IMPROVING SOFTWARE FUNCTIONAL SIZE MEASUREMENT

Serge Oligny[1], Alain Abran[1], Denis St-Pierre[2]

[1]: UQAM's Software Engineering Management Research Laboratory (http://www.lrgl.uqam.ca/), Montréal, oligny.serge@uqam.ca and abran.alain@uqam.ca.
[2]: dstpierr@crim.ca.

Abstract

Software functional size measurement is regarded as a key aspect in the production, calibration and use of software engineering productivity models because of its independence of technologies and of implementation decisions. In 1997, Full Function Points (FFP) was proposed as a method for measuring the functional size of real-time and embedded software. Since its introduction, the FFP measurement method has been field-tested in many organizations which have provided feedback on ways to improve it.

Based on this feedback and in association with the Common Software Measurement International Consortium (COSMIC), version 2.0 of the COSMIC-FFP measurement method will be released in October 1999 for field-testing. This paper describes the new features of COSMIC-FFP version 2.0, including: a generic software model adapted for the purpose of functional size measurement, a two-phase approach to functional size measurement (mapping and measurement), a simplified set of base functional components (BFC) and a scalable aggregation function. Through its generic software model of functional users requirements, version 2.0 of the COSMIC-FFP measurement method is applicable to a broad range of software, including embedded, MIS, middleware and system software.

1. INTRODUCTION

Full Function Points (FFP) is a measurement method introduced in 1997 [1] for measuring the functional size of real-time and embedded software. Since its introduction, this measurement method has been field-tested and is currently used in many organizations [2, 3, 4].

The feedback received from the field tests [3], as well as from day-to-day usage of the method, the prospect of ISO certification in accordance with the recently released standard 14143 [5] and the momentum provided by the Common Software Measurement International Consortium (COSMIC) [6] have led to the development of version 2.0 of this measurement method. This paper presents some of the key improvements introduced by COSMIC-FFP, version 2.0.

Section 2 summarizes the feedback gathered from the industrial users, section 3 presents four key improvements, and section 4 puts these improvements in the historical context of functional size measurement evolution.

2. MOTIVATION FOR IMPROVEMENTS

Since its introduction, the Full Function Points measurement method has not only been applied to real-time or embedded software, but also to a variety of technical and system software and to some MIS software. Applying the method to such a wide range of software revealed:

a) The need to improve the mechanism by which the functional user requirements embedded in the software engineering artifacts are mapped onto the base functional components (BFC) [5] to measure the functional size of the corresponding software;

b) The need to refine the concept of "software boundary" in order to address functional user requirements allocated not only to the pieces of software interacting with end-users (application software), but also to pieces of supporting software which are part of the operating environment when all these pieces are part of a given project;

c) The need to simplify the set of BFC used to measure the functional size of software.

d) The need to increase the flexibility of the measurement method in order to offer a scalable result.

These requirements, along with those required by ISO compliance, were used as guides to produce version 2.0 of the COSMIC-FFP measurement method.

3. <u>COSMIC-FFP VERSION 2.0</u>

Many of the improvements incorporated into version 2.0 of the COSMIC-FFP measurement method are marked by explicit concepts. Such an approach means that the concepts are distinguished from the rules and procedures used to implement them. The measurement method therefore gains in flexibility and usability by allowing the practitioners to quickly grasp the aim of the rules and, consequently, adapt them to the measurement context of their organization.

Requirement a), in section 2 above, was mostly fulfilled by refining the measurement process itself. The result is presented in section 3.1. Requirement b) was fulfilled by enriching the software model used by the measurement method. The result is presented in section 3.2. The simplification of the BFC set (requirement c) above) led to a more rigorous definition of the elements contributing to the functional size of a piece of software, and is presented in section 3.3. Better scalability of the result (requirement d) above) was also achieved through this newer definition of the pieces contributing to the functional size of software; the result is presented in section 3.4.

3.1 A measurement process model

In essence, the COSMIC-FFP measurement method consists of the application of a set of rules and procedures to a given piece of software; the result of the application of these rules and procedures is a numerical figure representing the functional size of the software.

The method is designed to be independent of the implementation decisions embedded in the operational artifacts of the software to be measured. To achieve this characteristic, measurement is performed on a generic software model onto which functional user requirements found in the software artifacts are mapped. Figure 3.1, below, depicts this process.

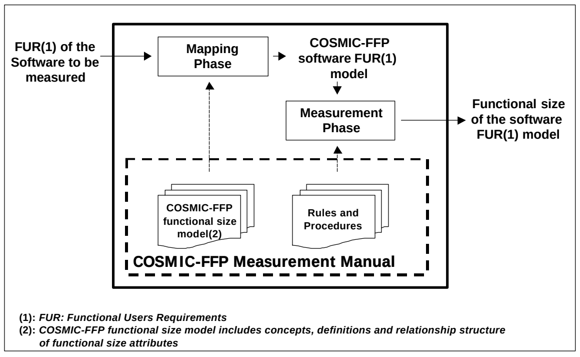
FUR(1) of the Software to be measured
Mapping Phase
COSMIC-FFP software FUR(1) model
Measurement Phase
Functional size of the software FUR(1) model
COSMIC-FFP functional size model(2)
Rules and Procedures
COSMIC-FFP Measurement Manual
(1): FUR: Functional Users Requirements
(2): COSMIC-FFP functional size model includes concepts, definitions and relationship structure of functional size attributes

Figure 3.1 – COSMIC-FFP measurement process model

This model illustrates that, prior to applying the measurement rules and procedures, the software to be measured must be mapped onto a generic model (the COSMIC-FFP software model) that captures the concepts, definitions and relationships (functional structure) required for a functional size measurement exercise.

3.2 A generic software model

A key aspect of software functional size measurement lies in the establishment of what is considered to be part of the software and what is considered to be part of the operating environment of the software. As a functional size measurement method (FSM as defined in [5]), COSMIC-FFP aims at measuring the size of software based on identifiable functional user requirements. Depending on how these requirements are allocated, the resulting software might be implemented in a number of pieces. While all the pieces exchange data, they will not necessarily operate at the same "level". The COSMIC-FFP software context model, illustrated in Figure 3.2, recognizes this general configuration by providing rules to identify different LAYERS of software.

As illustrated in Figure 3.2, the functional user requirements in this example are allocated to three distinct pieces, each exchanging data with another through a specific organization: one piece of the software lies at the application level and exchanges data with the software's users and with a second piece lying at the operating system level. In turn, this second piece of the software exchanges data with a third piece lying at the device driver level. This last piece then exchanges data directly with the hardware. The COSMIC-FFP context model associates each level with a specific LAYER. Each layer possesses an intrinsic boundary for which specific users are identified. The functional size of the software described through the functional user requirements is thus broken down into three pieces, each piece receiving parts of the functional user requirements.

Four important concepts relating to this model are defined.

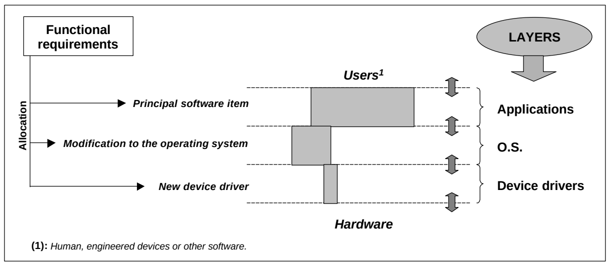

Figure 3.2 – COSMIC-FFP software context model

Concept 1 – Layers
The software to be measured can be partitioned into one or more pieces so that each piece operates at a different level of functional abstraction in the software's operating environment. There is a relationship between each particular level of abstraction based on the data exchanged between them. Each such level is designated as a distinct layer. Each layer encapsulates functionality useful to a client layer and uses the functionality provided by supporting layers. One of these layers interacts with external end-users through I/O hardware and another interacts with storage hardware through device drivers.

Concept 2 – Boundary
Within each identified layer, the piece of software to be measured can be clearly distinguished from its surrounding peers by a boundary. Furthermore, an implicit boundary exists between each identified layer. The software boundary is therefore a set of criteria, perceived through the functional requirements of the software, which allows a clear distinction to be made between the items that are part of the software (inside the boundary) and the items that are part of the software operating environment (outside the boundary). By convention, all users of a piece of software lie outside the boundary of this software.

Concept 3 – Software users
Within each identified layer, it is possible to identify one or more users benefiting from the functionality provided by the piece of software lying inside the layer. By definition, users can be human beings, engineered devices or other software. Also by definition, pieces of the measured software lying inside the immediate neighboring layers are considered as users (considered as "other software").

Concept 4 - Functional requirements
Software purposes can be formally described through a finite set of requirements. The parts of these requirements describing the nature of the functions to be provided are designated as functional requirements and institute the exclusive perspective from which the functional size of the software is to be measured. The parts of the requirements describing how the software functions are to be implemented are NOT considered for the purposes of measuring the functional size of the software.

Figure 3.3 illustrates the software model proposed by the COSMIC-FFP measurement method. This model describes the perspective from which the pieces of software identified within each layer are perceived for the purpose of functional size measurement.

According to this model, software functional requirements are implemented by a set of functional processes. Each of these functional processes is an ordered set of sub-processes performing either a data movement or a data manipulation.

The COSMIC-FFP generic software model distinguishes four types of data movement sub-process: entry, exit, read and write. All data movement sub-processes move data belonging to exactly one data group. *Entries* move the piece from outside the software I/O boundary to the inside; *exits* move it from inside the software toward the outside of the I/O boundary; reads and writes move data across the storage boundary. These relationships are illustrated in Figure 3.4.

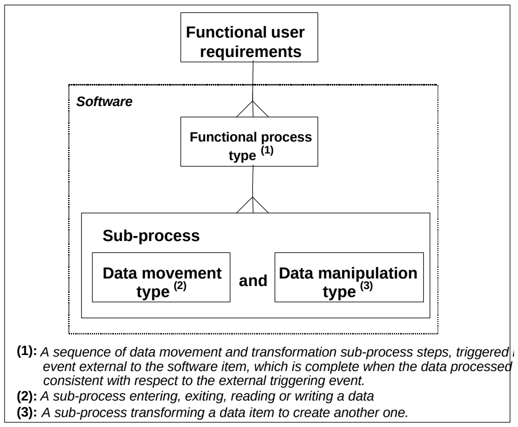
Functional user requirements
Software
Functional process type (1)
Sub-process
Data movement type (2)
and
Data manipulation type (3)
(1): A sequence of data movement and transformation sub-process steps, triggered
event external to the software item, which is complete when the data processed
consistent with respect to the external triggering event.
(2): A sub-process entering, exiting, reading or writing a data
(3): A sub-process transforming a data item to create another one.

Figure 3.3 – COSMIC-FFP generic software model

By using the concepts, definitions and structure of the COSMIC-FFP measurement method, the functional user requirements embedded in the artifacts of a piece of software are mapped onto the COSMIC-FFP software model, thereby instantiating it. This instantiated model will contain all the elements required for measuring its functional size, while hiding information not relevant to the measurement of its functional size.

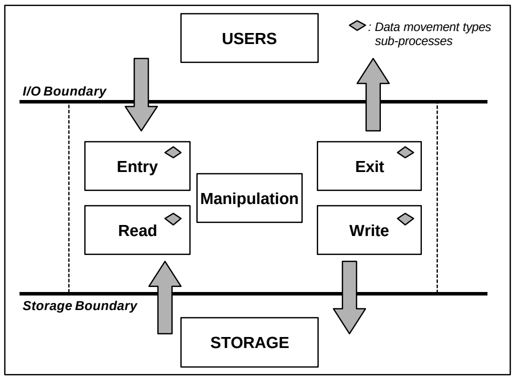

Figure 3.4 – COSMIC-FFP sub-process types

The COSMIC-FFP measurement rules and procedures are then applied to this instantiated COSMIC-FFP model in order to produce a numerical figure representing the functional size of the software.

Therefore, two distinct and related phases are necessary to measure the functional size of software: mapping the functional user requirements embedded in the artifacts of the software to be measured onto the COSMIC-FFP software model and then measuring the specific elements of this software model.

3.3 Measurement system

The COSMIC-FFP measurement system comprises a measurement principle, base functional components (BFC) and a standard unit of measure.

Measurement principle

The COSMIC-FFP measurement phase uses as input an instance of the COSMIC-FFP software model and, through a defined set of rules and procedures, produces a numerical figure the magnitude of which is directly proportional to the functional size of the model, based on the following principle:

> The functional size of a piece of software is directly proportional to the number of its data-moving sub-processes.

COSMIC-FFP measurement principle

By convention, this numerical figure is then extended to represent the functional size of the software itself.

Two elements characterize this set of rules and procedures: the BFCs which constitute the arguments of the measurement function and the standard unit of measurement, which is the yardstick defining one unit of functional size (one COSMIC functional size unit or COSMIC-FSU).

Base functional components

Version 2.0 of COSMIC-FFP uses only four base functional components (BFC): entry, exit, read and write. Data manipulation sub-processes are not used as base functional components. Version 2.0 of COSMIC-FFP assumes, as an acceptable approximation for many types of software, that the functionality of this type of sub-process is represented among the other four types of sub-process.

Standard unit of measurement

The standard unit of measurement, that is, 1 COSMIC-FSU, is defined by convention as equivalent to one single data movement at the sub-process level.

The COSMIC-FFP measurement method does not presume to measure all aspects of software size. Dimensions of software "size" that differ from elementary data movements are not captured by this measurement method. A constructive debate on this matter would first require commonly agreed upon definitions of the other elements and dimensions within the ambiguous concept of "size" as it applies to software. Such definitions are still, at this point, the object of further research and much debate.

3.4 Scalability

From the level of granularity offered by the sub-process types, version 2.0 of the COSMIC-FFP measurement method offers a scalable result at the layer level, through the use of an aggregation function.

Thus, the functional size of any functional process is defined as the arithmetic sum of the sizes of its constituent sub-processes. Results can also be aggregated by layers by simply adding the functional sizes of the constituent functional processes.

It is worth noting that the smallest theoretical functional size for a piece of software is 2 COSMIC-FSU, since any software exhibits, at least, one input (entry or read) and one output (exit or write). Furthermore, there is no upper limit to the functional size of a piece of software and, notably, there is no upper limit to the functional size of any of its measured functional processes.

4. CONCLUSION

Functional size measurement of software emerged 20 years ago from the empirical results gathered on a sample of MIS applications [7]. As it gained wider practitioner acceptance in the ‘80s, the methods then available have been regularly criticized, notably for their inability to correctly measure the size of real-time software [8, 9, 10, 11, 12, 13, 14, 15, 16]. Although many alternatives have been proposed, from the mid-’80s to the mid-’90s to address this problem, none of them seemed to have gained sufficient recognition from practitioners to be used on a regular basis across a large number of organizations and in many countries.

Version 1.0 of Full Function Points was proposed in 1997 as a public domain alternative to solve this persistent difficulty. Since then, field tests and repeated usage in many organizations throughout North America, Europe and Asia [2, 3, 4] have demonstrated that FFP offers meaningful results not only to measure the functional size of real-time and embedded software, but also to measure the functional size of a wide range of technical and system software and, in some cases, of MIS software as well.

The improvements proposed in version 2.0 of the COSMIC-FFP measurement method are aimed at supporting these encouraging results by providing enhanced applicability through a) a solid metrological framework, and b) explicit concepts enabling practitioners to perform measurements in a more efficient manner. Field tests are currently under way to verify that those objectives have been met.

5. ACKNOWLEDGMENTS

The authors wish to thank Nortel, Bell Canada and Hydro-Québec, as well as other industrial partners, for providing project funds, industrial data and valuable feedback from real-time software practitioners.

The authors would also like to thank the members of the COSMIC Group [6], notably Charles Symons, Pam Morris and Grant Rules, for the thoughtful comments, suggestions and ideas they provided during the development of version 2.0 of the COSMIC-FFP measurement method.

UQAM’s Software Engineering Management Research Laboratory co-sponsored the research leading to this publication. The Laboratory is supported through a partnership with Bell Canada. Additional funding for the Laboratory is provided by the Natural Sciences and Engineering Research Council of Canada.

6. REFERENCES

[1] St-Pierre D., Maya M., Abran A., Desharnais J.M., Bourque P., “Full Function Points: Counting Practices Manual”, Technical Report 1997-04, Université du Québec à Montréal, Montréal, Canada, 1997. Available at http://www.lrgl.uqam.ca/ffp.html.

[2] Morris P., Desharnais J.M., “Measuring all software, not just what the business uses”, Proceedings of the IFPUG Fall Conference, Orlando, Florida, September 21-25, 1998.

[3] Oligny S., Abran A., Desharnais J.M., Morris P., “Functional size of real-time software: overview of field tests”, Proceedings of the 13th International Forum on COCOMO and software cost modeling, Los Angeles, California, October 6-8, 1998.

[4] Kececi N., Li M., Smidts C., “Function point analysis: an application to a nuclear reactor protection system”, Proceedings of the Probability Safety Assessment ’99 Conference, Washington, D.C., August 22-25, 1999.

[5] ISO/IEC 14143-1: Information technology – Software measurement – Functional size measurement – Definition of concepts, October 1997.

[6] See www.cosmicon.com for details.

[7] Albrecht A.J., "Measuring Application Development Productivity", Proc. of the IBM Applications Development Symposium, Monterey, California, 1979.

[8] Conte S.D., Shen V.Y., Dunsmore H.E., *Software Engineering Metrics and Models*, Benjamin Cummins Publishing, 1986, 396 pages.

[9] Galea S., *The Boeing Company: 3D Function Point Extensions, V. 2.0, Release 1.0*, Boeing Information and Support Services, Research and Technology Software Engineering, June 1995.

[10] Grady R.B., *Practical software metrics for project management and process improvement*, Prentice Hall, New-Jersey, 1992, 270 pages.

[11] Hetzel B., *Making software measurement work*, QEB Publishing Group, 1993, 290 pages.

[12] Ince D.C., "History and industrial applications", in Fenton N.E., *Software Metrics: A Rigourous Approach*, Chapman & Hall, UK, 1991, 337 pages.

[13] Jones C., *A short history of Function Points and Feature Points*, Software Productivity Research Inc., Cambridge, Mass., 1988.

[14] Jones C., *Applied Software Measurement – Assuring Productivity and Quality*, McGraw-Hill, 1991, 493 pages.

[15] Kan S.H., *Metrics and Models in Software Quality Engineering*, Addison-Wesley, 1993, 344 pages.

[16] Whitmire S.A., "3-D Function Points: Scientific and Real-Time Extensions to Function Points", Proc. of the 1992 Pacific Northwest Software Quality Conference, 1992.