Baris Özkan[1]

Automated Functional Size Measurement for Three-Tier Object Relational Mapping Architectures

Abstract

Software Functional Size Measurement (FSM) methods are generic. Additional measurement rules, procedures and concept mappings between the generic measurement constructs and specific software artifacts are required in order to improve the measurement objectivity, consistency and enable measurement automation. Although FSM methods provide guidance for measuring software that has well-known architectures, there are opportunities to improve and automate measurements from the artifacts of such architectures. In this paper, a functional size measurement approach for the software that has the Three-Tier Object Relational Mapping architecture is proposed. The approach is based on two components: a mapping between the elements of this software architecture and the COSMIC method constructs and a prototype measurement tool that implements a measurement procedure based on this mapping. In the study the components of the approach are described, the results of a measurement case study are presented and the opportunities and the limitations of the approach are discussed.

Keywords: COSMIC, Object Relational Mapping, Three-Tier architecture, Functional Size Measurement, automation

1. Introduction

Functional size measurement (FSM) is a key practice used in software management. Functional size not only provides valuable information regarding requirements control and project estimation but also constitutes a versatile measure that normalizes various metrics across different activities and processes

[1] Atilim University, Department of Information Systems Engineering, Ankara, Turkey, baris.ozkan@atilim.edu.tr

in software management[2]. FSM methods provide generic software models, concepts and rules which need to be adapted to specific software contexts in order to produce accurate, repeatable and consistent measurement results[3]. Many studies propose concept mappings and procedures for functional size measurement from different software models, specification languages and architectural styles[4]. FSM methods also provide guidance in order to support measures in interpreting the concepts and applying the measurement rules for software architectures. For instance, COSMIC FSM includes examples and explanations for sizing Three-Tier architectures and also provides a guideline for measuring applications that have the Service Oriented Architecture[5].

[2] B. Özkan, O. Turetken, O. Demirörs, *Software Functional Size: For Cost Estimation and More*, in: *Software Process Improvement*, Springer 2008, pp. 59–69.

[3] B. Özkan, O. Demirörs, *Formalization Studies in Functional Size Measurement: How Do They Help?*, in: *Software Process and Product Measurement*, Springer 2009, pp. 197–211.

[4] H. Diab, M. Frappier., R. St-Denis, *A Formal Definition of Function Points for Automated Measurement of B Specifications*, Formal Methods and Software Engineering, Springer 2002, pp. 483–494; D. Hassan, M. Frappier, R. St-Denis, *A Formal Definition of COSMIC-FFP for Automated Measurement of ROOM Specifications*, 4th European Conference Software Measurement and ICT Control 2001; E. Lamma, P. Mello, F. Riguzzi, *A System for Measuring Function Points from an ER–DFD Specification*, "The Computer Journal" 2004, no. 47, pp. 358–372; B. Marín, G. Giachetti, O. Pastor, *Measurement of Functional Size in Conceptual Models: A Survey of Measurement Procedures Based on COSMIC*, Software Process and Product Measurement, Springer 2008, pp. 170–183; B. Marín, O. Pastor, G. Giachetti, *Automating the Measurement of Functional Size of Conceptual Models in an MDA Environment*, Product-Focused Software Process Improvement, Springer 2008, pp. 215–229; C. Monsalve, A. Abran, A. April, *Measuring Software Functional Size from Business Process Models*, "International Journal of Software Engineering and Knowledge Engineering" 2011, no. 21, pp. 311–338; M.A. Sag, A. Tarhan, *Measuring COSMIC Software Size from Functional Execution Traces of Java Business Applications*, Software Measurement and the International Conference on Software Process and Product Measurement (IWSM–MENSURA), Joint Conference of the International Workshop 2014, IEEE, pp. 272–281; C. Symons, A. Lesterhuis, *Guideline for Sizing Service-Oriented Architecture Software, v.1.1*, The Common Software Measurement International Consortium (COSMIC) 2015; T. Edagawa, T. Akaike, Y. Higo, S. Kusumoto, S. Hanabusa, T. Shibamoto, *Function Point Measurement from Web Application Source Code Based on Screen Transitions and Database Accesses*, "Journal of Systems and Software" 2011, no. 84, p. 976–984; 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*, IWSM/MENSURA 2010; A. Živkovič, I. Rozman, M. Heričko, *Automated Software Size Estimation Based on Function Points Using UML models*, "Information and Software Technology" 2005, no. 47, pp. 881–890.

[5] C. Symons, A. Lesterhuis, *Guideline for Sizing Service-Oriented Architecture Software, v.1.1*, The Common Software Measurement International Consortium (COSMIC) 2015; A. Abran, J.-M. Desharnais, S. Oligny, DD. St-Pierre, C. Symons, *The COSMIC Functional Size Measurement Method v.4.0.1*, COSMICON 2015.

Another challenge in FSM is that the manual measurement process usually requires high time and effort costs and is prone to measurement errors by the measurers. An automated measurement process brings in several advantages which help measurers to reduce the cost, time, effort, subjectivity and improve the objectivity, repeatability and consistency in the measurement results[6].

In this study, we propose an FSM approach for the automated measurement of business application software that has the Three-Tier Object Relational Mapping (ORM) architecture, which addresses these two challenges. The proposed approach has been developed for COSMIC FSM which is an ISO/IEC 14143 compliant and internationally recognized FSM method[7]. COSMIC FSM has also detailed guidelines on architectural aspects of FSM[8].

The study explained in this paper is motivated by two observations. The first one is that the Three-Tier architecture -one of the most popular software architectures used in Object Oriented development and which is addressed in the COSMIC official manual and user guide- allows the identification of a set of COSMIC software model concepts (e.g. boundary, layer, scope) with an improved consistency. Secondly, ORM architectural components allow the

[6] B. Özkan, O. Demirörs, *Formalization Studies in Functional Size Measurement: How Do They Help?*, in: *Software Process and Product Measurement*, Springer 2009, pp. 197–211; H. Diab, M. Frappier, R. St-Denis, *A Formal Definition of Function Points for Automated Measurement of B Specifications*, Formal Methods and Software Engineering, Springer 2002, pp. 483–494; D. Hassan, M. Frappier, R. St-Denis, *A Formal Definition of COSMIC-FFP for Automated Measurement of ROOM Specifications*, 4th European Conference Software Measurement and ICT Control 2001; B. Marín, O. Pastor, G. Giachetti, *Automating the Measurement of Functional Size of Conceptual Models in an MDA Environment*, Product-Focused Software Process Improvement, Springer 2008, pp. 215–229; A. Živkovič, I. Rozman, M. Heričko, *Automated Software Size Estimation Based on Function Points Using UML models*, "Information and Software Technology" 2005, no. 47, pp. 881–890; H. Diab, F. Koukane, M. Frappier, R. St-Denis, μ c ROSE: *Automated Measurement of COSMIC-FFP for Rational Rose Real Time*, "Information and Software Technology" 2005, no. 47, pp. 151–166; V.T. Ho, A. Abran, *A Framework for Automatic Function Point Counting from Source Code*, International Workshop on Software Measurement 1999; M.S. Jenner, *Automation of Counting of Functional Size Using COSMIC FFP in UML*, COSMIC Function Points: Theory and Advanced Practices 2011, p. 276; K. Paton, *Automatic Function Point Counting Using Static And Dynamic Code Analysis*, International Workshop on Software Measurement 1999; A.T. Rana Gonultas, *Runtime Calculation of COSMIC Functional Size via Automatic Installment of Measurement Code into Java Business Applications*, SEAA-EUROMICRO 2015.

[7] *ISO/IEC: 19761 Software Engineering-COSMIC-A Functional Size Measurement Method*, International Organization for Standardization, Geneva 2011.

[8] C. Symons, A. Lesterhuis, *Guideline for Sizing Service-Oriented Architecture Software, v.1.1*, The Common Software Measurement International Consortium (COSMIC) 2015; A. Abran, J.-M. Desharnais, S. Oligny, D. St-Pierre, C. Symons, *The COSMIC Functional Size Measurement Method v.4.0.1*, COSMICON 2015.

abstraction of implementation details regarding data persistence and allow the identification of COSMIC software model measurement constructs (e.g. Data Movements, Data Groups) from Object Oriented constructs. Therefore, business application software which has a Three-Tier ORM architecture offers opportunities for automated FSM by taking advantage of the possibilities of automated extraction of COSMIC Software Model constructs from software architecture and code by a systematic analysis. Following our motivation, the proposal has two main components:

- A Mapping of Three-Tier ORM business application architecture to the COSMIC Software Model concepts, which provides a basis for automated measurement;
- An automation tool which implements the proposed mapping.

The proposed mapping relies on COSMIC definitions and guidelines for Three-Tier architectures and extends it with ORM concepts. The tool implements the proposed mapping for a selected set of Three-Tier ORM business application architecture technologies (e.g., development language, platform, components).

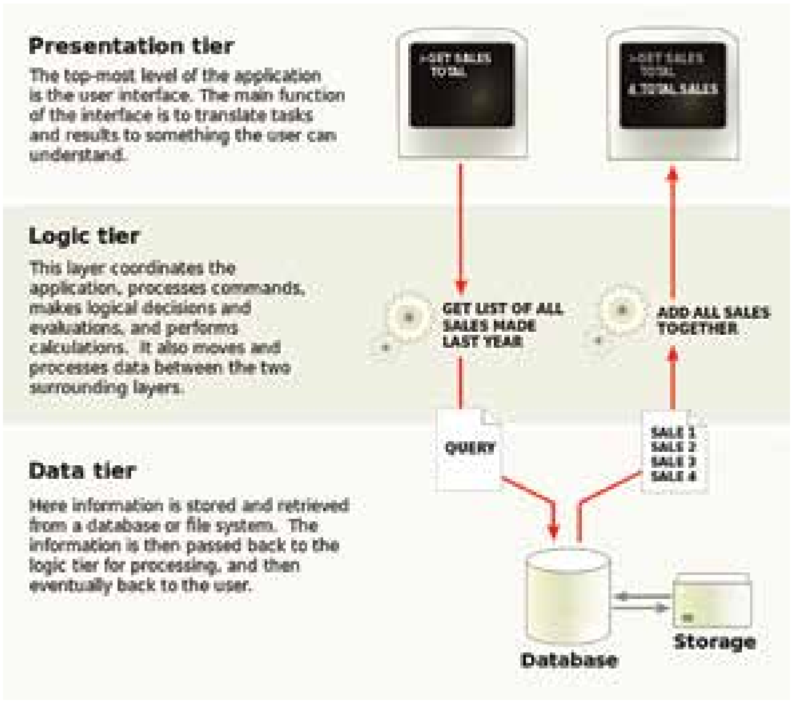
In section two, a background on the Three-Tier architecture and ORM is given. In section three, the mapping between the concepts of the COSMIC Software Model and Three-Tier ORM architecture is given and the automated measurement tool is explained. In section four we present the results of a case study.

2. Background

2.1. Three-Tier Architecture

Three-Tier is an architectural pattern that is based on the separation of the layers of an application. This pattern separates data, business logic and clients (or user interface) from each other. It provides a more secure, flexible and convenient way to match with the business application logic[9]. The layers of the application and their descriptions are given in Fig. 1.

[9] R.N. Taylor, N. Medvidovic, E.M. Dashofy, *Software Architecture Foundations, Theory and Practice*, John Wiley and Sons 2010; A. Aarsten, D. Brugali, G. Menga, *Patterns for Three-Tier Client/Server Applications* 1996.



The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

Figure 1. Three-Tier Layers
Source: the authors' own study.

2.2. Object Relational Mapping

ORM is a mapping technique used in Object Oriented Development which maps domain objects (a.k.a. business data objects) of an application to the data objects of a two dimensional database[10]. It allows persistence and retrieval of data at the object level hiding the implementation of specific details regarding the underlying data structure and access protocol. ORM tries to find the best specific way to combine the Object Oriented Logic to Relational Table Model of the application. ORM also supports the modeling of complex object relationships such as inheritance or composition, thus preserves the properties of the objects and their relationships. Some of the popular ORM frameworks and components common in software community are Hibernate for Java[11], NHibernate for.NET[12]

[10] M. Keith, M. Schnicariol, *Object-Relational Mapping*, Apress 2010.
[11] C. Bauer, G. King, *Hibernate in Action*, Dreamtech Press 2007.
[12] J. Dentler, *NHibernate 3.0 Cookbook*, Packt Publishing, Birmingham 2010.

and *ActiveAndroid ORM* for Android[13] and GORM[14] environments. ORM libraries have standard application programming interfaces (API) through which standardized calls for object retrieval and persistence are made.

Fig. 2 illustrates the mapping of two domain objects (*Customer* and *BankAccount*) to their corresponding data tables. Once this mapping is defined (manually or optionally by ORM) the ORM components synchronize data tables on database to the *Customer*, *BankAccount object instances* throughout the application life cycle without requiring any further involvement of the software developer.

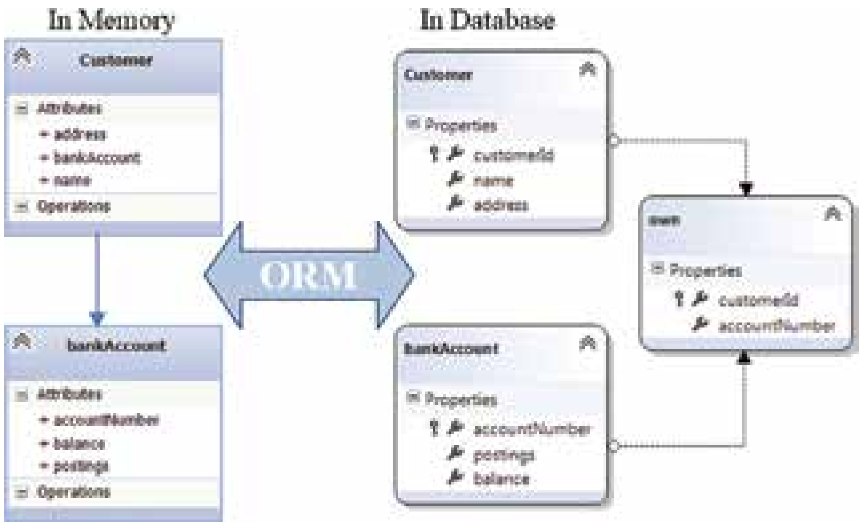


Figure 2. Object-to-Relational-Mapping for Two Classes
Source: the authors' own study.

3. The Mapping and Measurement Tool

3.1. Mapping

The mapping between COSMIC Software Model constructs is the specialization and the extension to the COSMIC Three-Tier model given in the COSMIC

[13] ActiveAndroid, http://www.activeandroid.com/
[14] C. Richardson, *Orm in Dynamic Languages*, Communications of the ACM 52 2009, pp. 48–55.

Method[15]. The COSMIC Three-Tier model and the extended model we use in this study is shown in Fig. 3 and Fig. 4, respectively. The Business Application *Data Layer* (Fig. 4) matches a persistent storage in the COSMIC context (*Data Service Component* in Fig. 3). Similar to the *Business Logic Layer* in the Three-Tier context; *Business Rules Layer* will get and send data from/to the Functional Users of the application or write and read data to/from the storage. The *Presentation-Layer* in Three-Tier and *User-Interface Layer* in COSMIC are functional users of the business layer. Software or a component, a control mechanism or a sensor may also be a Functional User (FU) of business layers.

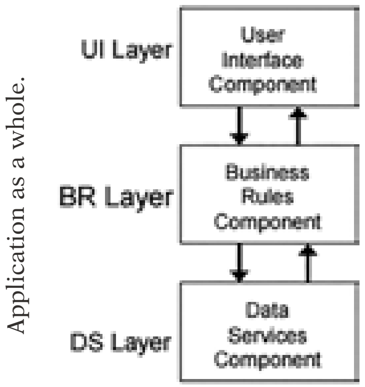


Figure 3. Three Views of the Layers of an Application[17]

Source: the authors' own study.

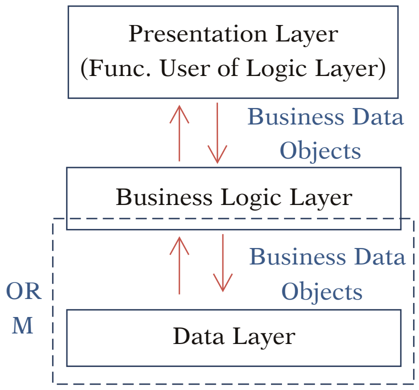


Figure 4. An ORM Based Three-Tier Business Application Data Flow Chart[18]

Application Layer: According to COSMIC, business application software is presented in the application layer which comprises User Interface, Business Rules and Data Service peer components[18]. In our mapping, the three corresponding components (i.e., architectural layers) are Presentation, Business Logic and Data Layers, respectively.

Boundary: According to the COSMIC Generic Software Model[19], the boundary is a conceptual interface between the software and its functional users.

[15] A. Abran, J.-M. Desharnais, S. Oligny, D. St-Pierre, C. Symons, *The COSMIC Functional Size Measurement Method v.4.0.1*, COSMICON 2015.
[16] Ibidem.
[17] Ibidem.
[18] Ibidem.
[19] Ibidem.

According to the Three-Tier ORM architecture, the presentation layer is “thin” in the sense that it does not contain any data processing logic but only conveys data/events between the functional users and the application. Therefore, the boundary corresponds to the presentation layer component of the architecture which includes interface elements and behavior (e.g. GUI, Web Service interface) for its users.

Functional Users are the types of the sender and/or the intended recipient in the Functional User Requirements of software. A COSMIC functional user may be a human, another program, a service or a device[20]. Like these users, the users of business logic are the presentation layers which are used by application users (humans, services, another program etc.).

Object of interest: In COSMIC, an object of interest is defined as a “thing” from a point of view of Functional User Requirements. The domain objects of the Three-Tier architecture are the corresponding object of interests. The storage and the retrieval of these objects are managed by ORM components.

Data Groups: In COSMIC, a data group consists of a unique set of data attributes that describe a single object of interest. In our mapping data groups are attribute subsets of domain objects as configured for ORM components. ORM allows the identification of complex object relations such as inheritance, composition and aggregation, thus we allow complex attribute types.

Functional Process: An elementary component of a set of Functional User Requirements (FURs) comprising a unique, cohesive and independently executable set of data movement types. This corresponds to the *Business Application Task* which is a complete sequence of service (or method) invocations in the business logic layer that starts with a user initiated event at the presentation layer and may utilize functions of the data access layer.

Triggering Entry: The Entry data movement of a functional process that moves a data group generated by a functional user that the functional process needs to start processing. The data group moved by the triggering Entry is generated by a functional user in response to a triggering event[21]. In Three-Tier business applications, a triggering event is any event that starts a *Business Application Task*.

Data Movements:

- Enter (E): A data movement that moves a data group from a functional user across the boundary into the functional process where it is required. In the

[20] Ibidem.
[21] Ibidem.

Three-Tier ORM architecture these movements correspond to movements of data about a domain object from the presentation layer.

- Exit (X): A data movement that moves a data group from a functional process across the boundary to the functional user that requires it. In the Three-Tier ORM architecture these movements correspond to movements of data about a domain object to the presentation layer.
- Read (R): A data movement that moves a data group from the persistent storage into the functional process which requires it. In the Three-Tier ORM architecture these movements correspond to the retrieval of domain objects via standard ORM API methods.
- Write (W): A data movement that moves a data group lying inside a functional process to the persistent storage. In the Three-Tier ORM architecture these movements correspond to the persistence of domain objects via standard ORM API methods.

3.2. Measurement Tool

A prototype measurement tool for the functional size measurement of the Three-Tier ORM architecture has been developed by implementing the mapping given in section 3.1 and a measurement procedure. It has been designed to measure C# and ASP.NET based Three-Tier ORM business applications which use *NHibernate* ORM component[22] and map database relations to domain objects via XML configuration files.

Fig. 5 gives an overview of the structure of the target application that can be measured by the tool. It presents the software artifacts that can be extracted from source-code and configuration files.

Fig. 6 depicts the five main steps of the measurement procedure and the final results generated by the tool. It 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. Finally, after gathering all necessary data, a measurement process is executed by the tool to generate measurement results and other information that support the interpretation of the results.

[22] J. Dentler, *NHibernate 3.0 Cookbook*, Packt Publishing, Birmingham 2010.

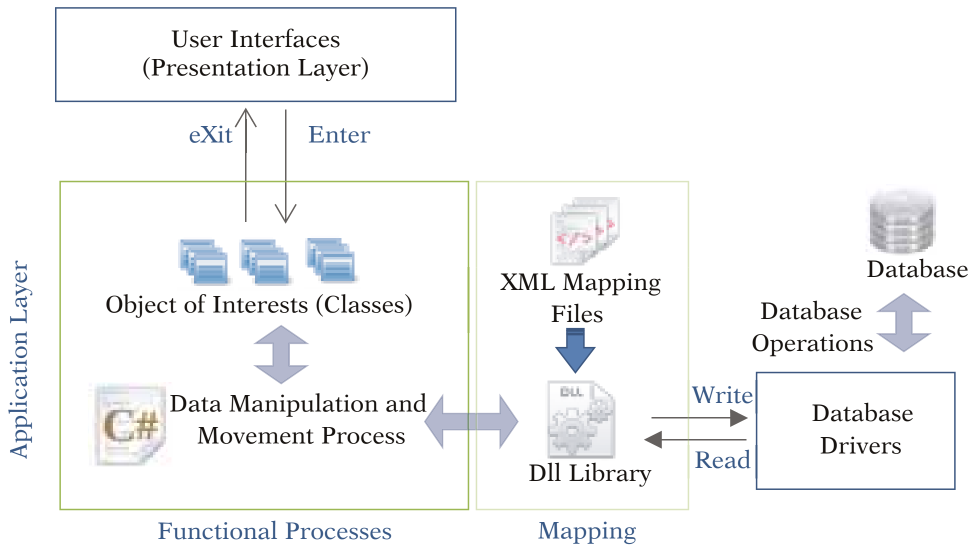


Figure 5. Target Application Structure

Source: the authors' own study.

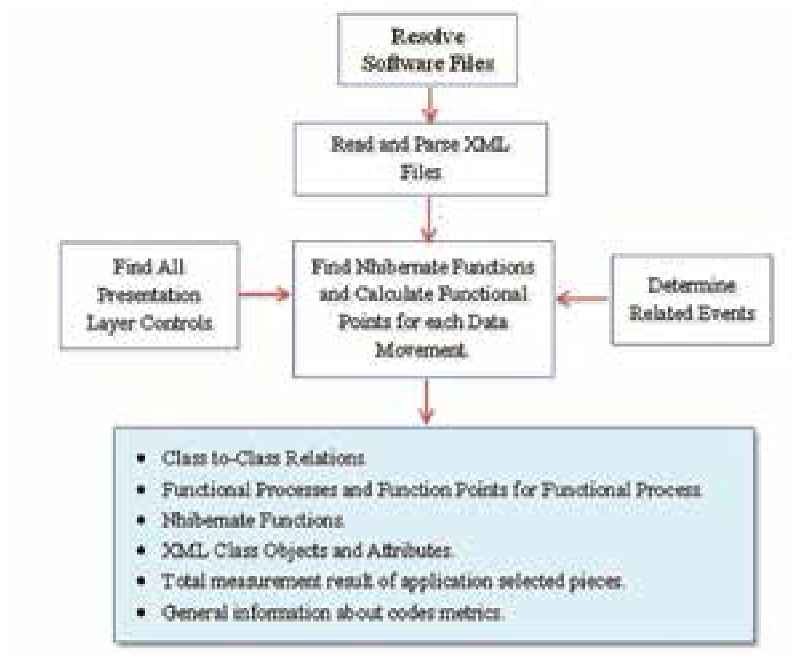
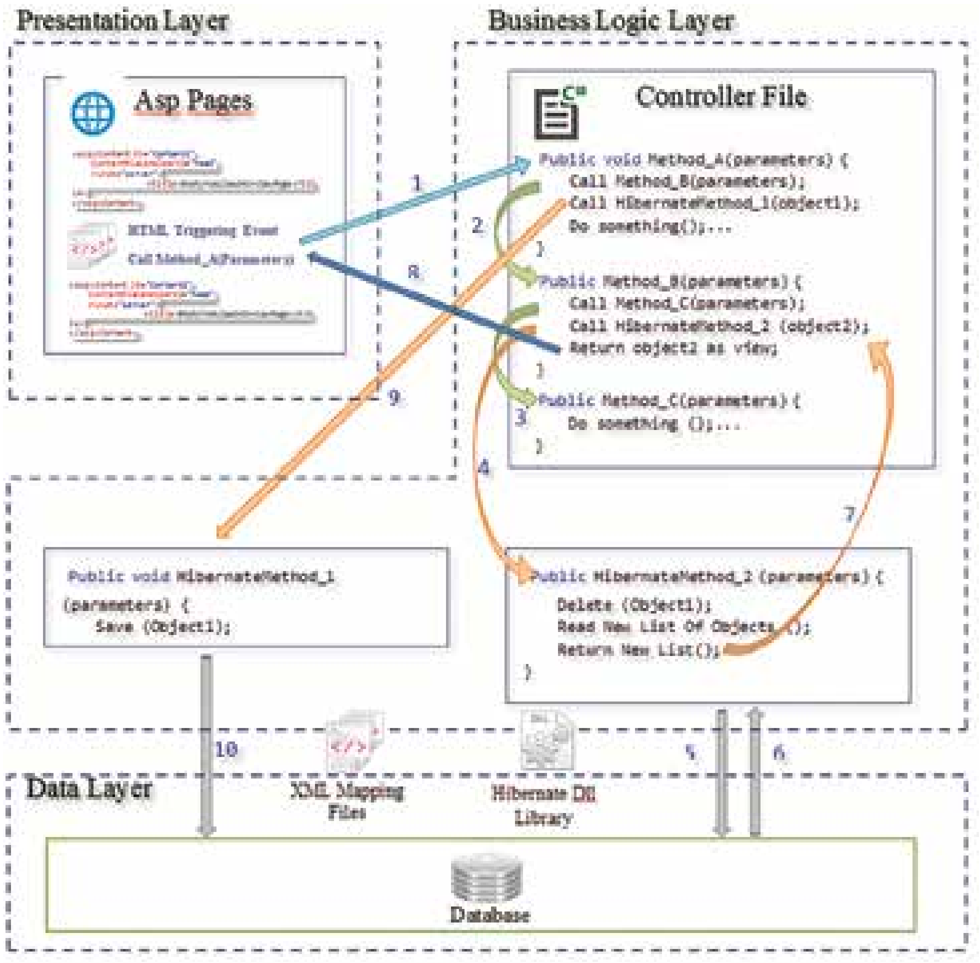

Resolve Software Files
Read and Parse XML Files
Find All Presentation Layer Controls
Find Nhibernate Functions and Calculate Functional Points for each Data Movement
Determine Related Events
Class to-Class Relations
Functional Processes and Function Points for Functional Process
Nhibernate Functions
XML Class Objects and Attributes
Total measurement result of application selected pieces
General information about codes metrics

Figure 6. Measurement Processes of the Measurement Tool

Source: the authors' own study.

Fig. 7 illustrates the source code-architectural layers relation of the Three-Tier application to be measured by our tool and shows how they relate to COSMIC concepts with respect to our mapping. The numbered steps show a typical flow of events in a functional process once a triggering event has been started. Accordingly, data movements 1, 8, 10 and 5, 6 can be defined as E, X, W and R data movements, respectively.



```
Public void Method_A(parameters) {
    Call Method_B(parameters);
    Call HibernateMethod_1(object1);
    Do something();...
}
Public Method_B(parameters) {
    Call Method_C(parameters);
    Call HibernateMethod_2 (object2);
    Return object2 as view;
}
Public Method_C(parameters) {
    Do something ();...
}
```

```
Public void HibernateMethod_1
(parameters) {
    Save (Object1);
```

```
Public HibernateMethod_2 (parameters) {
    Delete (Object1);
    Read New List Of Objects ();
    Return New List();
}
```

Figure 7. Event Trigger and the Consequent Operations from the Measurement Tool Perspective

Source: the authors' own study.

4. Case Study

A case study has been conducted to evaluate the approach based on the comparison of the measurement results obtained by the tool to the manual measurement results obtained by measurers.

4.1. Business Application

The selected Cuya-hoga application is an open source portal management software that has the Three-Tier ORM architecture[23]. It is implemented in the Visual Studio environment. It contains all necessary program files such as code files, xml files and web pages and 16.7 Source Lines of Code.

4.2. Data Collection

Functional User Requirements of Cuya-hoga are derived from software code files, user screens and other product documentation as there was not any explicit requirement specification found on the product web page[24]. Manual measurements were performed by a COSMIC Certified measurement expert, which were later reviewed by the authors of this paper. A total of 29 hours were spent on this measurement and review process. It took less than a minute to configure the tool and get the results on a standard PC.

4.3. Results

The comparison of the manual measurement results (M) extracted by the measurers; and the automated measurement results (A) provided by the measurement tool automatically is based on data movements and FPs. Fig. 8 shows the identified measurement constructs in a Venn diagram. Accordingly, the common constructs set (C) is derived from the intersection of M and A, missing (m) constructs which could not be detected by the tool and extra (e) constructs which were not included in the manual measurements but identified by the tool. In the analysis of the results, we assumed the manual measurements were exact and

[23] Cuya Hoga, http://cuyahoga-project.org/
[24] Ibidem.

all FPs, Data Movements and Triggering Events are valid and comply with the rules in the COSMIC Manual.

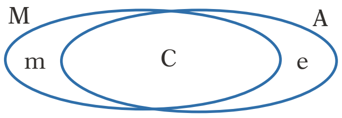


Figure 8. Data Movement Classification of the Measurement

Source: the authors' own study.

The results for C, m and e are shown in Table 1. The *detection accuracy* is calculated with the equation of *C*100/M* which indicates the percentage of the measurement constructs that were accurately identified by the tool. In other words, it gives the percentage coverage of C in M. The number of common items can be identified by the equation of *A-e* or *M-m*. In the identification of missed (m) and extra (e) items FPs in M were taken into account, thus, false positives were taken into account in the calculations.

Table 1. Manual and Automated Result Comparison Table

	E	W	R	X	Total DM	Total FP
Manual/ Actual	71	111	140	88	410	86
Auto:	39	116	167	63	385	93
Missed:	33	5	5	25	68	11
Extra:	5	14	28	4	51	18
Common:	34	102	139	59	334	75
Detection Accuracy:	47.9%	91.9%	99.3%	67.0%	81.5%	87.2%

Source: the authors' own study.

As it can be seen from the values in Table 1, Fig. 9 and Fig. 10, the automated W and R data movement detection accuracy rates are much higher than E and X data movements.

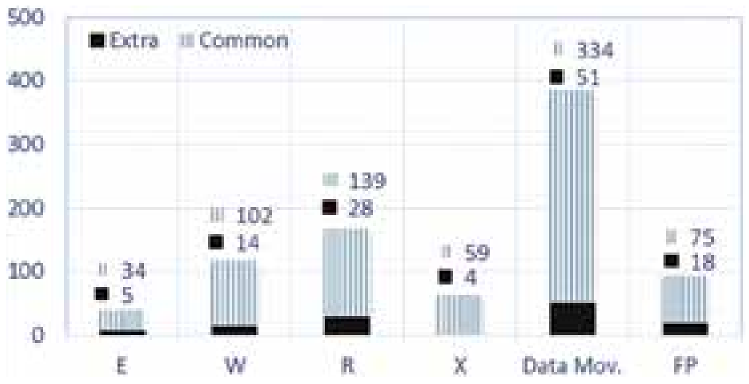


Figure 9. Comparison of Common (C) and Extra (e) Items by Their Type
Source: the authors' own study.

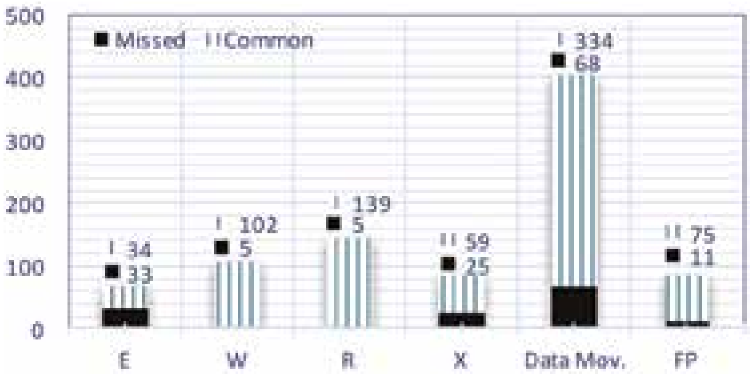


Figure 10. Comparison of Common (C) and Missing (e) Items by Their Type
Source: the authors' own study.

4.4. Discussion of the Results

After the analysis of the source code in detail, we concluded that the high accuracy detection rates for R and W data movements relate to the fact that retrieval (R) and persistence (W) of all application business data were implemented using ORM API properly, which provides a standardized and uniform way for capturing data movements. However, the graphical user interface (GUI) implementation was not uniform where a critical number of data movements that pass between GUI and the logic layer of the application (see Fig. 5) could not be detected. For example, the presentation layer was partly implemented in the JavaScript language and many data groups and triggering events were not resolved from the source. The majority of extra FPs identified by the tool were internal steps of a functional process identified by the manual measurer. This result is not surprising because the mapping identifies a separate FP (business task) for all consequent data movements after an event (button or link clicks) that occurs at the GUI. However, it is not capable of relating business tasks to each other. There was not a systemic cause found for the missing data movements.

One validity threat in the evaluation of the results is the errors or miscalculations that could be made in the manual measurement by the measurer. In order to avoid and keep the errors minimum, measurements by a COSMIC certified measurer were reviewed and used. Another threat is the generalizability of the results of this case study. Firstly, the case study subject was a small project; secondly, it was an open source project where there was not any control on code standardization.

Another threat is related to the measurement time improvement obtained through automation. The time spent on manual measurement is rather high. There was not any requirements specification for the case study application, which would be the main source for capturing FURs. This might cause a longer measurement time for the measurer of this application.

5. Conclusion

In this study, we introduced a direct approach to the measurement of ORM based Three-Tier business applications. The approach was implemented in an automated measurement prototype and evaluated in a case study. The initial results and improvement opportunities from the application of the tool are promising in the sense that it can be used to provide size information quickly and with an acceptable accuracy rate to its users such as project managers.

The tool also provides extra information such as file counts, objects of interest and relations among them, events and triggering components, method traces and related parameters and return types for each method call. Such information can be useful in a number of ways that needs investigation: it can help measurers and project managers to further analyze the relationship between the number and types of components.

The tool has several limitations. Since the tool measures the functional size from the source code, it may not be effectively used before FURs are implemented.

Thus, it may not provide much value in settings where detailed requirements exist and measurements are mainly performed for estimation. On the contrary, it can provide a significant value in software development environments where there is not any or up-to-date FURs document available for measurement. It can also be effectively used in settings where the development process is not driven by detailed requirements but by executable software releases such as agile or evolutionary development models.

As described above, the tool is able to detect data movements and functional processes with high accuracy rates only if they are implemented in a standardized way. When programmers do not follow a standard convention for implementing the program codes, it limits the capability of the measurement tool to detect and trace data movements.

Finally, the tool was not validated in large scale industrial business applications and the implementation was limited to a selected set of application technology configuration (C#, ASP.NET, Nhibernate with xml mapping). We recommend performing further case studies with open source and commercial software of different sizes, technology as future work.

References

Aarsten A., Brugali D., Menga G., *Patterns for Three-Tier Client/Server Applications* 1996.

Abran A., Desharnais J.-M., Oligny S., St-Pierre D., Symons C., *The COSMIC Functional Size Measurement Method v.4.0.1*, COSMICON 2015.

ActiveAndroid, http://www.activeandroid.com/

Bauer C., King G., *Hibernate in Action*, Dreamtech Press 2007.

Cuya Hoga, http://cuyahoga-project.org/

Dentler J., *NHibernate 3.0 Cookbook*, Packt Publishing, Birmingham 2010.

Diab H., Frappier M., St-Denis R., *A Formal Definition of Function Points for Automated Measurement of B Specifications*, Formal Methods and Software Engineering, Springer 2002, pp. 483–494.

Diab H., Koukane F., Frappier M., St-Denis R., μ c ROSE: *Automated Measurement of COSMIC-FFP for Rational Rose Real Time*, "Information and Software Technology" 2005, no. 47, pp. 151–166.

Edagawa T., Akaike T., Higo Y., Kusumoto S., Hanabusa S., Shibamoto T., *Function Point Measurement from Web Application Source Code Based on Screen Transitions and Database Accesses*, "Journal of Systems and Software" 2011, no. 84, p. 976–984.

Hassan D., Frappier M., St-Denis R., *A Formal Definition of COSMIC-FFP for Automated Measurement of ROOM Specifications*, 4th European Conference Software Measurement and ICT Control 2001.

Ho V.T., Abran A., *A Framework for Automatic Function Point Counting from Source Code*, International Workshop on Software Measurement 1999.

ISO/IEC: 19761 Software Engineering-COSMIC-A Functional Size Measurement Method, International Organization for Standardization, Geneva 2011.

Jenner M.S., *Automation of Counting of Functional Size Using COSMIC FFP in UML*, COSMIC Function Points: Theory and Advanced Practices 2011, p. 276.

Keith M., Schnicariol M., *Object-Relational Mapping*, Apress 2010.

Lamma E., Mello P., Riguzzi F., *A System for Measuring Function Points from an ER–DFD Specification*, "The Computer Journal" 2004, no. 47, pp. 358–372.

Marín B., Giachetti G., Pastor O., *Measurement of Functional Size in Conceptual Models: A Survey of Measurement Procedures Based on COSMIC*, Software Process and Product Measurement, Springer 2008, pp. 170–183.

Marín B., Pastor O., Giachetti G., *Automating the Measurement of Functional Size of Conceptual Models in an MDA Environment*, Product-Focused Software Process Improvement, Springer 2008, pp. 215–229.

Monsalve C., Abran A., April A., *Measuring Software Functional Size from Business Process Models*, "International Journal of Software Engineering and Knowledge Engineering" 2011, no. 21, pp. 311–338.

Ozkan B., Turetken O., Demirors O., *Software Functional Size: For Cost Estimation and More*, in: Software Process Improvement, Springer 2008, pp. 59–69.

Ozkan B., Demirors O., *Formalization Studies in Functional Size Measurement: How Do They Help?*, in: Software Process and Product Measurement, Springer 2009, pp. 197–211.

Paton K., *Automatic Function Point Counting Using Static and Dynamic Code Analysis*, International Workshop on Software Measurement 1999.

Rana Gonultas A.T., *Run-time Calculation of COSMIC Functional Size via Automatic Installment of Measurement Code into Java Business Applications*, SEAA-EUROMICRO 2015.

Richardson C., *Orm in Dynamic Languages*, Communications of the ACM 52 2009, pp. 48–55.

Sag M.A., Tarhan A., *Measuring COSMIC Software Size from Functional Execution Traces of Java Business Applications*, Software Measurement and the International Conference on Software Process and Product Measurement (IWSM–MENSURA), Joint Conference of the International Workshop 2014, IEEE, pp. 272–281.

Soubra H., Chaaban K., *Functional Size Measurement of Electronic Control Units Software Designed Following the AUTOSAR Standard: A Measurement Guideline Based on the COSMIC ISO 19761 Standard*, IWSM/MENSURA 2010.

Symons C., Lesterhuis A., *Guideline for Sizing Service-Oriented Architecture Software, v.1.1*, The Common Software Measurement International Consortium (COSMIC) 2015.

Taylor R.N., Medvidovic N., Dashofy E.M., *Software Architecture Foundations, Theory and Practice*, John Wiley and Sons Inc. 2010.

Živkovič A., Rozman I., Heričko M., *Automated Software Size Estimation Based on Function Points Using UML models*, "Information and Software Technology" 2005, no. 47, pp. 881–890.