programs using the COSMIC – ISO 19761. Section 4 presents the automated measurement prototype. Section 5 concludes the paper with directions for future research.

2. Relate work

2.1. Sizing software with COSMIC – ISO 19761

In recent decades, numerous FSM methods have emerged, including FPA [21], MkII [22], NESMA [23], FisMA [24], and COSMIC [25]. One of the primary motivations behind measuring software size is typically to estimate the effort or cost associated with its development. Compared with other options, the COSMIC FSM method is designed according to basic software engineering and metrology principles, is technology-independent [26], and is standardized ISO – 19761 [25]. The ISO 19761 standard describes the activities required to measure the size of software projects in COSMIC function points (CFP). Several studies have shown the strengths of COSMIC as an FSM method. For instance, it has been shown to perform better than Story Points when it comes to analyze objectively productivity and in building effort estimation models within an agile context [27, 28]. A number of the advances made by the COSMIC community over the years in software sizing and effort estimation are reported in Symons et al. [29].

Huijgens et al. [30] carried out a comprehensive survey involving 336 FSM specialists who agreed that automated FSM from source code is important but challenging. The majority of respondents think that automated FSM has the potential to 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.

This motivated the current choice of COSMIC as a measurement method to design automated functional size measurements for X86 assembly programs.

The COSMIC FSM method defines what is a functional process from the user point of view where software users can be human, hardware devices and other software applications. It also defines four types of data movement that are used by functional processes within a software:

1. ENTRY: A data movement that moves a data group from a functional user across the software 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 software 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.

The flexibility of the COSMIC method makes it possible to measure any type of software and is used around the world. The COSMIC guidelines and measurement case studies are freely available at https://cosmic-sizing.org.

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 requirements and source code analyses. The study in [31] proposed a programming language compiler based on COSMIC FSM. However, further tests are needed to improve the accuracy of the compiler. Moulla et al. [3] proposed an automated FSM for the C programming language using regular expressions. Soubra et al. [32] presented an approach for a theoretical 'universal' tool based on COSMIC ISO 19761 for the automated measurement of software written in different programming languages. Their study 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.

Darwish et al. [33] proposed an FSM procedure using 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. Tarhan et al. [34] developed an operational scenario for automated FSM from software code based on a comprehensive analysis and insights gained from their previous studies [15, 16, 17]. Additionally, they proposed a set of requirements that should be taken into account when implementing the automation process. The authors replicated the study in [35] by developing a tool called "COSMIC Solver" specifically tailored for Java Business Applications (JBA), and their findings revealed that there was a convergence of 77% between CFPs measured manually and the ones measured automatically. Chamkha et al. [18] introduced a "JavaCFP" plugin tool to measure the CFP of Java source code being developed. The proposed measurement tool serves multiple purposes, including controlling the completeness of the implemented functionality against specified requirements, identifying deviations, and generating progress reports regarding the implementation of new functions. Sahab et al. [36] developed a CFP4J library with the objective of automating COSMIC FSM from Java web applications using the Spring Web MVC framework. Their key contribution involves the definition of mapping rules from code and the development of a software library that is openly accessible.

Ungan et al. [37] 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. [38] proposed an ontology model that aims to transform requirements into COSMIC function point method concepts. They also developed a method to automatically measure the functional size of software by leveraging the constructed ontology. To validate their method, they implemented a software application using requirements data from several real projects. Their findings indicated that manual and automated measurement results were in agreement.

Meiliana et al. [14] used the XML structure of the UML sequence diagram to automate the measurements of software size. They used the COSMIC method to measure functional size, while the calculation of structural size was based on the control structures present in the sequence diagrams.

Zaw et al. [39] introduced an automated tool for measuring software size, which incorporated a generation model derived from UML, SysML, and Petri nets. They also presented general mapping rules between the COSMIC FSM and a generation model, enabling the measurement of software size.

Sellami et al. [40] designed an extended sizing method that takes into account the structural aspects of a sequence diagram to quantify its size. These functional and structural sizes can subsequently be employed as distinct independent variables to improve the models used for effort estimation. 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 [41].

Abrahão et al. [42] defined a measurement procedure (OO-HCFP) for Object-Oriented Hypermedia method (OO-H) web applications using the COSMIC method. 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. [43] 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 have proposed, to the best of our knowledge, an automated measurement solution for X86 assembly programs.

3. COSMIC FSM automation approach

This section presents an overview of X86-assembly program structure and the proposed measurement approach.

3.1. Overview of X86 assembly program

Figure 1 represents the general structure of an X86 instruction.

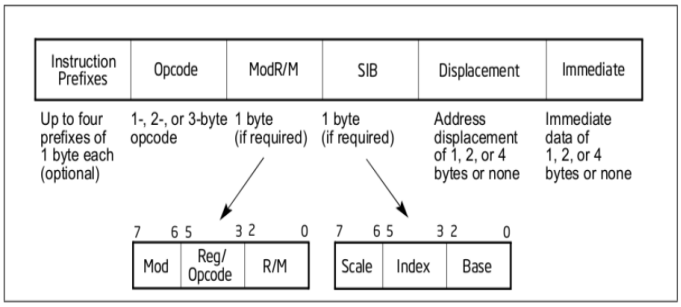

Figure 1: X86 instruction format [44]

An X86 instruction consists of Instruction Prefixes, Operation code (Opcode), Mode Register / Memory (ModR/M), Scale Index Base (SIB), Displacement, and Immediate data blocks.

Instruction prefixes are used to modify the behavior or interpretation instructions, including arithmetic and logical instructions, data movement instructions, and control transfer instructions. The Opcode serves as a fundamental building block of the instruction set, as it defines the set of operations that can be performed by the processor. The ModR/M (Mode Register / Memory) byte is used as part

of the instruction encoding to specify the addressing mode and operands involved in memory operations. By combining the ModR/M byte with the opcode, the processor can determine the specific operation, the addressing mode, and the operands required to execute an instruction accurately. The Scale Index Base (SIB) byte is used in conjunction with the ModR/M (Mode Register / Memory) byte to calculate the effective address of memory operands in instructions that involve scaled index addressing. Displacement refers to an offset or a displacement value used in memory addressing modes. It is an additional value added to a base address or an index register value to calculate the effective address of a memory operand. Immediate data refers to immediate values or constants that are directly included as part of an instruction. Immediate data is used to provide operands or additional parameters for instructions, without requiring a separate memory or register access.

There are several variations of the X86 architecture; this study used the X86-32 Instruction Set Architecture (ISA) version which includes hundreds of different instructions to perform a variety of tasks, ranging from basic arithmetic operations to flow control instructions and advanced multimedia operations.

3.2. The proposed approach: mapping COSMIC to X86 assembly program

In this section, we applied the COSMIC rules for sizing Functional User Requirements (FUR) of an X86 assembly program.

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

- Functional processes: The functional process is represented by the processing unit. The processing unit is responsible for carrying out the tasks submitted by the control unit for each instruction.
- Data groups: A data group consists of a unique set of data attributes that describes a single object of interest. We assumed that each mnemonic 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 X86 Instruction.

Table 1
Rules for identifying data movements in a X86 assembly program

COSMICS Concepts	Elements of X86 program
ENTRY data movement	Data movement that moves a group of data from the control unit across the software boundary to the processing unit.
EXIT data movement	Data movement that moves a group of data from the processing unit across the software boundary to the control unit.
READ data movement	Data movement that moves a group of data from registers to the processing unit that requires it.
WRITE data movement	Data movement that moves a group of data from the processing unit to the registers.

4. Measurement tool prototype

4.1. Tool architecture and description of the algorithm

The tool prototype is aimed at automating the COSMIC FSM from a X86 assembly program file, visualizing it, and saving the measurement results. The study in [45] identified 12 mnemonics that occurred most frequently out of a total of 843 mnemonics observed in 9,337 C/C++ applications and libraries in the Ubuntu 16.04 GNU/Linux distribution. In this study, we used the mostly used mnemonics including arithmetic instructions (ADD, SUB, INC, DEC, NEG, MUL, DIV), logical instructions (AND, XOR, OR), conditional statements (JMP, JE, JNE, JL, JLE, JG, JGE, JZE, JZ, JNZ), transfer instruction (MOV), and comparison instruction (CMP). The general architecture of the tool consists of three modules (Figure 2).

1. Graphical user interface (GUI) module: This module allows a user to select a X86 assembly program file to measure, visualize, and save the measurement results.
2. Filtering and data grouping instructions: this module sorts and groups instructions based on their category or addressing mode. It consists of two sub-modules: RegroupeCateg and RegroupeModeAddr.
   - The sub-module RegroupeCateg takes the .asm file as input and groups the instructions according to their categories. For this sub-module, we defined two functions: *GetListeInstruction()* (allows retrieving all the instructions contained in the .asm file) and *GetCateInstruction()* (allows grouping instructions based on their category).
   - For each category, the RegroupeModeAddr sub-module groups instructions based on the number of operands, and the addressing mode. For this sub-module, we defined five functions: *GetGroupeModeAdrIT(), GetGroupeModeAdrIL(), GetGroupeModeAdrIA(), GetGroupeModeAdrIC(), and GetGroupeModeAdrIBranch()* for transfer instructions, logical instructions, arithmetic instructions, comparison instructions and conditional statement respectively.
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 identified CFP are aggregated. This module is composed of two sub-modules, one to assign numerical values to each data movement and the second one to aggregate all the identified CFP. For this module, we defined six functions: *EstimeIT(), EstimeIL(), EstimeIC(), EstimeIA(), EstimeIBranch()*, and *GetResult()*.

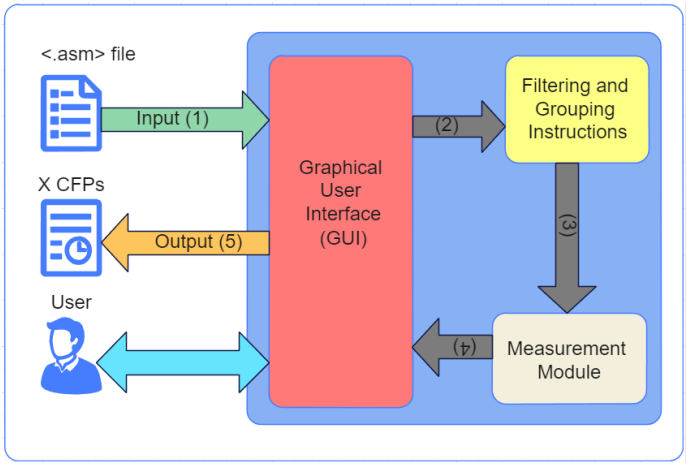

Figure 2: Architecture of the COSMIC-X86 tool prototype

The algorithm describing the tool functions is shown in Figure 3.

Algorithm: Measurement process

Input: asm. file
Outputs: Size
Begin
{**global variables:** *ListeInstruction, CategorieInstruction, InstructionModeAddrIT, InstructionModeAddrIL, InstructionModeAddrIA, InstructionModeAddrIC, InstructionModeAddrIBranch, ListeEstimeIT, ListeEstimeIL, ListeEstimeIA, ListeEstimeIC, ListeEstimeIBranch, Size*

ListeInstruction ← GetListeInstruction (x.asm);
CategorieInstruction ← GetCateInstruction (ListeInstruction);
InstructionModeAddrIT ← GetGroupeModeAdrIT (CategorieInstruction.ListeInstructionIT);
InstructionModeAddrIL ← GetGroupeModeAdrIL (CategorieInstruction.ListeInstructionIL);
InstructionModeAddrIA ← GetGroupeModeAdrIA (CategorieInstruction.ListeInstructionIA);
InstructionModeAddrIC ← GetGroupeModeAdrIC (CategorieInstruction.ListeInstructionIC);
InstructionModeAddrIBranch ← GetGroupeModeAdrIBranch (CategorieInstruction.ListeInstructionIBranch);
ListeEstimeIT ← GetEstimeIT (InstructionModeAddrIT);
ListeEstimeIL ← GetEstimeIL (InstructionModeAddrIL);

```
ListeEstimeIA ← GetEstimeIA (InstructionModeAddrIA);
ListeEstimeIC ← GetEstimeIC (InstructionModeAddrIC);
ListeEstimeIBranch ← GetEstimeIBranch (InstructionModeAddrIBranch);
Size ← GetResult (ListeEstimeIT, ListeEstimeIC, ListeEstimeIA, ListeEstimeIL, ListeEstimeIBranch);
}
End
```

Figure 3: Algorithm of the COSMIC measurement process

4.2. Validation of the prototype

The automated measurement tool prototype was implemented using Java programming language and Eclipse framework. As a test case example, two source code of assembly programs were used, namely: Fibonacci, and Consecutive sum programs. There are two ways to obtain the assembly code from a C/C++ file:

- From the C/C++ source code, the assembly code can be generated by the compiler;
- From the executable file, the 'objdump' can be used to disassemble the file and obtain the code.

Figure 4 shows the source code of the two assembly programs.

```
Fibonacci assembly program

push ebp
mov ebp, esp
push ebx
mov eax, [ebp + 8] ; f0
mov edx, [ebp + 12] ; f1
mov ecx, [ebp + 16] ; n
test ecx, ecx
jz .end
.while:
mov ebx, edx ; tmp = f1
add edx, eax ; f1 = f1 + f0 => f2, f3, ...
mov eax, ebx ; f0 = tmp => f1, f2, ...
dec ecx ; --n
jnz .while
.endwhile:
.end:
pop ebx
mov esp, ebp
pop ebp
ret
```

(a)

```
Consecutive sum assembly program

 xor eax, eax ;i1
 mov ecx, 1 ;i2
.for:
 cmp ecx, 10 ;i3
 jg .end_for ;i4
 add eax, ecx ;i5
 inc ecx
 jmp .for ;i7
.end_for:
 push eax ;i8
 push dword msg ;i9
```

(b)

Figure 4: Source code of Fibonacci (a), and Consecutive sum (b) assembly programs

The Fibonacci program calculates the numbers of the Fibonacci sequence up to a given term or up to a certain maximum number. The consecutive sum program calculates the cumulative sum of the first 10 numbers and displays the result.

While most of the papers published on FSM automation only mention verification on the total of CFP, this study uses the verification protocol proposed by Soubra et al. [46] to evaluate the accuracy of the tool prototype which is based on each measured function (mnemonic). Tables 3 and 4 present the comparison between the manual measurement results and the automated measurement results for the Fibonacci and Consecutive sum programs respectively using the COSMIC ISO 19761. Three people performed the manual measurements, and one of them was a certified COSMIC measurer.

Table 2

Fibonacci: Comparison of the CFP manual and automated measurement results

Fibonacci Asm instructions	CFP manual measurement	CFP automated measurement	Measurement Accuracy
push	(3E+3R+2W=8)	(5E+3R+2W=10)	75%
mov	(5E+2R+1W=8)	(5E+2R+1W=8)	100%
push	(3E+3R+2W=8)	(5E+3R+2W=10)	75%
mov	(5E+2R+1W=8)	(5E+2R+2W=9)	87.5%
mov	(5E+2R+1W=8)	(5E+2R+2W=9)	87.5%
mov	(5E+2R+1W=8)	(5E+2R+2W=9)	87.5%
test	(5E+2R+2W=9)	-	0%
jz	(3E+5R+2W=10)	(2E+4R+2W=8)	80%
mov	(5E+2R+1W=8)	(5E+2R+1W=8)	100%
add	(5E+2R+1W=8)	(5E+2R+1W=8)	100%
mov	(5E+2R+1W=8)	(5E+2R+1W=8)	100%
dec	(5E+2R+1W=8)	(5E+2R+1W=8)	100%
jnz	(3E+5R+2W=10)	(2E+4R+2W=8)	80%
pop	(5E+3R+2W=10)	(5E+3R+2W=10)	100%
mov	(5E+2R+1W=8)	(5E+2R+1W=8)	100%
pop	(5E+3R+2W=10)	(5E+3R+2W=10)	100%
ret	(1E+4R+2W=7)	-	0%
Total	**144**	**131**	**80.74%**

Table 4

Consecutive sum: Comparison of the CFP manual and automated measurement results

Consecutive sum Asm instructions	CFP manual measurement	CFP automated measurement	Measurement Accuracy
xor	(5E+2R+1W=8)	(5E+2R+1W=8)	100%
mov	(5E+2R+1W=8)	(5E+2R+1W=8)	100%
cmp	(5E+2R+2W=8)	(5E+2R+2W=8)	100%
jg	(3E+5R+2W=10)	(3E+5R+2W=10)	100%
add	(5E+2R+1W=8)	(5E+2R+1W=8)	100%
inc	(5E+2R+1W=8)	(5E+2R+1W=8)	100%
jmp	(1E+4R+2W=7)	(1E+4R+2W=7)	100%
push	(3E+3R+2W=8)	(3E+3R+2W=8)	100%
push	(3E+3R+3W=9)	(3E+3R+3W=9)	100%
Total	**75**	**75**	**100%**

From Tables 3 and 4, we can observe that the prototype tool presents an average accuracy of 80.74% and 100% for Fibonacci and Consecutive sum assembly programs, respectively. In Table 3, we can observe that the automated measurement missed two sets of instructions: test, and ret. This is due to the prototype current limitations because the two instructions were not used (defined) in this study. Additionally, we can also observe small size differences in various instructions, and only Entries data movements are responsible for this difference. In the Fibonacci program, the mov instructions have different sizes (sometimes as 9 CFP, others as 8 CFP) because these instructions are different (mov ebp, mov eax, mov edx, mov ecx, mov ebx, mov esp).

When the verification of the accuracy is done only on the total of CFP of each of the two assembly programs, the tool prototype presents an accuracy of 90. 97% and 100% for Fibonacci and Consecutive sum assembly programs, respectively. The accuracy values show a higher convergence between CFP measured manually and those obtained automatically.

5. Conclusion

Several COSMIC based FSM measurement automations have been proposed; however, to the best of our knowledge, none has tackled automated measurement solutions for X86 assembly programs. This study proposed a measurement approach for sizing X86 assembly programs using the COSMIC method.

An automated measurement tool prototype was introduced, with a test example of two assembly programs as a case study. The measurement results for the automated measurement tool prototype had an average accuracy of 80.74% and 100% for Fibonacci and Consecutive sum assembly programs respectively compared to those obtained manually.

Such work on the COSMIC FSM automation of X86 assembly programs can be useful for organizations and practitioners, specially from the embedded systems industry.

In terms of limitations this research work, while it proposes an automated measurement of COSMIC functional size for X86 assembly programs, the test case example relates exclusively to two X86 assembly programs and their sizes are relatively small. Thus, additional tests are required to verify the accuracy of the tool. We plan to use more mnemonics (instructions), and to extend our study to large projects and different types of application to evaluate the accuracy of the tool. The proposed prototype tool did not correctly identify Entries data movements in some instructions. However, this limitation will be addressed in future studies. We also plan to use the X86-64 Instruction Set Architecture (ISA) version. To further investigate limitations or threats to validity of automated measurement from X86 assembly program, we plan to compare the size measured based on the user requirements of the two assembly programs with those obtained from the automated tool.

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] D.K. Moulla, Oumate, E. Mnkandla, H. Soubra, and A. Abran, Automated COSMIC Function Points Measurement for C Program Using Regular Expressions, in: Proceedings of the 31st International Workshop on Software Measurement and 16th International Conference on Software Process and Product Measurement - IWSM-Mensura conference, Izmir, Turkey, 2022.

[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. Soubra, and A. Abran, Functional size measurement for the internet of things (IoT) an example using COSMIC and the arduino open-source platform, in: Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement – IWSM-Mensura, Göteborg, Sweden, 2017, pp. 122-128.

[6] H. Soubra, L. Jacot, and S. Lemaire, Manual and Automated Functional Size Measurement of an Aerospace Realtime Embedded System: A Case Study based on SCADE and on COSMIC ISO 19761, International Journal of Engineering Research and Science & Technology, 4 (2015), 79-100.

[7] H. Soubra, A. Abran, and M. Sehit, Functional Size Measurement for Processor Load Estimation in AUTOSAR, Joint 25th International Workshop on Software Measurement & 10th International Conference on Software Process and Product Measurement - IWSM-Mensura, Cracow (Poland),

Oct. 5-7, 2015, Springer International Publishing (Switzerland), Lecture Notes on Business Information Processing – LNBIP230, pp. 1-16, 2015. DOI: 10.1007/978-3-319-24285-9_8.
[8] Soubra, Hassan, and Toufik Azib. "Functional Size Measurement for Energy Needs early Estimation in Autonomous Drones." In IWSM-Mensura, pp. 48-53. 2018.
[9] Soubra, Hassan, and Khaled Chaaban. "Functional size measurement of electronic control units software designed following the autosar standard: A measurement guideline based on the cosmic iso 19761 standard." In 2012 Joint Conference of the 22nd International Workshop on Software Measurement and the 2012 Seventh International Conference on Software Process and Product Measurement, pp. 78-84. IEEE, 2012.
[10] K. Khattab, H. Elsayed, and H. Soubra, Functional Size Measurement of Quantum Computers Software, in: Proceedings of the 31st IWSM-Mensura, Izmir, Turkey, 2022.
[11] H.V. Heeringen, Automated Function Points, the Game Changer!, IT Conference – ISBSG, 2020.
[12] 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.
[13] T.M. Fehlmann, and E. Kranich, COSMIC functional sizing based on UML sequence diagrams, in: Proceedings of the MetriKon, Kaiserslautern, Germany, 2011.
[14] 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 International Conference on Cybernetics and Computational Intelligence (CyberneticsCom), IEEE, Phuket, Thailand, 2017, pp. 24–28. doi:10.1109/CYBERNETICSCOM.2017.8311709.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[19] A. Abran, Software Estimation: How to use ISBSG data for early software sizing?, IT Conference – ISBSG, 2020.
[20] E. Blem, J. Menon, and K. Sankaralingam, Power struggle: Revisiting the RISC vs. CISC debate on contemporary arm and x86 architectures, in: Proceedings of the 19th International Symposium on High Performance Computer Architecture (HPCA), Shenzhen, China, 2013, pp. 1–12. https://doi.org/10.1109/HPCA.2013.6522302.
[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] cosmic-sizing.org, The benefits of COSMIC software sizing, https://cosmic-sizing.org/cosmic-sizing/intro/benefits-of-cosmic/, 2022.

[27] 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.
[28] M. Salmanoglu, T. Hacaloglu, and O. Demirors, "Effort estimation for agile software development: comparative case studies using COSMIC functional size measurement and story points," 27th International Workshop on Software Measurement & MENSURA Conference - IWSM-Mensura, Göteborg, Sweden, 2017, ACM, pp. 41-49.
[29] 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.
[30] 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.
[31] Y. Attallah, and H. Soubra, Towards a COSMIC FSM Programming Language Compiler, in: Proceedings of the 31st IWSM-Mensura, Izmir, Turkey, 2022.
[32] H. Soubra, Y. Abufrikha, and A. Abran, Towards Universal COSMIC Size Measurement Automation, in: Proceedings of the 30th IWSM-Mensura, Mexico City, Mexico, 2020.
[33] A. Darwish, and H. Soubra, COSMIC Functional Size of ARM Assembly Programs, in: Proceeding of the 30th IWSM-Mensura. Mexico City, Mexico, 2020.
[34] 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.
[35] 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).
[36] 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.
[37] 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.
[38] S. Bagriyanik, and A. Karahoca, Automated COSMIC Function Point measurement using a requirements engineering ontology, Information and Software Technology, 72 (2016), pp. 189-203.
[39] 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.
[40] 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), pp. 222–232.
[41] 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 Diagrams, in: Proceedings of the 30th IWSM-Mensura, Mexico City, Mexico, 2020.
[42] 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), pp. 144-161.
[43] 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), pp. 171–198.
[44] Intel, Intel® 64 and IA-32 Architectures Software Developer's Manual Combined (Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4), 2023. URL: Intel® 64 and IA-32 Architectures Software Developer Manuals

[45] A. Akshintala, B. Jain, C.-C. Tsai, M. Ferdman, and D. E. Porter, X86-64 instruction usage among c/c++ applications, in: Proceedings of the 12th International Conference on Systems and Storage, ACM, Haifa, Israel, 2019, pp. 68–79. https://doi.org/10.1145/3319647.3325833.

[46] H. Soubra, and A. Abran, Verifying the Accuracy of Automation Tools for the Measurement of Software with COSMIC – ISO 19761 including an AUTOSAR-based Example and a Case Study, Joint Conference of the 24th International Workshop on Software Measurement & 9th International Conference on Software Process and Product Measurement - IWSM-Mensura, Rotterdam (Netherlands), Oct. 6-8, 2014. IEEE Press, pp. 23-31. DOI 10.1109/IWSM.Mensura.2014.26