Software Functional Sizing Automation from Requirements Written as Triplets

Bruel Gérançon, Sylvie Trudel, Roger Kkambou, Serge Robert
Department of Computer Science
Université du Québec à Montréal (UQAM)
Montréal, Canada
e-mail: gerancon.bruel@uqam.ca, trudel.s@uqam.ca, nkambou.roger@uqam.ca, robert.serge@uqam.ca

***Abstract*—The domain of software functional size measurement automation, from software specification documents, has been a research topic over the last years. The literature consulted shows that attempts to automate the process of measuring the software functional size has obtained little success at the industry level. Several tools for automating the measurement of software functional size have been developed according to the Common Software Measurement International Consortium (COSMIC) method (ISO 19761) website and that of International Function Point User Group (IFPUG). However, these tools encountered many flaws, constraints, and limitations. Moreover, the methods, techniques and tools for writing software specification documents used in the industry are far from allowing easily the automation of the measurement of software functional size. In industry, software requirements are often written in natural language, and no technical details are specified. Thus, software requirements are usually incomplete, inconsistent, and prone to ambiguities, and therefore, the analysts can easily make errors of interpretation. Therefore, automating the software functional size measurement is not an easy task. This article introduces a new technique for writing software requirements that could help to automate the functional size measurement process. More precisely, we propose a "triplet approach" for writing software specifications. Furthermore, this procedure is proven, tested, and validated by the development of a new tool for automating the measurement of software functional size, as defined by the COSMIC method. This tool allows to generate triplets (subject, predicate, object) from use cases written in natural language and determines this way the software functional size. Our tool integrates a set of techniques to create a complex artificial intelligence which helps to measure COSMIC function points.**

Keywords-COSMIC; Automation; Functional size; Triplet; Artificial intelligence.

I. INTRODUCTION

The measure of software functional size plays an important role in software engineering, in dealing with new information and in communication technologies (NICT). It is a key factor that allows estimating the effort and the cost of developing software products. Up to now, several estimation methods and approaches have been proposed. As an example, Boehm [5] proposed the COnstructive COst [1] MOdel (COCOMO) method to estimate the cost and duration of software projects. COCOMO is based on the estimation of the number of lines of code to be written for a software. Thus, the number of lines of code corresponds to the physical size of the software. Albrecht [6] proposed a method based on the number of function points, the principle being to identify and quantify user functionalities, thus giving rise to the notion of functional size. Several software measurement methods, such as COSMIC, IFPUG, NESMA, Mark II and FISMA have been proposed and approved by the International Organization for Standardization (ISO). Among the various existing methods and tools, COSMIC is a recent measurement method, developed with the aim of overcoming some limitations of the other methods. A particularity of the COSMIC method is that it can be applied early in the software life cycle and on a set of software components[2].

Although several software measurement methods have been proposed in the literature, the measure of the software functional size is still little used in the industry. The application of software measurement methods remains until now a difficult task [7]. Therefore, the software engineering industry needs tools to automate the functional size measurement process of software [8]. According to the literature consulted, one of the main avenues or research approaches for automating the process of measurement of the functional size of software starts from specifications [1]. In such an approach, the functional size of software is measured from specification documents. However, we can ask: do software requirement writing techniques facilitate the automation of software functional size measurement?

In this article, we will review in section 2 the main techniques and methods for writing software requirements. In section 3, we will describe the limitations of these techniques. Subsequently, we will introduce in section 4, the COSMIC method for sizing software. In section 5, we will introduce our new approach and technique for writing software requirements that could help automate the software functional size measurement process from these specifications, as well as our tool newly developed for supporting the process. Lastly, we will present, in section 6, the results of our research and the future work to be done.

II. TECHNIQUES AND METHODS FOR WRITING SOFTWARE REQUIREMENTS

The automation of the measuring process of software functional size depends necessarily on the mechanism for

 ISBN: 978-1-61208-894-5

writing the software requirements, in other words, it depends on the tools, techniques, and methods used to write the specifications. Several techniques and methods for writing requirements have been proposed in the literature, and these techniques are used in industry. For example, Jacobson et al. [9] propose the technique of "Use Cases" to write software requirements. Beck and West [10], on their part, propose "User Stories" as a technique for writing software requirements. These techniques are texts widely used to identify and record the software functional requirements. By definition, Use Cases are textual descriptions used for document software specifications. They influence all the components related to the software development process, including analysis, design, implementation, and testing. Use Cases describe textually how an actor or user will interact with a software system in order to achieve a goal. The purpose of Use Cases is to identify, describe and document the software functionality, specifying how the system can be used to enable different stakeholders and users to achieve their goals. Note that Use Cases are expressed in natural or technically neutral language, without specifying any technical terminology. User Stories consist in a few lines of text that describe a functionality that the software must offer to allow a given actor or user to achieve a specific goal. User Stories are generally written in natural language and do not include technical terms. One of the major advantages of this approach is that it is centered on the system user.

III. Limitations of Approaches, Methods, and Techniques for Writing Requirements

We present, in this section, the definition of software requirement and the limitations of approaches, methods, and techniques for writing requirements.

A. Software Requirements

Requirements engineering is an important phase of the software development life cycle. By definition, a software requirement is a condition that a software or system must be able to meet. In other words, a software requirement is a capability that a system must exhibit to satisfy a contract between a customer and a supplier. In software engineering, the process of requirements engineering, more specifically the activities of elicitating, analyzing, specifying, verifying, and validating requirements are all important for software engineers. Wiegers [3] defines the elicitation of requirements as a process of exploration, discovery, and invention. It helps to uncover the requirements of a software system by communicating with customers, users, and other stakeholders having an interest in the development of the system [3]. The requirements, once discovered, will be analyzed and described in a software requirements specification document. This document will constitute, after the client's verification and validation, the contractual basis between the software engineers and the client. Requirements engineering is an interdisciplinary activity that acts as an intermediary between the supplier and the customer in order to be able to specify and manage the requirements that must be satisfied by the system. It therefore consists in identifying the goals and the scope of the software and in specifying the context in which the software will be used. As for Boehm [5], the requirements engineering process is the upstream part of the software development process. It allows, among other things, the passage of informal needs expressed by stakeholders into abstract requirements until a software requirements specification is obtained [5]. The upstream requirements, once produced, will be described in a specification. This document (the specifications) is the entry point for the software development phases between the customer and the developers.

B. Limitations of Software Requirements Writing Techniques

The techniques used in industry to write software requirements have several limitations. According to Ambler [13], these techniques increase the risk of failure of software development projects, since they describe a large percentage of software specifications that are never implemented. Additionally, the classical approach to requirements writing fails to solve the problem of requirements semantics, since natural language is inherently ambiguous [4]. Indeed, one of the main limitations of the classic or traditional approach to writing software requirements comes from the fact that the techniques, in particular Use Cases and Use Stories used to specify and describe software requirements, are in natural language, with unnecessary details. So, they do not facilitate the automation of the software functional size measurement process. It is necessary to propose a new technique for writing software requirements that could overcome these difficulties.

IV. The COSMIC Method for Sizing Software

Functional size is based on software functionality. The idea is to quantify the amount of functionality provided to a user for a given software product. This implies that the functional size represents the size of the derived software by quantifying the required user functionality (ISO 14143-1). There are different methods for measuring functional size. Within our research, we adopted the COSMIC method. It involves applying a set of principles, rules, and processes to measure the user's functional requirements of a given software. The result is a numerical value as defined by ISO 19761 and which represents the functional size of the software. With COSMIC, we measure the data movements applicable to data groups manipulated by each functional process. A data movement can be of different types (Entry, Exit, Read, or Write). *Figure 1* summarizes the measurement process of the COSMIC method.

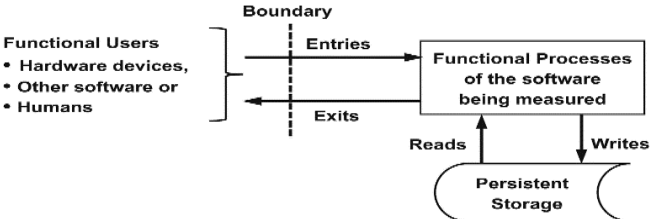

Figure 1. The measurement process of the COSMIC method.

V. Specification of Software Requirements Using our TripletS Structure

We describe, in this section, the software requirements writing technique proposed, and the role of its three (3) components.

A. Triplets Structure

The software requirements writing technique that we propose to facilitate the process of automating the functional sizing of software is a "triplet approach". In other words, each triplet is a single sentence (subject, predicate, object). The subject represents the actor (i.e., the functional user) who interacts with the system. a composite predicate represents the Use case scenario; an atomic predicate represents the methods, transactions or events triggered by the actor (functional, other system, hardware device). Lastly, the object represents a software entity. The goal of the triplet approach is to allow analysts to write or express customers' needs in a simple and effective way with little information. This means that the triplet structure provides an atomic and succinct description of software requirements, expressing the user need with little or no superfluous details. It indeed emphasizes the clarity and brevity of the software requirements. Therefore, the triplet approach could make it easier to perform automatic processing of software requirements, which could automate the software functional size measurement process. Correspondingly, the triplet structure is a requirement writing technique that could complement Use Cases and User Stories. In this case, we developed a tool that automatically extracts triplets from Use Cases, User Stories, or any text written in natural language, and which detects the unnecessary details in the software requirements specification document.

B. Mapping between the Concepts of the Triplet and COSMIC

The triplet structure is a trio of concepts where the subject corresponds to the functional user; the composite predicates correspond to the functional process and the atomic predicates correspond to the data movements. And the objects correspond to the data group manipulated by the functional process. Lastly, the triplet represents a part of the functional process of the software to be measured.

C. Model of Triplet (Triple Store) for Writing Software Requirements

The proposed triplet model is a model that allows to represent the software functional requirements as a triplet, to facilitate the software functional sizing automation. It contains the concepts and knowledge about the COSMIC measurement method, as well as the functional processes of the software to be measured as a triplet. Subsequently, we developed a tool that automatically generates triplets from functional requirements written in the form of a Use Case or a User Story. The structure targeted by the tool is represented as [subject, predicate, object]. The goal is to represent the software requirements as a triplet, consisting of a subject, a predicate and an object. The subject is the functional user of the software; the predicates represent the data movements. As for the object, it corresponds to the data group that is manipulated by the functional process. In addition, the triplet represents a portion of the functional process of the software to be measured. Here is the proposed triplet model for writing software requirements.

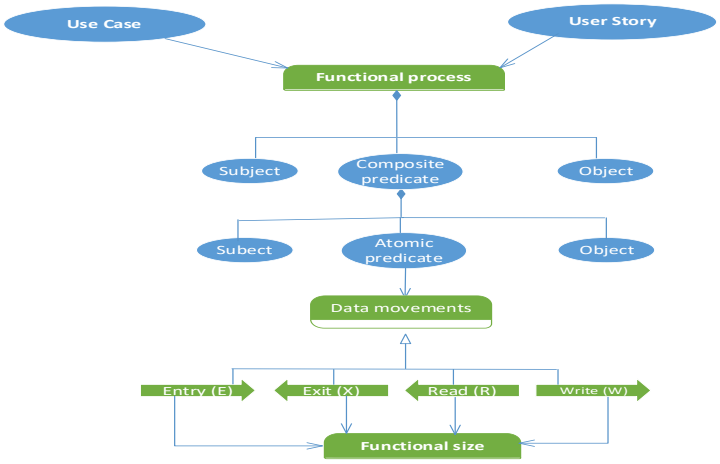

Figure 2. Triplet Model.

Here is a Use Case example written as a triplet:

Description:
Title: Register a new product
The sales manager asks to add a new product. The system displays the product form. The sales manager enters the new product information. The system checks the data. The system records the new product. The system confirms the recording of the new product.

In this example, we described the Use Case using a triplet form (subject, predicate, object). The proposed triplet structure to describe the software functional requirements is simple and effective. It facilitates the process of automating the functional size of the software.

D. Tool for Generating Triplets and Calculating the Functional Size

We have also developed a tool for generating triplets, which contains two modules. The first module is used to automatically generate triplets from Use Cases, User Stories or functional requirements written in natural language. It targets the structure (subject, predicate, object). We assumed that the writing of software requirements is done with dyadic predicates, that is, predicates with two arguments *f (x, y)*. The predicate is expressed by a verb, which is an action to do, and which corresponds to a data movement. The "x" variable is the subject of the action, while the "y" variable is the object of the action. We supposed that in a rule-based system, the rules are based on the idea that writing software requirements

 ISBN: 978-1-61208-894-5

is the construction of dyadic functions. In such a perspective, we associate the function **"f"** with the **"x"** and **"y"** variables. We used a descriptive logic to represent the sentences to be splitted into first order predicate formulas. We were inspired by the natural language analyzer offered by Stanford NLP Software Group to generate triplets from functional requirements written in natural language. This language analyzer is a set of libraries in the artificial intelligence domain, more specifically in the field of automatic natural language processing (NLP), which makes possible to determine the syntactic and semantic structure of sentences. Indeed, this language analyzer contains a class called "TagWord", which semantically identifies the words of a text written in natural language as being made of: subjects, verbs, and objects. Inspired by this software program, we constructed and applied our own rules and algorithms that allow to associate the subject, the predicate, and the object, and to generate the triplets from the Use Cases or User Stories written in natural language.

The second module of the tool is used to obtain the functional sizing of the software to be measured. In fact, the generated triplets are seen as processing rules that allow to infer the functional size. This module quantifies the number of atomic predicates (verbs) of each triplet. Subsequently, a set of rules is applied to make each predicate correspond to a type of data movement (Entry, Exit, Read, or Write). Furthermore, we used the repository framework for automation tools for measuring the software functional sizing, proposed by Abran and Paton [15] to ensure that an automation tool could interact with our technique of software requirements writing. This framework describes a set of desirable characteristics for software functional size measurement automation tools. The main characteristics of the reference framework recommended by Abran and Paton [15] can be summarized as follows:

- Automation tools must be associated with recognized standards.
- The tools must offer, for example, the possibility of interacting with the tools for writing software requirements.
- The tools must provide a presentation of the measurement results to facilitate analyzes.

E. Solving Missing Words with Grammatical Ellipse

The ellipse is a rhetorical figure of speech intended to omit one or more elements in order to make a sentence shorter, while promoting comprehension. In the context of the grammatical ellipse, we tend to omit, for example, a verb or an object. Indeed, in the requirements writing domain, analysts, in order to avoid repetition, omit a predicate (verb) or a noun (object). Let's illustrate with an example a Use Case where there is such an omission of an object: the system checks and saves the data. In this Use Case, the system is checking or verifying the data. Subsequently, it will proceed to their recording (to save the data). We suppose that the writing of software requirements is done with dyadic predicates (f (x, y)). The description in formal logic by could be: ∃ object, ∃ subject, such as predicate (object, subject).

Subject:
{x} = The system
Object:
{y} = data
Predicates:
{f1} =checks
{f2} =saves

We obtained the following logical formula:

$$\exists x\ [f1(x, y1) \wedge f2(x, y2) \quad (1)$$

This Use Case is splitted in two (2) triplets that are respectively:

- The system, checks, the data
- The system, saves, the data

F. Generation of Triplets by Multiple Splittings

In the description of Use Cases, there are sentences containing several objects (complements) and which are connected by logical connectives (AND, OR…), by coordinating conjunctions or by punctuation signs (,). In the automatic text generation literature, there are methods that allow to aggregate structural sentences (subject, predicate, object) using logical connectives. As part of our tool, we were inspired by these methods to proceed by disaggregation. Let's illustrate the following Use Case as an example: "The system verifies the information, saves the data, or returns an error message". We transform each of these actions into a series of predicates of the form: ∃ object, ∃ subject such as predicate (object, subject). We describe use cases in formal logic by variables to represent subjects, predicates, and objects as follows:

Subject:
{x} = The system
Objects:
{y1} = information
{y2} = data
{y3} = error message
Predicates:
{f1} =verifies
{f2} =saves
{f3} =returns

We then obtain the following logical formula:

$$\exists x\ [f1(x, y1) \wedge f2(x, y2) \wedge f3(x, y3)] \quad (2)$$

We give, in the next section, an example of Use Case presenting the manual functional size, as well as the list of triplets generated by the tool and the automatic functional size obtained from the tool.

G. Description of a Use Case and its Manually Measured Functional Size

TABLE I. I illustrates a functional process, for which the data groups are identified, as well as the data movements (EXRW).

The sales manager asks to add a new product. The system verifies the sales manager credentials and displays the new product form or displays a credential error message. The sales manager enters the new product information and asks the system to save the new product. The system verifies the data, records the product, and returns a confirmation message for the addition of the new product or an error message if the product already exists.

TABLE I. EXAMPLE OF MANUALLY MEASURED FUNCTIONAL SIZE

Functional Process Elements	Data Groups	E	X	R	W	Sum of CFP
Asks to add a new product	Credentials	1				1
Verifies the sales manager credentials	Credentials			1		1
Displays a credential error message	Error message		1			1
Displays the new product form	[New product form]					-
Enters the new product information	New Product	1				1
Verifies the data	Data			1		1
Records the product	Product				1	1
Returns a confirmation message	Confirmation message		1			1
Returns an error message	Error message					-
Total:		**2**	**2**	**2**	**1**	**7**

H. Description of a Use Case and its Automatically Measured Functional Size

Figure 3. illustrates the same functional process from the previous example, divided by the tool into several triplets. It determines the functional size and identifies the data movement types (Entry, eXit, Read or Write). It is important to mention that the tool obtains the same functional size result as the manual functional size established by the human expert.

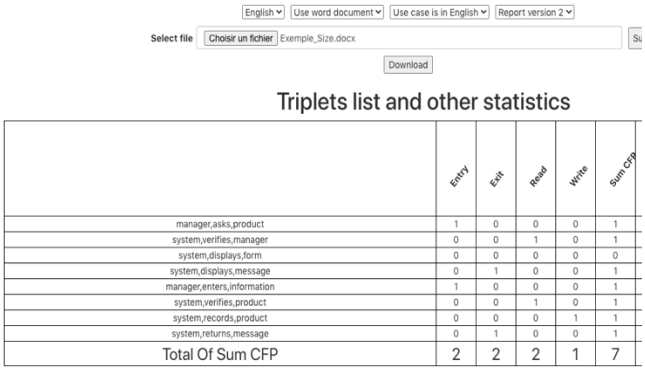

Figure 3. Example of Automaticaly Measured Functional Size.

VI. THE AUTOMATED MEASUREMENT RESULTS OF THE TOOL

We present, in this section, the automated measurement results of our tool developed. The results presented by the tool are compared with those of human experts certified with the COSMIC method.

A. Rules for Identifying Data Movements Types

We adjusted the NLP module by applying a set of rules that allow to construct and extract triplets from Use Case or User Stories written in natural language. Our tool integrates a set of techniques to create a complex artificial intelligence which helps to measure COSMIC function points. Then, we implemented a set of rules to identify the types of data movement (Entry, eXit, Read, Write). These rules are applied once the functional sizing (data movements) has been determined. The size of a functional process is equal to the number of its data movements. Each data movement corresponds to an atomic predicate of each triplet and has a size of 1 COSMIC function point (CFP). *Table II* provides a list of the mapping rules that were implemented for identifying the data movement types of each triplet generated by the tool.

TABLE II. DATA MOVEMENT MAPPING RULES

ID	Definition of Mapping Rules (MR)
MR01	Any data movement from the functional user (human, other software, hardware devices) is considered as an Entry (E).
MR02	A data request to the functionality is treated as an Entry(E)
MR03	Any data movement from a functional process to the functional user is considered as an eXit (X).
MR04	All formatting and data presentation manipulations required to send the data attributes to the functional user is treated as an eXit (X).
MR05	Searching of a data group to persistent storage is considered as a Read (R).
MR06	The logical processing and / or mathematical calculation necessary to read the data are considered as a Read (R).
MR07	Any read request functionality is considered as a Read (R).
MR08	Moving a unique data group to persistent storage is considered as a Write (W).
MR09	The logical processing and / or mathematical calculation necessary to create data to be saved are considered as a Write (W).

B. Software Projects Measured

We presented the functional size results of three (3) measured projects, which requirement documents written in natural language are publicly available on the COSMIC website as case studies. First, COSMIC experts manually measured the functional size of each project according to the measurement manual definitions and rules. Subsequently, we

 ISBN: 978-1-61208-894-5

use our developed tool to automatically determine the functional sizing of these projects from their requirements. We compared the results generated by the tool against those published by experts. Then, we described the observed differences.

C. Automatic Functional Sizing Results from the Tool

We summarized in the *Table III* the automatic functional sizing results of the three (3) projects of software requirements specification documents obtained from the tool. The tool generates the triplets from Use Cases or User Stories described in natural language. Then, it determines the functional size.

D. Evaluation and Validation of Results by COSMIC Experts

Within the framework of this article, we tested the tool with three (3) projects and the results presented were compared with those of human experts certified with the COSMIC method. The manually measured results of these projects are published and available on the COSMIC website. First, our tool generates the triplets from requirements written in natural language, from Use Cases or User Stories written in English or in French. Then, it determines the functional sizing by quantifying the number of atomic predicates (verbs) of each triplet. The research results showed that the generation of triplets from Uses Cases or User Stories can be exploited by measurement automation tools. In fact, our proposed tool presents automated results that are consistent with the manual results validated and published by experts, with an average accuracy of 98.30%, as shown in *TABLE* III, where the accuracy varies between 96.97% and 100%.

TABLE III. AUTOMATIC AND MANUAL FUNCTIONAL SIZE COMPARISONS

Project	Manual Functional Sizing	Automatic Functional Sizing	Accuracy
Resto Sys	119	117	98.32%
ACME Car Hire System	33	32	96.97%
Rice Cooker	24	24	100.00%
Total	**176**	**173**	**98.30%**

E. Threats to Validity

Requirements are generally written with active verbs and not with passive verbs. Nevertheless, it is likely to meet cases where some Use Case scenarios are described in the passive form, i.e., subject and object are swapped. One limitation is that the tool could not generate triplets for sentences written in the passive form. However, the tool detects sentences written with a passive voice and raises the issue as a potential error. Also, the proposed tool is not able to detect format elements of the requirements document, such as headers, footers, titles, etc. Requirements text has to be uploaded from a Word or PDF file into our tool, where this file should contain only requirements text without any format element. Because of manual manipulations to create that file, there is a possibility of human errors, such as some requirement text not copied or copied twice.

VII. CONCLUSION AND FUTURE WORK

We proposed in this article a new method for writing software requirements that could help to automate the functional size measurement process. This technique is a triplet approach. It is proven, tested, and validated by the development of a new tool for automating the measurement of the functional size of software, as defined by the COSMIC method. This tool allows generating triplets from Use Cases or User Stories written in natural language, more specifically in English or in French (Use Cases or User Stories written in English or in French). In addition, it determines the functional sizing of software, in adding the sum of predicates and identifying the types of data movements. The tool approximates the human experts at about 98.30%.

In the future, our perspective will try to integrate a new module which would ensure that the tool could generate triplets for sentences written in the passive form and that would detect the format of the requirements documents. Furthermore, we will work on a machine learning module which would allow that the tool could improve gradually during its implementation. The goal will be to allow the tool to learn to solve problems by itself, without necessarily needing the intervention of human experts.

REFERENCES

[1] V. Bévo, “Analyse et formalisation ontologique des procédures de mesure associées aux méthodes de mesure de la taille fonctionnelle des logiciels: De nouvelles perspectives pour la mesure” (Ontological analysis and formalization of measurement procedures associated with software functional size measurement methods: New perspectives for measurement), Ph.D. thesis, Montreal, Université du Québec à Montréal, 314p, 2005. [Online]. Available from: https://dic.uqam.ca/upload/files/theses/bevo_these.pdf, [Retrieved: April, 2021].

[2] S. Azzouz and A. Abran, “A proposed measurement role in the Rational Unified Process: Automated Measurement of COSMIC-FFP for Rational Rose Real Time”, In Information and Software Technology, vol. 47, no. 3, pp. 151-166, 2004.

[3] K. Wiegers, “More About Software Requirements. New York: O'Reilly Media”, Inc, 2009.

[4] S. Trudel, “The COSMIC ISO 19761 functional size measurement method as a software requirements improvement mechanism”, Ph.D. thesis, École de Technologie Supérieure (ETS), 2012.

[5] B. Boehm, “Software cost estimation with COCOMO II”, Upper Saddle River, N.J; London: Prentice Hall International, 2000.

[6] A. Albrecht, and A.J. Gaffney, “Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation”, IEEE Transactions on Software Engineering, vol. 9, no. 6, pp. 639-648, 2000.

[7] A. Abran, “Software Metrics and Software Metrology. Hoboken”, N.J.: Wiley Los Alamitos, Calif.: IEEE Computer Society, 328 p, 2010.

[8] S. Black and D. Wigg, “X-Ray: A Multi-Language, Industrial Strength Tool”, IWSM’99, Lac Supérieur, Canada, vol. 8, no. 10, pp. 39-50, 1999.

 ISBN: 978-1-61208-894-5

[9] I. Jacobson, "The unified Software Development Process", The Journal of Object Technology, vol. 2, no. 4, pp.1-22, 2003.

[10] K. Beck and D. West, "User Stories in Agile Development", In Scenarios, Stories, Use Cases: Through the Systems Developments Life-Cycle, 2004.

[11] A. Cockburn, "Writing effective use cases", Addison-Wesley Longman, 2001.

[12] "IEEE Standard Glossary of Software Engineering Terminology", in IEEE Std 610.12-1990, vol., no., pp.1-84, 31 Dec. 1990, doi: 10.1109/IEEESTD.1990.101064.

[13] S.W. Ambler, "Examining the Big Requirements Up Front (BRUF) Approach", Ambysoft inc., [Online]. Available from: http://agilemodeling.com/essays/examiningBRUF.htm, [Retrieved: April, 2021].

[14] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap", In Proceedings of the Conference on The Future of Software Engineering, New York (USA), pp. 35-46, 2000.

[15] A. Abran and K. Paton, "Automation of Function Points Counting: Feasibility and Accuracy", Université du Québec à Montréal, pp. 1-21, 1997.

[16] A. Abran *et al.*, "The COSMIC Functional Size Measurement Method Version 5.0", [Online]. Available from: www.cosmic-sizing.org, [Retrieved: April, 2021].

[17] K. Manoj, "What is TDD, BDD & ATDD ?", Assert Selenium, Nov. 5th, 2012, [Online]. Available from: http://www.assertselenium.com/atdd/difference-between-tdd-bdd-atdd/, [Retrieved: April, 2021].

[18] M. Downing, M. Eagles, P. Hope, and Ph. James, "ACME Car Hire Case Study v1.0.1", August 2018. [Online]. Available from: https://cosmic-sizing.org/publications/acme-car-hire-case-study-v1-0-1/, [Retrieved: April, 2021].

[19] A. Sellami, M. Haoues, and H. Ben-Abdallah, "Sizing Natural Language/User Stories/UML Use Cases for Web and Mobile Applications using COSMIC FSM", May 2019. [Online]. Available from: https://cosmic-sizing.org/publications/restosys-case-v1-2/, [Retrieved: April, 2021].

 ISBN: 978-1-61208-894-5