

COSMIC Solver: A Tool for Functional Sizing of Java Business Applications

A. Tarhan, and M.A. Sağ

Abstract—Functional Size Measurement (FSM) provides a ground during software project life-cycle to estimate planning parameters and track progress. Since it is time-consuming, costly, and error-prone when functional size is measured manually, automating the process of measurement has come to the fore. The literature includes studies that automate FSM from software artifacts such as requirements specifications, design models, and software code. In this study we focus on automation of FSM from software code, and share our experience towards developing a tool called ‘COSMIC Solver’ for COSMIC FSM of Java Business Applications (JBAs). The tool automates the following steps: (i) Eliciting textual representations of UML sequence diagrams from functional execution traces of a JBA, (ii) tagging of these textual representations with the help of AspectJ technology to measure COSMIC functional size, and (iii) calculating functional size of user scenarios run in the JBA from the information on the tags according to COSMIC FSM rules. In this paper we explain features and measurement method of COSMIC Solver (v1.0), and the share results obtained from functional sizing of an open source JBA by using the tool.

Index Terms—Functional size, function points, automation, tool, COSMIC FSM, software code, UML, sequence diagram, AspectJ.


I. INTRODUCTION

SINCE ALBRECHT introduced the concept of ‘functional size’ as a new dimension to size software products [1], it has been favored important for estimating and tracking cost of software projects [2, 3]. Widely known for its function point (FP) metric, functional size can be measured earlier in project life-cycle and as independent of design technology, programming language, or developer skills. Because of these properties, it provides a solid ground to estimate planning parameters and track progress during software life-cycle. The term Functional Size Measurement (FSM) has emerged and been defined as “the process of measuring software’s functional size derived by quantifying the functional user requirements that describe what the software shall do in terms of tasks and services” [4]. Several FSM methods that have been adopted as ISO standards in the last two decades, which include Mk II FPA

[5], NESMA FSM [6], FISMA FSM [7], IFPUG FSM [8], and COSMIC FSM [9]. FSM can be useful to track software development progress by using earned value based on functional size [10–13]. Earned value enables software practitioners to track the actual value achieved (earned) during the project in comparison to the planned value [14]. Function points, in this regard, is a valuable metric as the base to calculate the planned and earned values as independent of programming language or developer. The benefits of FSM is more prominent in iterative development [15–17] or agile development [18–20], where practitioners can identify the amount of functionality delivered to customer by software releases and create or revise work plans accordingly. FSM of completed products (e.g., software code) can also be useful for software organizations in building their historical size databases, based on which project estimations can be developed. Due to mentioned benefits, FSM is useful for wider and cost-effective adoption of size-based project management practices by software community.

Since it is significant that functional size be measured correctly, timely, and practically, automating the process of FSM has come to the fore as a solution. FSM automation is claimed to minimize human errors and decrease costs [21–23]. A recent survey on FSM based on code [24] reveals that FSM is considered an important tool for decision making in software projects (with agreement of 87% of the respondents). The same survey indicates that FSM automation is perceived as important but difficult to realize, and that automating FSM directly from source code can help measurement specialists and decision makers. These findings show there is awareness in community that FSM is a valuable practice but the means (e.g., methods and tools) to apply it in a useful way, typically as part of daily project routine, are not yet mature enough. Emerging research on automated FSM might be helpful to create the base for effective and efficient (i.e., target-oriented and resource-economic) adoption of FSM, and narrow the gap between software project management theory and practice.

The literature includes studies that automate FSM from early software artifacts such as analysis and design models [25–29] as well as the ones that automate FSM from software code [23, 30–33]. In this study, we focus on automation of FSM from software code. In a previous work of ours [32] covered by a thorough review of studies on this topic [34], we introduced an automation prototype for COSMIC FSM [9] and evaluated its accuracy on a toy application that we developed. In this paper

✉ A. TARHAN, is with Department of Computer Engineering, Hacettepe University, Ankara, Turkey, (e-mail: atarhan@hacettepe.edu.tr) 

M.A. SAĞ, is with Turkish Academic Network and Information Center (ULAKBIM) of the Scientific and Technological Research Council of Turkey (TUBITAK), Ankara, Turkey, (e-mail: muhammetalisag@gmail.com)

Manuscript received August 9, 2017; accepted Dec 21, 2017.

DOI: [10.17694/bajece.401986](https://doi.org/10.17694/bajece.401986)

we explain features and design of an improved version of the prototype which has evolved into a tool called COSMIC Solver (v1.0). We also elaborate on the results from functional sizing of an open source Java Business Application (JBA) using the tool, and from the evaluation of measurement accuracy of the tool with respect to a verification protocol aimed for automated FSM tools [35].

The remainder of this study is organized as follows. Section 2 provides background on COSMIC FSM and AspectJ. Section 3 summarizes related studies by claiming similarities and contribution of our proposal in comparison to theirs. Section 4 explains attributes and user requirements of COSMIC Solver as well as its measurement method and components. Section 5 overviews results from manual and automated functional sizing of the open source JBA, evaluates the results, and highlights opportunities for improving the tool. Finally, Section 6 closes the study by providing general conclusions and plans for future work.

II. BACKGROUND

A. COSMIC FSM method

ISO/IEC 19761 (COSMIC FSM) [9] defines principles and rules to decompose a system into layers, and to size layered software based on a generic software model identifying *functional processes*, *data groups*, and *data movements* in response to *functional user requirements (FURs)*.

Functional processes can be defined as independent data movements of a FUR. A functional process is a series of data movements of four types:

- *Entry (E)* moves a data group from a functional user across the boundary into a functional process;
- *Exit (X)* moves a data group from a functional process across the boundary of the application to a functional user;
- *Read (R)* moves a data group from persistent storage through a functional process; and
- *Write (W)* moves a data group lying inside a functional process to persistent storage.

The COSMIC Function Points (CFP) measurement method states that the number of data movements is proportional to functional size of the measured software [36]. FURs are related with data transfer, transformation, persistence, and data retrieval. Functional process is the base component of a FUR and is triggered by events (i.e., an Entry data movement) that take place in user's world. It then executes in response to the triggering event. A functional process includes an Entry data movement and either a Write or an Exit data movement – i.e., it includes at least two data movement types. Each data movement in a functional process is counted as one CFP.

COSMIC FSM method [36] defines three measurement phases as we explain in the following paragraphs.

(1) In *Measurement Strategy Phase*, the purpose and the scope of measurement, functional users of the application to measure, and the level of granularity of measurement are specified. The purpose clarifies why the measurement is done and for which reason its results will be used. Specification of

measurement purpose is also important for identifying the scope and the functional users. Scope means the requirements of the functional users who will be incorporated into the measurement. Functional users are the users who receive or send data for a software unit in the defined FURs. Finally, the level of granularity indicates the detail level at which the measurement will be done. Measurements should be made at the same granularity level to compare with each other.

(2) In *Mapping Phase*, event masses (i.e., functional processes) that are triggered by the user are identified. Then, data groups and data qualifications related with these processes are determined. A data group is the data that provides same *object of interest* with a holistic view and that clusters for an ad hoc query. If the query interacts with multiple object of interests, number of all interacted object of interests will be taken into account in calculating CFP.

(3) In *Measurement Phase*, data movements are detected for all functional processes, the measurement function is implemented, and the results are consolidated. Sub-processes consist of data movements and data manipulations. Data manipulations are not taken into account within the scope of COSMIC FSM method. The total size is calculated by consolidating the data movements. While calculating the consolidated size; if the same level of granularity is used for measurement, the values obtained on the basis of functional processes are summed mathematically. Otherwise, they are added after they are scaled to the same level.

B. Aspect-oriented programming and AspectJ

Typical enterprise and web applications today have to address 'concerns' as security, transactional behavior, and logging. The subsystems that provide these services can be implemented in a modular way. However, to use these services, the duplicate code fragments must be inserted into various places in the application to invoke them. For example, to produce logs for certain operations in an application, log writing procedures must be inserted into the beginning of every method as needed. Aside from the redundancy issue, it would be difficult and error-prone to modify or replace this logging approach later, should that become necessary. In addition, the application code is mixed up with the code for logging concern, which both compromises clarity and makes it hard to reuse the code in another context.

To solve mentioned problems, Aspect-Oriented Programming (AOP) was proposed [37]. AOP addresses the 'cross-cutting concerns' that compromise the modularity of object-oriented systems by modularizing the concerns as aspects and by providing mechanisms to combine aspect and object modules to compose applications [37, 38]. Concerns (or *crosscutting concerns*) like logging or authentication, typically cut across a number of application module boundaries (e.g., classes). That is, the points of intersection are defined once, in one place, making them more understandable, rectifiable, and usable. The other modules require no modifications to be advised by the aspects.

It is important to explain four terms in AOP [38] as the key to our solution in this study:

- *Join point* is a well-defined point in the execution of an application.
- *Pointcut* is a way of specifying a join point by means of configuration or code. Pointcuts allow a programmer to specify join points (in the execution of a program like method call, object instantiation, or variable access) in program flow.
- *Advice* is a way of expressing a cross cutting action that needs to occur. Advices allow a programmer to specify code to run at a join point matched by a pointcut.
- *Weaving* consists of executing the aspect advice to produce only a set of generated classes that have the aspect implementation code woven into it [38].

The implementation of AOP in Java environment is enabled by AspectJ as a seamless extension to Java. AspectJ is designed to encapsulate the previously mentioned crosscutting concerns and thereby restore overall system modularity. These concerns or aspects are identified, modularized, independently developed, and then combined with the main object model in a structured way to compose the application through weaving [39]. AspectJ performs weaving during program compilation or load time. In our proposal, we use load-time weaving (LTW) which presents an opportunity to work with byte code. As a result, even though the source code is unreachable, we could perform weaving operations with LTW. AspectJ LTW technology is using ‘java agent’, which holds the key to monitoring and profiling ‘Java Virtual Machine’, and is able to modify Java classes dynamically as they are being loaded.

While weaving target application at runtime, ‘pointcuts’ catch specified signature and ‘advices’ decide what to do after the catch. AspectJ divides an advice into three sections according to its runtime: The one that runs *before* its join point, the one runs *after* its join point, and the one that runs in the exact place of (or *around*) its join point. While generating sequence diagrams, the use of ‘before’ and ‘after’ advices are sufficient in our measurement method. In other words, the actions in advices are being performed before and after the specified ‘join point’. With qualified pointcuts, javaagent, and AspectJ technology, we can catch and tag the candidate functional processes at runtime.

III. RELATED WORK

In this section we summarize related studies that inspired our proposal in certain features, and highlight similarities and contribution of our proposal in comparison to theirs. Interested readers may refer to [34] for a wider set of studies on FSM automation from software code.

Kusumoto et al. [40] refer to traditional functional size calculations and evaluate the possibility of automatic calculation of function points from source code by applying statistical analysis. By focusing on screen transitions, the authors propose a method that counts data and transaction functions of web applications. The study demonstrates the feasibility of automated measurement of function points from source code within IFPUG measurement program, and shows that UML models can be successfully used for FSM. Bévo et al. [25] map the concepts in UML class diagrams and use case diagrams onto the abstractions of COSMIC FSM model to

measure functional size based on high-level specifications of software. Jenner [41] discusses the granularity aspect of use cases in the proposal of Bévo et al [25]. Each functional process is represented by a sequence diagram as an adequate abstraction level of a use case. Both of these proposals focus on finding an appropriate mapping between UML model elements and FSM concepts. Levesque et al. [26] also apply COSMIC FSM method to measure functional size from use case diagrams and sequence diagrams. The authors group functional processes in two categories as data movement and manipulation types, and focus on the deduction of COSMIC functional size by using actor-object flow diagrams while considering UML message exchanges.

In our proposal, we have been inspired by the studies of Jenner [41] and Lévesque et al. [26], which show the feasibility of mapping between sequence diagrams and COSMIC FSM concepts. According to Lévesque et al. [26], data movements can be mapped to messages in a UML sequence diagram and consequently, it becomes easy to calculate the functional size by aggregating all messages exchanged in the sequence diagram.

Our proposal is also in parallel with Kusumoto et al.’s proposal [40] that is considered successful yet having lack of elasticity in being generic. We offer a more flexible and adaptable method in getting the sequence diagrams with the help of aspect-orientation and AspectJ technology.

Bévo et al. [25] and Jenner [41], on the other hand, provide different interpretations of a functional process because the concept of a ‘triggering event’ is not explicitly presented in UML. Triggering event problem is widely known in UML assisted FSM methods, and our proposal handles this problem by finding and tagging the methods of user interface components in the mapping phase of automated measurement. In addition, source code or binary code of JBAs can be taken as input to our proposal for calculating CFP.

IV. COSMIC SOLVER

COSMIC Solver is aimed to automate FSM from software code of JBAs in accordance to COSMIC FSM method [9]. In the following sub-sections, we provide an overview of COSMIC Solver with respect to attributes and user requirements that we see critical for automation, and explain the details of automated measurement.

A. Attributes and User Requirements

In a recent study involving the first author [34], a set of attributes have been proposed for automating COSMIC FSM from software code based on previous experiences. The proposed attributes are trigger handling, functional process identification, external interface definition, data persistence interface definition, data group identification, domain class identification, scope identification, and technology identification. These attributes are described in Table I together with any support by the current implementation (v1.0) of COSMIC Solver. Most of the attributes are supported by the tool, and the only missing attribute is domain class identification. In addition, functional process identification and data group identification are embedded in automation software and cannot be tailored by measurer.

TABLE I.
ATTRIBUTES OF COSMIC SOLVER FOR FSM AUTOMATION FROM SOFTWARE CODE

Attribute	Description	Support by COSMIC Solver (v1.0)
Trigger handling	The specific identification of triggering events that start functional processes.	√
Functional process identification	The identification of a functional process which is a unique, cohesive and independently executable set of data movement types.	√
External interface (E/X) definition	The identification of external interfaces where information is exchanged with functional users, and which will be used to catch Entry and Exit data movements.	√
Data persistence (R/W) interface definition	The identification of data persistence interfaces, which will be used to catch Read and Write data movements.	√
Data group identification	The identification of a data group that is a set of attributes that describe particular aspect of an object of interest.	√
Domain class identification	The identification of objects that are of interest to its functional users (i.e., domain classes) and the exclusion of classes that are implemented for only design purposes.	
Scope identification	The identification of measurement boundary and the selection of functional user requirements to measure.	√
Technology identification	The identification of specific architecture and technology of target applications.	√

User requirements of COSMIC Solver to measure the functional size of a JBA are shown in Fig. 1 by a UML use case diagram that includes three main use cases: (1) Export Pointcuts, (2) Compile and Run Pointcuts, and (3) Calculate CFP. The last use case further includes four use cases, which are Parse Sequence, Process Sequence, Find Duplicate Sequences, and Find Conflicted Sequences, executed internally by the tool. The ‘sequence’ here indicates a number of data movements that make up a functional process. The first two use cases, “Export Pointcuts” and “Compile and Run Pointcuts”, serve for preparing the pointcuts for CFP calculation, and should be carried out consecutively prior to execution of the third use case “Calculate CFP”. In addition, while the third use case is typically executed by a Standard User who wants to size a JBA, the first two use cases should be carried out by a Measurement Specialist who has knowledge about Java, AspectJ, and COSMIC FSM method.

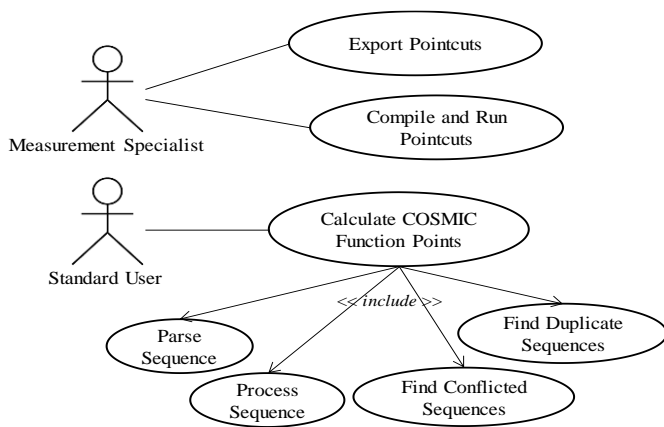


Fig. 1. User Requirements of COSMIC Solver by a UML Use Case Diagram

B. Measurement Method

We use black-box approach to automate FSM from source or binary code of a two- or three-tier JBA without a need to modify its code. A two- or three-tier architecture means a client-server architecture in which the presentation, the application

processing and the data management are logically separate processes. We identify and presume a constant mapping between technology elements of the JBA and the COSMIC FSM data movements, but we enable some degree of tailoring for the measurer based on selected technologies.

The tool first elicits textual representations of user scenarios from functional execution traces of a JBA, which correspond to sequence diagrams in Unified Modeling Language (UML). It then tags these textual representations of execution traces by considering four data movement types (Entry, Exit, Read, and Write) in COSMIC FSM method with the help of AOP concepts, more specifically by using AspectJ technology. It finally calculates functional size of the user scenarios executed at runtime from their tagged representations.

The phases and components of the automated measurement process are shown in Fig. 2. As it is seen from the figure, COSMIC Solver has two components, which are *Tracer* and *Calculator*. ‘Tracer’ component retrieves sequence diagrams during dynamic execution of a JBA with the help of AspectJ. ‘Calculator’ component, on the other hand, applies COSMIC FSM rules to calculate the function points.

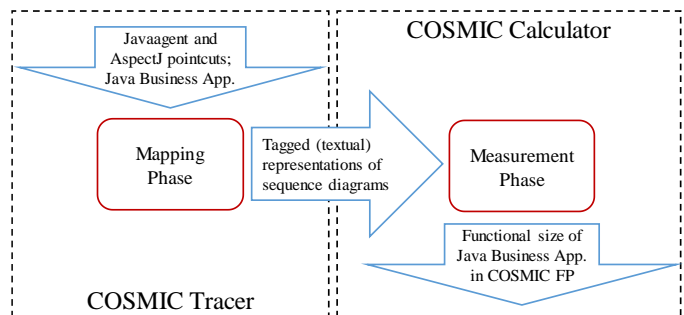


Fig. 2. Phases of Automated Measurement Process

We use AspectJ technology and related concepts of pointcuts and advices in order to set and observe the stated mappings, and allow the measurer to revise the pointcuts which are indeed the basic constructs for observing data movements in candidate

functional processes. The pointcuts are created statically in the measuring software which then monitors the JBA and calculates its functional size at runtime. AspectJ catches execution traces of user scenarios and transform these traces into behavioral specifications by tagging. Tagging is performed to prepare the execution traces of the scenarios for measuring CFP according to COSMIC FSM rules. Once the traces are transformed into textual specifications, CFP is automatically calculated at runtime for the executed user scenarios.

1) *COSMIC Tracer*

Tracer component provides a useful framework to capture interactions (messages) between classes while executing functional processes. To do that, it uses ‘before’ and ‘after’ advices and pre-defined ‘pointcuts’. Almost half of the measurement operation is implemented via this component. Tracer component captures all join point executions and calls, in order to construct the structured text versions of sequence diagrams. These executions and calls use some specific method signatures to get neat and optimal sequence diagrams which do not contain every system call and execution but include only required ones that belong to developer space.

In order to get textual representations of sequence diagrams with tags (e.g., SWING, DIALOG, JDBC) for specifying entry points, dialog box operations, database calls, and etc., we use specific method signatures and object references. For example, to catch JDBC operations we use two pointcut patterns as:

```
execution(* java.sql.Statement.exec*(..))
call(* java.sql.Statement.exec*(..))
```

Apparently, a JBA can contain function types other than the ones specified. To handle them, appropriate pointcuts should be defined in Tracer component, which is possible by extending its current implementation. Then, with the help of advices, specified tags could be appended to support the identification of data movement types; and a piece of code, which constitutes an advice, runs at defined join point that is picked out by the related pointcut.

Kind of an advice determines the behavior of the running code, and how it interacts with the join point. Using these advices before and after every call/execution, we can intercept processes. Then, we call a specific function to write the method signature as shown in Table II (a). These advices produce output archetypes given in Table II (b).

TABLE II.
‘BEFORE’ & ‘AFTER’ ASPECT ADVICES AND TRACE LOG

(a) Advices	<pre>before() : jdbcCall() jdbcExecution() { [Start: {tag}>] <print signature> } after() : jdbcCall() jdbcExecution() { [End: {tag}>] <print signature> }</pre>
(b) Trace Log	<pre><Start:tag>> returnType PackageName. ClassName. MethodName([Parameters]) [(if exist)SQL Statement] <End:tag>> returnType PackageName. ClassName. MethodName([Parameters])</pre>

As a result, for every user interaction on target JBA, a textual and tagged representation of a sequence diagram is generated with the help of pointcuts and advices. The generated sequence

diagram has tree style; in which every node has ‘Start’ and ‘End’ tags and might contain another method call or execution. This information is processed with Calculator component to identify functional size in CFP.

2) *COSMIC Calculator*

Calculator component calculates functional size of the JBA by applying COSMIC FSM rules to tagged (textual) representations of UML sequence diagrams. The basic steps to calculate CFP are (1) Analyze Candidate FURs, (2) Identify Application Boundary, (3) Evaluate Functional Processes, and (4) Calculate CFP. We explain the key points for each in the following paragraphs.

(1) While generating sequence diagrams, there may be multiple interactions with the same functional processes. This brings out duplicate sequences that should be ignored. Occasionally, some functional processes might have the same trigger but different flows. This causes a more complicated problem than duplication. If there is a situation like that the functional size is measured by an approximation approach.

(2) Application boundary indicates the border between the JBA being measured and external applications or user domain. It determines which functions will be included or not in counting the function points. To deal with ‘triggering event’ problem, we specify user interface components of the target application and search for specific method signatures. For example, if an application uses Swing or AWT libraries for user interface, we define events via these interface components as the starting points of functional processes.

(3) The COSMIC FSM method states that a FUR is considered as a functional process if it is independently executable and triggered by an event in the world of functional users. Using the tags and COSMIC FSM rules, we identify if a user operation initiates and leads to a functional process.

(4) For each functional process identified, data movement types are also detected from the tags. CFP is then calculated by summing the number of data movements for all functional processes in all FURs.

V. FSM OF AN OPEN SOURCE JBA

Measurement validity of COSMIC Solver was initially tested for its prototype on a toy (personnel and payment database) application, and it was found that manual and automated measurement results were very convergent with an error rate of 1/32 [32]. After the prototype has been evolved into a tool with some generic features such as creating and compiling pointcuts, we aimed to evaluate the performance of COSMIC Solver on an open source JBA that meets measurement constraints of the tool. Consequently, we performed a case study to answer the following research questions (RQs):

- RQ-1: How effective is COSMIC Solver in measuring CFP of an open source JBA?
- RQ-2: How efficient is COSMIC Solver in measuring CFP of the JBA in comparison to manual measurement?

The steps that we followed to answer these RQs are explained in the sub-sections below.

A. Application Selection and Information Gathering

We searched for JBAs in the open source software repositories such as GitHub, SourceForge and Google Code, and identified candidate applications that meet the measurement constraints of COSMIC Solver.

More specifically, we checked the following criteria: The application has been developed in Java programming language and by using Java Platform, Standard Edition (Java SE) 1.5 or later; the application has a two- or three-tier software architecture; the user interface layer of the application has been developed using Java Swing; the application is rich in Create, Read, Update, Delete, and List (CRUDL) operations; the libraries used by the application are compatible with the standards of Java API for XML-based RPC (JAX-RPC), JPA, and JDBC; and the application conforms to the principles of the Software Context Model and Software Generic Model defined in Appendix D of COSMIC FSM method [36].

TABLE III.
PROPERTIES OF NYAGUA AQUARIUM MANAGEMENT JBA

Operation Platform	Runs on every O.S. that can run Java applications
Licence	It has GNU GPL v.2 licence; it can be freely used.
Java Version	Java SE 1.7
GUI	Java Swing
Database	SQLITE
JDBC/Persistence	Native JDBC
JDBC URL	jdbc:sqlite://localhost/<file>
Dev. Platform	NetBeans Standard IDE
Libraries Used	sqlite-jdbc-3.7; Jcalendar – 1.4; JDK 1.7

Among the candidate applications identified according to these criteria, we selected ‘Nyagua Aquarium Management Application’ in the SourceForge repository. General and technical properties of the application, which best satisfied expected criteria, are shown in Table III. The application has its source code and no other documents in the repository. Therefore, manual identification of functional processes and CFP measurement were done by navigating use cases of the application at runtime, and reviewing its source code.

B. Manual and Automated Measurement Results

To frame and equate the scopes of manual and automated measurements, we navigated user interfaces of the Nyagua Aquarium JBA prior to measurement, and identified use case groups (FURs) and objects of interest candidate for COSMIC FSM. As a result, we selected 12 FURs for functional sizing to verify operational performance of COSMIC Solver.

An expert who succeeded COSMIC foundation examination manually measured CFP of Nyagua Aquarium JBA in the framed scope, and recorded manual measurement effort. Then, we automatically measured CFP of the JBA for the same scope by using COSMIC Solver, and recorded the duration of automated measurement. Table IV summarizes manual and automated measurement results. Manually measured size was 356 CFP regarding 76 functional processes identified in 12 FURs targeted for FSM, and duration of manual measurement was 6 hours. For automated measurement, we ran 12 FURs in the framed scope, and calculated COSMIC functional size as 273 CFP regarding 49 functional processes identified

automatically by COSMIC Solver in almost 30 minutes. The duration of automated measurement included the time for preparing the application for automated measurement (e.g., creating and compiling the pointcuts, and transforming the application to include AspectJ features).

TABLE IV.
RESULTS OF MANUAL AND AUTOMATED MEASUREMENTS

	Manually	Automated
FURs measured	12 FURs	12 FURs
Functional Processes identified	76 Func.Proc.	49 Func.Proc.
CFP measured	356 CFP	273 CFP
Duration of measurement	6 hours	½ hours

C. Evaluation of Results

We compared the results of manual and automated measurements by following a verification protocol proposed by Soubra et al. [35] for tools that automate COSMIC FSM method [9]. The aim of the protocol is to ensure that the whole measurement chain produces the right measurement results. According to the protocol, measurement accuracy of a tool is verified in three phases. *Phase-1* requires a comparison of measurement results (i.e., CFP) calculated by tool and obtained manually. *Phase-2* requires a detailed comparison of these results (e.g., at levels of functional processes and data movement types) to understand reasons of deviations, if any. *Phase-3* requires verification of automation tool and inputs to determine which module of the tool is responsible for error.

In Phase 1 of the protocol, we compared the results of manual and automated measurements. We found that in the former we measured 356 CFP while in the latter COSMIC Solver calculated 273 CFP, which resulted in an automation accuracy rate of 77%. This is less than 94% in [23] and 92% in [31], both requiring measurement code insertion into JBA, and 81% in [33] that does not handle triggering events. Since our tool measures CFP from the JBA as is without a need for code insertion and handles the triggering events, we consider its rate successful as an initial result on a third party JBA.

In Phase 2 of the protocol, we compared the number of functional processes and data movements and found that neither of them were identical. The number of functional processes identified was 76 in manual measurement and 49 in automated measurement. Table V shows the FURs and objects of interest in the sample frame, together with numbers of functional processes and CFP values that were measured manually and by using the tool. As observed from the table, the manual and automated numbers of functional processes or CFP values showed deviation for each FUR.

In Phase-3 of the protocol, we investigated the reasons of these deviations. From the logs of manual and automated measurements it was clear that Tracer component was only partially effective in tagging the manually identified functional processes. This was mainly because: (i) we could not define all pointcuts to handle the variety in the triggering events (e.g., for retrieving hidden objects of interest), and (ii) Tracer component could not identify different functional processes started by the same triggering event. These findings indicated a need for reviewing and improving the mapping phase of the automation, especially for the pre-defined pointcuts.

TABLE V.
USE CASE GROUPS, OBJECTS OF INTEREST, AND FUNCTIONAL PROCESSES FOR COSMIC FSM

Use case group (FUR) No	Object of Interest	# Functional Processes		Calculated CFP		Manually Calculated CFP for Automatically Identified Functional Processes
		Manual	Automated	Manual	Automated	
1	Aquarium	4	4	18	18	15
2	Maintenance	6	4	29	21	21
3	FishBase	7	4	30	24	17
4	InvertBase	7	4	30	27	21
5	PlantBase	7	4	30	24	21
6	Fish	7	4	34	16	23
7	Plant	7	4	34	16	23
8	Inverts	7	4	34	16	23
9	Expense	6	4	49	30	24
10	Reading	5	4	22	16	19
11	Device	7	4	25	16	16
12	Schedule	6	5	21	37	15
TOTAL		76	49	356	273	238

To understand the reasons of deviations in the level of data movement types, we also manually calculated CFP values in the scope of automatically identified functional processes, as shown in the rightmost column of Table V. The totals of manually and automatically measured CFP values in this narrowed scope were closer (238 and 273 with an automation accuracy rate of %85) but still not equal. This finding implied that there were also deviations in the types or numbers of the data movements identified. When we checked the logs of manual and automated measurements we noticed that for the same functional processes, a greater number of Read and Exit data movements were identified in some cases (e.g., for retrieving objects of interest that were part of the main ones, and returning error messages or reports) in manual measurement. In some other cases, however, Calculator component counted a greater number of Entry and Exit data movements, which was led by redundant tagging via Tracer component.

Based on the findings and related investigations mentioned above, we identified the following opportunities for improving COSMIC Solver at the end of the case study implementation:

- Pointcuts pre-defined in COSMIC Solver (SWING, JDBC, JPA, DIALOG, JAX-RPC) should be revised and extended to catch additional function types (e.g., standard file read/write operations, and seldom used Swing components);
- Tracer component should be revised to handle: (i) different functional processes initiated by the same triggering event, (ii) hidden objects of interest that are part of the main ones (e.g., aquarium image); and (iii) redundant tagging of Entry/Exit data movements.

D. Potential Threats to Validity

There is a number of validity threads to discuss for the case study implementation, as suggested by Yin [42]. *Construct validity* is related to identifying correct operational measures and avoiding subjective judgements. We followed COSMIC FSM method [36] in measurements and used a three-stage verification protocol [35] to cope with subjectivity. *Internal validity* requires seeking to establish a causal relationship as distinguished from a spurious one. The fact that the second author who developed COSMIC Solver participated in the case study can be considered an internal thread, and its effect on evaluation was mitigated by supervision of the case study by the first author and also by manual measurement of CFP by a

COSMIC FSM expert. *External validity* is about defining the context to which a study's findings can be generalized. We cannot yet argue that COSMIC Solver would be effective and efficient in FSM of other JBAs, and we plan further studies to deal with this threat. As the last test, *reliability* is related to repeatability of the operations in a case study, which is possible by following the steps in sub-sections 5.A thru 5.C.

VI. CONCLUSION

The progressing state of FSM in software industry reveal the utmost importance of its automation. Consequently, in this article we focused on automated measurement of CFP from software code of JBAs. We 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.

For the case study, we identified two research questions. RQ-1 was related to the effectiveness of COSMIC Solver in measuring CFP of the JBA. The evaluation results showed that CFPs measured manually and automatically were convergent by 77%. This indicated that COSMIC Solver was effective in functional sizing of the JBA when compared to the features and success rates of the previous proposals (e.g., [23], [31] and [33]). The RQ-2 was related to the efficiency of the tool in measuring CFP in comparison to manual measurement. The durations recorded during manual and automated measurements were 6 hours and 30 minutes, respectively. Therefore, COSMIC Solver was 12 times more efficient than manual measurement in functional sizing of the JBA.

In future work, we plan to revise the current version (1.0) of the tool by considering improvement opportunities identified by the case study, and the requirements of Automated Function Points specification [43]. We also plan to perform further case studies, especially to cope with the validity threads regarding the generalizability of the automated measurement results and the independence of the measurer.

ACKNOWLEDGMENT

The authors thank Onat Ege Adalı for manual functional sizing of Nyagua Aquarium Management Application.

REFERENCES

1. Albrecht AJ (1979) Measuring application development productivity. In: IBO Conf. Appl. Dev. pp 83–92
2. Jones C (2004) Project Management Practices : Success versus Failure. Crosstalk 5–9.
3. Jorgensen M, Shepperd M (2007) A Systematic Review of Software Development Cost Estimation Studies. IEEE Trans Softw Eng 33:33–53. doi: 10.1109/TSE.2007.256943
4. ISO/IEC (2011) ISO/IEC 14143/1: Information technology – software measurement – FSM. Part 1 Definition of concepts.
5. ISO/IEC (2002) ISO/IEC 20968: Software engineering - Mk II Function Point Analysis - Counting Practices Manual.
6. ISO/IEC (2005) ISO/IEC 24570: Software engineering - NESMA functional size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis.
7. ISO/IEC (2008) ISO/IEC 29881: Information technology – Software and systems engineering – FiSMA 1.1 functional size measurement method.
8. ISO/IEC (2009) ISO/IEC 20926: Software and systems engineering - Software measurement - IFPUG functional size measurement method.
9. ISO/IEC (2011) ISO/IEC 19761: Software engineering - COSMIC: A functional size measurement method.
10. Brandon DM (1998) Earned Value Easily and Effectively. Proj. Manag. J. 29:2
11. Garcia CAL, Hirata CM (2008) Integrating functional metrics, COCOMO II and earned value analysis for software projects using PMBoK. In: Proc. 2008 ACM Symp. Appl. Comput. - SAC. p 820
12. Jin-hua L, Chang-jiang W, Jing L, Qiong L (2008) Earned value project management of model-centric software development. In: Wirel. Commun. Netw. Mob. Comput. WiCOM '08. 4th Int. Conf. pp 1–4
13. Lu X, Bai X, Wang S (2008) Earned value analysis for software project based on function point method. In: 2nd Int. Conf. Manag. Sci. Eng. Manag. pp 301–308
14. Fleming Q, Koppelman J (1998) Earned Value Project Management. CROSSTALK J Def Softw Eng 19–23. doi: 10.1016/j.drudis.2010.11.015
15. Pow-sang JA, Jolay-vasquez E (2006) An Approach of a Technique for Effort Estimation of Iterations in Software Projects. In: Proc. 20th Asia-Pacific Softw. Eng. Conf. pp 367–376
16. Balbin D, Ocrosopoma M, Soto E, Antonio Pow-Sang J (2009) TUPUX: An Estimation Tool for Incremental Software Development Projects. In: AST 2009 Int. E-CONFERENCE Adv. Sci. Technol. Proc. pp 39–43
17. Pow-sang JA, Imbert R (2012) Effort Estimation in Incremental Software Development Projects Using Function Points. In: Comput. Appl. Softw. Eng. Disaster Recover. Bus. Contin. pp 458–465
18. Hussain I, Kosseim L, Ormandjieva O (2010) Towards Approximating COSMIC Functional Size from User Requirements in Agile Development Processes Using Text Mining. In: Proc. Nat. Lang. Process. Inf. Syst. 15th Int. Conf. Appl. Nat. Lang. to Inf. Syst. pp 80–91
19. Santana C, Leoneo F, Vasconcelos A, Gusmão C (2011) Using Function Points in Agile Projects. In: Agil. Process. Softw. Eng. XP. pp 176–191
20. Hussain I, Kosseim L, Ormandjieva O (2013) Approximation of COSMIC functional size to support early effort estimation in Agile. Data Knowl Eng 85:2–14. doi: 10.1016/j.datak.2012.06.005
21. Robiolo G (2011) How Simple is It to Measure Software Size and Complexity for an IT Practitioner? Online Inf Rev 33:40–48.
22. Akca AA, Tarhan A (2013) Run-time measurement of COSMIC functional size for Java business applications: Is it worth the cost? In: Proc. - Jt. Conf. 23rd Int. Work. Softw. Meas. 8th Int. Conf. Softw. Process Prod. Meas. IWSM-MENSURA 2013. pp 54–59
23. Gonultas R, Tarhan A (2015) Run-Time Calculation of COSMIC Functional Size via Automatic Installment of Measurement Code into Java Business Applications. In: Softw. Eng. Adv. Appl. (SEAA), 2015 41st Euromicro Conf. pp 112–118
24. Huijgens H, Bruntink M, Van Deursen A, et al (2016) An Exploratory Study on Functional Size Measurement based on Code. In: Int. Conf. Softw. Syst. Process. pp 56–65
25. Bévo V, Lévesque G, Abran A (1999) Application de la méthode FFP à partir d'une spécification selon la notation UML : COMPTE RENDU DES PREMIERS ESSAIS D'APPLICATION ET QUESTIONS. In: Proc. 9th Int. Work. Softw. Meas. pp 230–242
26. Levesque G, Bevo V, Cao DT (2008) Estimating software size with UML models. In: Proc. C3S2E Conf. pp 81–87
27. Lavazza L, Del Bianco V (2009) A case study in COSMIC functional size measurement: The rice cooker revisited. In: LNCS 5891. pp 101–121
28. Fehlmann TM, Kranich E (2011) COSMIC Functional Sizing based on UML Sequence Diagrams. In: Proceedigns Metr. 2011. p 16
29. Bianco V, Lavazza L, Liu G, Morasca S (2013) Model-based Simplified Functional Size Measurement – an Experimental Evaluation with COSMIC Function Points. In: Proc. EESSMOD/MODELS. pp 13–22
30. Ho VT, Abran A (1999) A Framework for Automatic Function Point Counting from Source Code. In: Proc. IWSM. pp 249–255
31. Akca AA, Tarhan A (2012) Run-time Measurement of COSMIC Functional Size for Java Business Applications: Initial Results. In: 2012 Jt. Conf. 22nd Int. Work. Softw. Meas. 2012 Seventh Int. Conf. Softw. Process Prod. Meas. pp 226–231
32. Sag MA, Tarhan A (2014) Measuring COSMIC software size from functional execution traces of java business applications. In: Proc. - 2014 Jt. Conf. Int. Work. Softw. Meas. IWSM 2014 Int. Conf. Softw. Process Prod. Meas. Mensura 2014. pp 272–281
33. Demirel H, Özkan B (2015) Üç Katmanlı Nesne-İlişkisel Eşleme Mimarisi İçin Otomatik Fonksiyonel Büyüklük Ölçümü (Automated Functional Size Measurement for Three-Tiered Architecture Object-Relational Mapping) - in Turkish. In: Proc. 9th Natl. Softw. Eng. Symp. (9. Ulus. Yazılım Mühendisliği Sempozyumu -UYMS). pp 242–256
34. Tarhan A, Baris O, Icoz GC (2016) A Proposal on Requirements for COSMIC FSM Automation from Source Code. In: Iwsm-Mensura, 5-7 Oct 2016, Berlin. pp 195–200
35. Soubra H, Abran A, Ramdane-Cherif A (2014) Verifying the accuracy of automation tools for the measurement of software with COSMIC - ISO 19761 including an AUTOSAR-based example and a case study. Proc - 2014 Jt Conf Int Work Softw Meas IWSM 2014 Int Conf Softw Process Prod Meas Mensura 2014 23–31. doi: 10.1109/IWSM.Mensura.2014.26
36. Abran A, Fagg P, Woodward C (2015) The COSMIC Functional Size Measurement Method, Version 4.0.1. 98.
37. Kiczales G, Lamping J, Mendhekar A, et al (1997) Aspect Oriented Programming. In: Akçsit M, Matsuoka S (eds) Proc. 11th Eur. Conf. Object-Oriented Program. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 220–242
38. Wang Y, Zhao J (2007) Specifying pointcuts in AspectJ. In: Proc. - Int. Comput. Softw. Appl. Conf. pp 5–10
39. Paton K (1999) Automatic Function Point Counting Using Static and Dynamic Code Analysis. In: Int. Work. Softw. Meas. p 6
40. Kusumoto S, Imagawa M, Inoue K, et al (2002) Function point measurement from Java programs. In: Proc. Int. Conf. Softw. Eng. pp 576–582
41. Jenner MS (2001) COSMIC-FFP 2.0 and UML: Estimation of the Size of a System Specified in UML - Problems of Granularity. In: Proc. 4th Eur. Conf. Soft. Meas. ICT Control. pp 173–184
42. Yin RK (2013) Case Study Research: Design and Methods. Sage Publications, Inc; 5th ed.
43. OMG (2014) Automated Function Points (AFP). 32.

BIOGRAPHIES



AYÇA TARHAN received the B.S. and M.S. degrees in computer engineering from Ege University in 1995 and from Dokuz Eylül University in 1999, and the Ph.D. degree in information systems from Informatics Institute of Middle East Technical University (METU) in 2006.

She was a visiting researcher in 2013–2015 in the Department of Industrial Engineering and Innovation Sciences of Eindhoven University of Technology. Her research interests include internal and external software quality, software metrics, software development methodologies, process maturity, and business process management. Since 2007, she has been working as a Lecturer and an Assistant Professor in Computer Engineering Department of Hacettepe University in Ankara, Turkey.



MUHAMMET ALİ SAĞ received the B.S. and M.S. degrees in computer engineering from Hacettepe University in 2010 and 2014. In 2010–2012, he worked as a software engineer and researcher in Software Technologies Research Institute of the Scientific and Technological Research Council of Turkey (TUBITAK).

Since 2012, he has been working as a system engineer and researcher in Turkish Academic Network and Information Center (ULAKBIM) of TUBITAK.