Towards a COSMIC FSM Programming Language Compiler

Youssef Attallah[1], Hassan Soubra[1]

[1]German University in Cairo, New Cairo, Egypt

Abstract

COSMIC ISO 19761 is considered a second generation of functional size measurement methods. It has met its primary design goal to be of practical value in software project performance measurement and in estimating activities, both in research and industry. The method's partial and full automation -with very high accuracy results- helped in accelerating its adoption and widespread use. This paper presents a new approach for automatically and rapidly obtaining COSMIC functional size of Software requirements by introducing a new means to directly express and implement these requirements using the COSMIC vocabulary. An example is also presented as a proof of concept of the approach proposed.

Keywords

COSMIC ISO 19761, Functional Size Measurement, Measurement Automation, Compiler, Programming languages

1. Introduction

The concept of Functional Size measurement (FSM) has become a key factor in Software management, hence it is crucial to be able to measure it accurately and efficiently. Automating FSM is important for organizations needing to measure a large number of projects within a short time frame, provided, of course, that the results automatically generated are accurate, particularly when such measurement is based on an international measurement standard.

The COSMIC FSM method [1] was designed to overcome the weaknesses of the 1st generation methods. It defines a three-phase process to measure the functional size of any software artifact. To speed up the measurement process and limit the possibility of human errors, there has been several attempts to automate COSMIC based FSM procedures, particularly in industry related projects.

Automating functional size measurement is a high-importance task when its reported benefits and the potential for wider usage are considered. Despite the challenges, one can observe the increasing number of studies that deal with automating COSMIC FSM [2].

This paper examines first the scientific literature covering topics related to the automation of COSMIC based FSM procedures using different Software artifacts, e.g.: source code, textual requirements or a models. Second, the implementation of a Source-to-Source compiler for a COSMIC programming language that acts as an executable binary, while also automatically

IWSM/MENSURA 22, September 28–30, 2022, Izmir, Turkey

youssef.attallah@student.guc.edu.eg (Y. Attallah); hassan.soubra@guc.edu.eg (H. Soubra)

obtaining the functional size without the need of any human intervention, is also presented in this paper.

This paper is organized as follows: Section 2 examines the literature review on COSMIC automation. Section 3 presents overviews of COSMIC, Compilers and Source-to-source compilers. Section 4 presents the proposed approach and its implementation. Conclusions follow in section 5.

2. Literature Review

This section presents the literature review of existing procedures and tools that automate COSMIC FSM. Over 30 papers have matched the criteria of keywords related to automated method or a tool for COSMIC functional software size. The search engine used for this review is Google Scholar[3].

2.1. Measuring Functional Size using source code

The study in [2] introduced one early tool for COSMIC FSM named $\mu cROSE$, which includes a C++ parser which works for real time C++ projects.

The studies in [4] and [5] proposed a functional size measuring procedure for Model-View-Controller (MVC) applications from source code. [4] presents the mapping rules from code and a publicly available software library to automate COSMIC functional size of Web Java applications that use the Spring MVC framework. The study in[6] proposed to measure software functional size automatically for Java business applications, using a tool named Cosmic Solver. The method accomplishes this aim in two steps; tracing functional process sequences with AspectJ directly from source code and applying measurement rules with reference to the UML sequence diagram.

The study in [7] explained a procedure to automatically measure COSMIC functional size of Java business applications having three-tier architecture, and demonstrated its usage on an measurement example.

The study in [8] provided a critical review of four studies carried out on automating COSMIC FSM from software code. It identified a set of attributes to compare the features of the methods and shared the results from the comparison.

The study in [9] focused on automated measurement of CFP from software code of any JBA (Java Business Application). It introduced a tool called COSMIC Solver, presented the results of a case study on evaluating the performance of the tool from sizing of an open source JBA, and identified opportunities for improving the tool.

The study in [10] presented the JavaCFP tool that can be used as a basis to assist the developers in writing Java Swing applications and generating their COSMIC functional size at any time during the development life cycle. This JavaCFP tool can be used by developers having different programming styles and even when they are not familiar with the COSMIC method.

The study in [11] proposed an FSM procedure based on COSMIC to measure software artifacts expressed in ARM's base 32-bit Assembly code. An FSM automation prototype tool is also introduced.

The study in [12] proposed an approach for a 'universal' tool based on COSMIC for automated measurement of software written in different programming languages. As a proof of concept, [12] presented a prototype tool based on COSMIC and MIPS, with a small-scale validation.

2.2. Measuring Functional Size using Textual Requirements

The study in [13] presented a procedure for measuring automatically the COSMIC functional size from SRS (Software Requirements Specification), taking text as an input.

The study in [14] presented some of the features of the automatic COSMIC size measurement tool ScopeMaster®. To their knowledge this tool is the first commercial tool that performs COSMIC measurement on free form textual requirements.

The study in [15] proposed an automated estimation tool of user stories in order to reduce the problems found in software size estimation. The approach is based on Natural Language Processing and defines a dictionary to map it to the COSMIC method.

The study in [16] introduced the MENSURA tool, which integrates the COSMIC concepts and utilizes them for communicating what users need in terms that the developers could understand and allow to define specific units of size. [16] presented an approach of the automation of the software FSM through a tool that could be used in two periods: from the new outline requirements and once the requirements have been specified.

2.3. Measuring Functional Size using Model-based tools

2.3.1. The AUTOSAR architecture model

The study in [17] presented a guideline for measuring functional size in accordance with the COSMIC ISO 19761 measurement method for ECU Application Software designed following the AUTOSAR architecture.

The study in [18] proposed a three phase verification protocol for automation tools implementing the COSMIC-ISO 19761 measurement method. The proposed protocol uses the representation created to extract all the necessary information to obtain the functional size of the software to be measured using the COSMIC method, the example verification was of an AUTOSAR-based FSM automation prototype tool developed at ESTACA.

The study in [19] introduced an automation prototype tool in JAVA was developed at ESTACA. The inputs are SYMTA/S simulation files that include both AUTOSAR models and ECU processor load information. This tool makes it possible to measure automatically software functional size, in CFP, designed following the AUTOSAR methodology and meta-model.

2.3.2. BPM (Business Process Model)

The study in [20] introduced a script named e-Cosmic, and the core functionality of the script is it reports functional size of the software product from BPM.

The study in [21] presented a functional size measurement procedure called E-FSMP. E-FSMP applies the standard COSMIC measurement method to BPMN(Business Process Model and Notation) diagrams by means of the application of 11 rules. In addition, a tool that automates the

application of E-FSMP has been presented. With this tool, measurement results can be obtained quickly and avoiding precision errors that commonly occurs with manual measurements.

The studies in [22] and [23] introduced UPROM tool which can be used to automatically generate COSMIC FSM using BPM.

The study in [24] presented an RPA (Robotic process automation). A robot from a banking operation was used to illustrate how the measurement of an RPA software can be made by using the COSMIC method by gathering functional user requirements from BPMN, proving the applicability of the COSMIC method measurement process to this type of software.

2.3.3. UML (Unified Modeling Language)

The study in [25] presented an approach and a tool for the automation of functional size measurement with RUP. The design of this tool is based on the direct mapping between COSMIC-FFP and UML concepts and notation. [6] also uses UML sequence diagram as a reference. The study in [26] showed the automated generation method of COSMIC FSM by using generation model based on UML metamodel.

The study in [27] introduced an automated measurement tool that can generate COSMIC FSM from the XML representation of UML, SysML and Petri net by using generation model and mapping rules.

The study in [28] introduced J-UML COSMIC, a tool that supports the proposed measurement procedure from UML and simplifies project monitoring and control, because it automatically obtains accurate measurements.

2.3.4. Other models

The study in [29] proposed formal measurement rules for ROOM(Real-time Object Oriented Modeling language) specifications according to COSMIC-FFP informal definitions given in previous research.

The study in [30] introduced OOmCFP, which is an FSM procedure for object-oriented applications generated in MDA (Model-Driven Architecture) environments. OOmCFP allows the measurement of the functional size in the conceptual models that will be transformed in the generated applications.

The study in [31] proposed a COSMIC based functional size measurement (FSM) procedure for real time embedded software requirements documented using the Simulink modeling tool.

The studies in [32] and [33] proposed a procedure for measuring functional side of Synchronous Languages, which maps COSMIC method to LUSTRE programming language and the LUSTRE based SCADE model.

The study in [34] presented work that provides a first evidence of the viability of measuring requirements expressed via Problem Frames by means of the COSMIC FSM method.

The study in [35] introduced a procedure that maps ORM architecture with COSMIC, the proposed tool starts with resolving files and XML mapping files to identify the business domain objects, and then it finds presentation layer controls (buttons, textboxes, html-tags etc.), which have a triggering action and related events attached to these actions.

To our best knowledge, no COSMIC based programming language that also acts as a tool to

measure COSMIC FSM directly from source code has been proposed proposed. The existing efforts made are semi-automatic external plugins dedicated for some languages to parse the code, and map each keyword to a COSMIC data movement.

3. Background

3.1. Overview of COSMIC FSM

The COSMIC method provides a standardized method for measuring the functional size of software from both the Management Information Systems domain and the real-time domain. The COSMIC method is considered a second generation FSM. This method has been accepted as an International Standard: ISO/IEC 19761 Software Engineering – COSMIC – A functional size measurement method (hereafter referred to as ISO 19761). The latest version of the COSMIC manual available on the COSMIC website is version 5.0. While this release includes a number of refinements, the original principles of the COSMIC method have remained unchanged since they were first published in 1999. The COSMIC method measures the Functional User Requirements (or FUR) of software. The result obtained is a numerical 'value of a quantity' (as defined by the ISO) representing the functional size of the software.

The COSMIC method divides the software's Functional User Requirements (FUR) into four different types of data movements:

- Entry (E): Data Group passing from functional user to a functional process.
- Exit (X): Data Group leaving from a functional user to a functional process.
- Read (R): Reading Data Group from persistent storage.
- Write (W): Writing Data Group into persistent storage.

The COSMIC measurement process has three phases:

- The Measurement Strategy Phase
- The Mapping Phase
- The Measurement Phase

Each phase has a dedicated purpose to help organize the process of FSM. The Measurement Strategy phase is about gathering FUR, and defining the purpose of the measurement. The second phase, the Mapping phase, is meant to map an artifact to the COSMIC Generic Software Model to identify Functional Processes and Data Movements. The third phase, the Measurement phase, is where the actual measurement in CFP (COSMIC Function Point) is done by summing up the number of data movements identified.

3.2. Overview of Compilers

A compiler is system software that translates code written in a high-level programming language to a lower-level programming language.

A compiler is generally divided into two parts: Front end, and Back end. The front end has 3 stages:

- Lexical Analysis: The source code is first read through, then separated into tokens, each of which corresponds to a programming symbol.
- Syntax analysis: This phase organises the list of tokens generated in the preceding phase into a tree-structure called Abstract Syntax Tree (AST), which reflects the program's structure.
- Semantic analysis: This step examines the program's AST to see whether it violates any of the program's consistency constraints.

The back end has also 3 stages:

- Intermediate code generation: The compiler generates a low level or machine like intermediate representation of the code.
- Code optimization: The compiler optimizes that intermediate representation generated in the previous phase
- Code generation: The compiler translates the optimized intermediate representation into targeted machine code.

3.3. Overview of Source-to-source compilers

A Source-to-Source compiler (also called Transcompiler, or Transpiler) translates a small subset of several programming languages into several others. It takes a lot of effort to write a comprehensive back-end for a language. If some generic intermediate representation (IR) to target machine code generator already exists, it can be used as a base for front end. Otherwise, a front end for a target language needs to be created, and instead of creating a back end that reduces the semantics to to primitive target machine code, it translates the original source code to another high level language that can use the existing tools of that language, as an alternate mean, which in this case is called a Source-to-Source compiler. It is basically a front end -Lexical and Semantic analysis- with a high level code translator only, instead of using an IR code generator.

4. Implementation

The first step towards developing a compiler, is to decide on what set of compiler technologies to use. However, for the sake of practicality, we did not use any. A simple source-to-source compiler can be implemented with any plain programming language.

The COSMIC method is better suited to be mapped to a functional programming language, so that each functional process may be represented as a function. Mapping COSMIC FSM should be done in the semantic analysis phase, since it is when we have a full parsable Abstract Syntax Tree to identify the data movements of each functional process.

The main aim is to develop a language having a grammar and keywords in accordance with the COSMIC method, so that anyone who is familiar with the method can instantly recognize the pattern of the language, and start coding.

We used Node.js as the run-time environment of our source-to-source compiler. Node.js has many available modules that can efficiently help in developing the front-end part for the compiler.

As for the back-end of the Source-to-source Compiler, it was deemed more convenient to translate the source code into Javascript to instantly run on Node.js without the need of installing other modules.

4.1. Compiler

A lexer and a parser are needed in the front-end.

4.1.1. Lexer

The generated Lexer takes as input the original source code file of the CPL and generates the tokens to be later used by the parser. The lexer module used is Moo.js [36]. It is a lexer generator which is defined by specifying some regular expressions that define the tokens of the language. Moo.js is faster than other tokenizers and it complements the Nearley.js [37] parser, by enabling it to be instantly used.

4.1.2. Parsing

The generated parser uses the tokens produced by the lexer, goes through the Semantic Analysis phase, and outputs an parsable AST. The parser module used is Nearley.js which is a streaming parser with support for catching errors gracefully and providing all parsings for ambiguous grammars. It is compatible with Moo.js. It comes with tools for creating tests, railroad diagrams and fuzzers from your grammars, and has support for a variety of editors and platforms. It works in both Node.js and the browser.

4.1.3. Code Generation

The compiler takes the AST generated by the parser, traverses it, and accordingly translates it to Javascript on a file, that is directly run on Node.js

4.2. Error Handling

In programming, Error Reporting is needed. Syntax Errors are reported by the parser. Run-time errors are reported by Node.js as the back end runs entirely on Javascript.

4.3. COSMIC Programming Language Overview

A "Hello, world!" program is frequently used to demonstrate the appearance and feel of a programming language. The following is an example of a program written in the COSMIC programming language proposed in this paper:

```
funcproc main() {
    print("Hello, World!")
}
```

The CPL syntax can be summarized into these following statements:

- Comments:

  ```
  #Comment
  ```

- Declaration Statements:

  ```
  write varName = value    #global variable
  varName = value          #local variable
  ```

- Conditional Statements:

  ```
  if (condition) { #codeblock }
  ```

- Loop Statements:

  ```
  while (condition) { #codeblock }
  ```

- Functional Process Definition Statements:-

  ```
  funcproc name (parameters) { #codeblock }
  ```

4.4. Mapping COSMIC FSM to the language

The measurement process is done after the parsing of the source code is done, and an AST is generated, which is parsed to calculate the sum of CFP according to the following mapping rules:-

- Identify 1 functional process (FP) for each defined function in the source file.
- Identify 1 Entry (E) for each data group in each parameter in a FP.
- Identify 1 Exit (X) for a return in a FP.
- Identify 1 Entry (E) for functional process call trigger.
- Identify 1 Entry (E) for main functional process call trigger.
- Identify 1 Read (R) for each global variable read.
- Identify 1 Write (W) for each global variable write.
- Aggregate the CFP for each data movement in a FP to obtain the size of the FP.
- Aggregate the CFP of each FP to obtain the size of the whole file.

4.5. Code Example

```
#write
write a = 100
#write
write b = 200
#write
write c = [1, 2, 3]
funcproc foo() {
    #read
    x = read a
```

```
    if(x == 100){
        i = 0
        while(i<10){
            #write + read
            write a = read a + 100
            i = i + 1
        }
        #exit
        exit "if"
    }
    else{
        #exit (optional)
        exit "else"
    }
}
funcproc main() {   #main entry
    # foo entry
    x = foo()
    #print entry + 2 read + 2 exit
    print("a: "+ read a+ ", b: "+ read b)
    #write
    write c[0] = "one"
    #print entry + read + exit
    print("c: " + read c)
    #print entry + exit
    print(x)
}
```

When applying the mapping rules to the code example, 3 global variable writes are identified, which in this case is our persistent storage, and 2 Functional processes foo() and main() are also identified. When a main FP is defined, we assume there is main FP call trigger. A main() functional process is needed to execute the code. The COSMIC size of the code is 3 (Global Definitions) + 4 (Foo()) + 13 (Main()) = 20 CFP.

As shown in Fig. 1, the compiler runs on Node.js, and takes the file "test.cpl", then parses it, according to the mapping rules, it displays each CFP rule as explained above, then sums it up displaying the total size in CFP. Lastly it automatically translates and runs the Javascript file to display the code output.

5. Conclusion

This paper proposed a programming language compiler based on COSMIC FSM, designed with COSMIC related keywords to be familiar to anyone using COSMIC FSM.

After reviewing the scientific literature on COSMIC automation, Compilers and the COSMIC method are discussed. Next, the approach towards how to map the COSMIC method to the

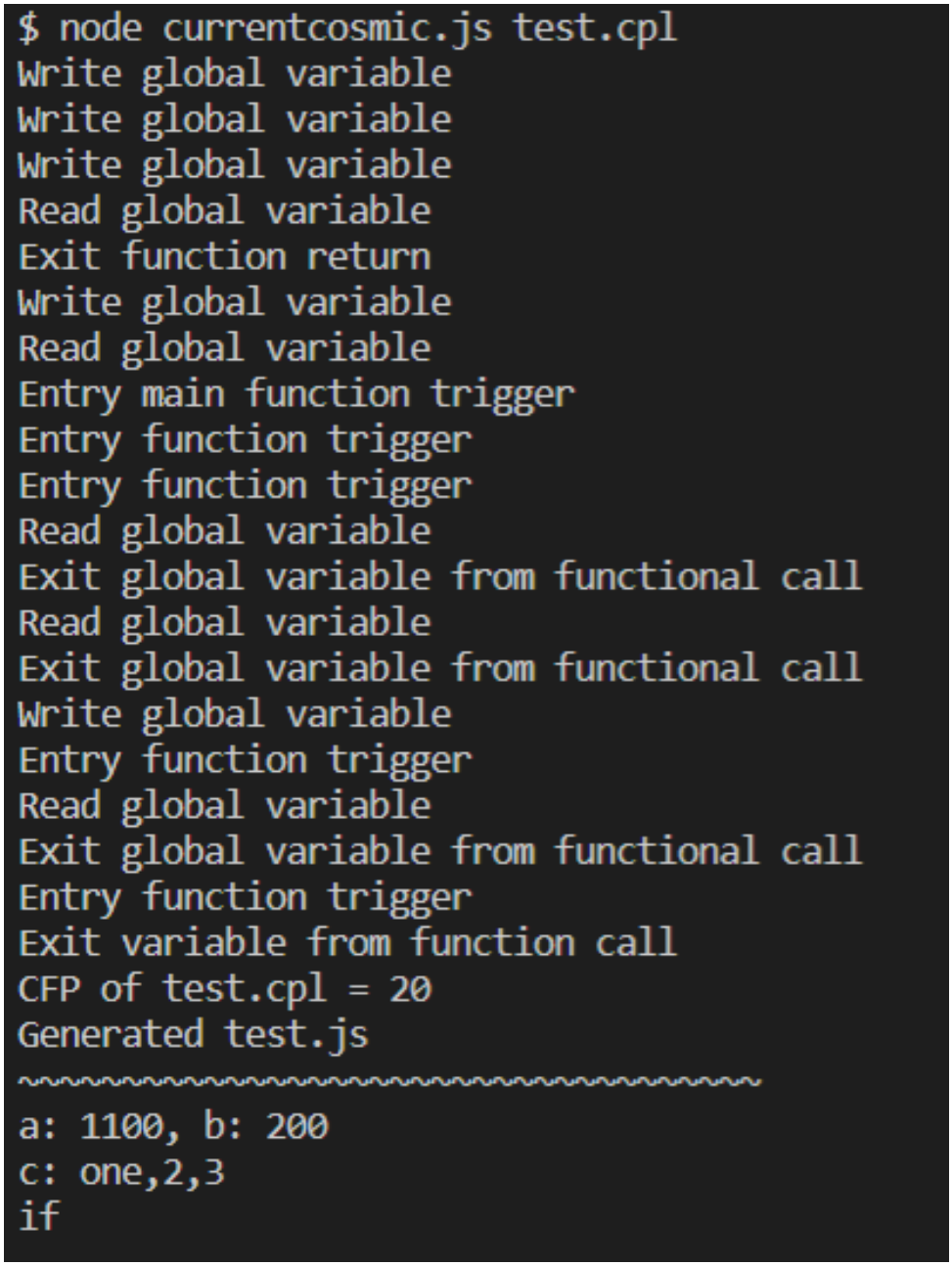

Figure 1: Code output in console.

COSMIC programming language (CPL) compiler is presented. An example of the measurement of a CPL program was presented to better illustrate the mapping approach.

We believe our approach would allow professionals in the Software industry to do away with the need of using additional specific tools dedicated only for size measurement. Our approach

would allow them to use a programming language to implement Software requirements, execute binaries and obtain systematically and automatically COSMIC function size in CFP.

In the future, we aim to further test the accuracy of the compiler. The language should be compared with other existing and in use languages to be able to identify missing features, to better develop the compiler and update the language with in need features to allow it to be used in more practical cases.

References

[1] COSMIC, 2022. URL: https://cosmic-sizing.org/.

[2] H. Diab, F. Koukane, M. Frappier, R. St-Denis, μcrose: automated measurement of cosmic-ffp for rational rose realtime, Information and Software Technology 47 (2005) 151–166.

[3] Google, Google scholar, 2022. URL: https://scholar.google.com/.

[4] C. Quesada-López, A functional size measurement procedure for mvc applications from source code: Design, automation and empirical evaluation (2014).

[5] A. Sahab, S. Trudel, Cosmic functional size automation of java web applications using the spring mvc framework., in: IWSM-Mensura, 2020.

[6] M. A. Sag, A. Tarhan, Measuring cosmic software size from functional execution traces of java business applications, in: 2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, IEEE, 2014, pp. 272–281.

[7] R. Gonultas, A. Tarhan, Run-time calculation of cosmic functional size via automatic installment of measurement code into java business applications, in: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, IEEE, 2015, pp. 112–118.

[8] A. Tarhan, B. Özkan, G. C. İçöz, A proposal on requirements for cosmic fsm automation from source code, in: 2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), IEEE, 2016, pp. 195–200.

[9] A. Tarhan, M. A. SAĞ, Cosmic solver: A tool for functional sizing of java business applications, Balkan Journal of Electrical and Computer Engineering 6 (2018) 1–8.

[10] N. Chamkha, A. Sellami, A. Abran, Automated cosmic measurement of java swing applications throughout their development life cycle., in: IWSM-Mensura, 2018, pp. 20–33.

[11] A. Darwish, H. Soubra, Cosmic functional size of arm assembly programs., in: IWSM-Mensura, 2020.

[12] H. Soubra, Y. Abufrikha, A. Abran, et al., Towards universal cosmic size measurement automation., in: IWSM-Mensura, 2020.

[13] I. Hussain, O. Ormandjieva, L. Kosseim, Mining and clustering textual requirements to measure functional size of software with cosmic., in: Software Engineering Research and Practice, 2009, pp. 599–605.

[14] E. Ungan, C. Hammond, A. Abran, Automated cosmic measurement and requirement quality improvement through scopemaster® tool., in: IWSM-Mensura, 2018, pp. 1–13.

[15] M. Ecar, F. N. Kepler, J. P. S. da Silva, Autocosmic: Cosmic automated estimation and

management tool, in: Proceedings of the XIV Brazilian Symposium on Information Systems, 2018, pp. 1–8.
[16] F. Valdés-Souto, Automated cosmic measurement through mensura® tool (????).
[17] H. Soubra, K. 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, IEEE, 2012, pp. 78–84.
[18] H. Soubra, A. Abran, A. Ramdane-Cherif, Verifying the accuracy of automation tools for the measurement of software with cosmic–iso 19761 including an autosar-based example and a case study, in: 2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, IEEE, 2014, pp. 23–31.
[19] H. Soubra, A. Abran, M. Sehit, Functional size measurement for processor load estimation in autosar, in: Software Measurement, Springer, 2015, pp. 114–129.
[20] M. Kaya, O. Demirors, E-cosmic: a business process model based functional size estimation approach, in: 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, 2011, pp. 404–410.
[21] B. Marín, J. Quinteros, A cosmic measurement procedure for bpmn diagrams., in: SEKE, 2014, pp. 408–411.
[22] B. Aysolmaz, O. Demirörs, Uprom tool: A unified business process modeling tool for generating software life cycle artifacts., in: CAiSE (Forum/Doctoral Consortium), 2014, pp. 161–168.
[23] B. Aysolmaz, O. Demirörs, Unified process modeling with uprom tool, in: International Conference on Advanced Information Systems Engineering, Springer, 2014, pp. 250–266.
[24] F. V. Souto, R. Pedraza-Coello, F. C. Olguín-Barrón, Cosmic sizing of rpa software: A case study from a proof of concept implementation in a banking organization., in: IWSM-Mensura, 2020.
[25] S. Azzouz, A. Abran, A proposed measurement role in the rational unified process and its implementation with iso 19761: Cosmic-ffp, in: Software Measurement European Forum, Rome, Italy, 2004.
[26] T. Zaw, S. Z. Hlaing, M. M. Lwin, K. Ochimizu, A Lightweight Size Estimation Approach for Embedded System using COSMIC Functional Size Measurement, Ph.D. thesis, MERAL Portal, 2017.
[27] T. Zaw, S. Z. Hlaing, M. M. Lwin, K. Ochimizu, An automated software size measurement tool based on generation model using cosmic function size measurement, in: 2019 International Conference on Advanced Information Technologies (ICAIT), IEEE, 2019, pp. 268–273.
[28] G. De Vito, F. Ferrucci, C. Gravino, Design and automation of a cosmic measurement procedure based on uml models, Software and Systems Modeling 19 (2020) 171–198.
[29] H. Diab, M. Frappier, R. S. Denis, Formalizing cosmic-ffp using room, in: Proceedings ACS/IEEE International Conference on Computer Systems and Applications, IEEE, 2001, pp. 312–318.
[30] B. Marín, O. Pastor, G. Giachetti, Automating the measurement of functional size of

conceptual models in an mda environment, in: International Conference on Product Focused Software Process Improvement, Springer, 2008, pp. 215–229.
[31] H. Soubra, A. Abran, S. Stern, A. Ramdan-Cherif, Design of a functional size measurement procedure for real-time embedded software requirements expressed using the simulink model, in: 2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement, IEEE, 2011, pp. 76–85.
[32] H. Soubra, Fast functional size measurement with synchronous languages: An approach based on lustre and on the cosmic iso 19761 standard, in: 2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement, IEEE, 2013, pp. 3–8.
[33] H. Soubra, L. Jacot, 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).
[34] V. Del Bianco, L. Lavazza, Applying the cosmic functional size measurement method to problem frames, in: 2009 14th IEEE International Conference on Engineering of Complex Computer Systems, IEEE, 2009, pp. 282–290.
[35] B. Ozkan, Automated functional size measurement for three-tier object relational mapping architectures, Journal of Software Engineering 21 (2011) 311–338.
[36] Moo.js, 2022. URL: https://github.com/no-context/moo.
[37] K. Chandra, T. Radvan, nearley: a parsing toolkit for JavaScript, https://github.com/kach/nearley, 2014. doi:10.5281/zenodo.3897993.