Information Theory-based Functional Complexity Measures and Functional Size with COSMIC-FFP

Alain Abran, Olga Ormandjieva**, Manar Abu Talib***

*École de Technologie Supérieure - ETS

1100 Notre-Dame Ouest, Montréal, Canada H3C 1K3

**Concordia University

1455 de Maisonneuve Blvd. W. Montreal, Quebec H3G 1M8, Canada

aabran@ele.etsmtl.ca , ormandj@cse.concordia.ca , m_abutal@cse.concordial.ca

Abstract:

This paper presents an exploratory study of related concepts across information theory-based measures and functional size measures. Information theory-based software measurement has been used in the design of an entropy-based measure of functional complexity in terms of an amount of information based on some abstraction of the interactions among software components. As a functional size measurement method, COSMIC-FFP, adopted in 2003 as the ISO/IEC 19761 standard, measures software functionality in terms of the data movements across and within the software boundary. In this paper, we explore some of the links between the two types of measures, and, in particular, the similarities (and differences) between their generic model of software functionality, their detailed model components taken into account in their respective measurement processes and, finally, their measurement function. Some further investigation avenues are also identified for extending the use of functional size measures for reliability estimation purposes and for scenario-based black-box testing.

Keywords

Software Complexity, Functional Complexity, Information Theory, Entropy, Functional Size Measurements, COSMIC-FFP, ISO 19761

1 Introduction

The functional size measurement method was developed by the Common Software Measurement International Consortium (COSMIC). COSMIC-FFP focuses on the "user view" of functional requirements and is applicable throughout the development life cycle, right from the requirements phase to the implementation and maintenance phases. This measurement method has been designed to measure the functional size of management information systems, real-time software and multilayer systems. Since the software systems targeted by the COSMIC-FFP method are large-scale and inherently complex, feedback on this

complexity would permit their effective management throughout the software life cycle. The mechanism for obtaining feedback on software characteristics is the software measurement. This paper proposes a new measure for obtaining feedback on software functional complexity in the context of COSMIC-FFP. In order to provide objective, reliable and consistent feedback, software measurement requires a clear definition of the characteristic to be measured, namely, functional complexity.

Software complexity is an essential characteristic of the software process/product, and is a multifaceted notion the definition of which depends on the context [10] [13] [4] [22] [20]. Similar to the classification given by Whitmire [20], we view the complexity of a software system in different dimensions, namely computational, representational, structural and functional. Computational complexity quantifies the time and resources required to complete the process, and is covered by the study of algorithmic efficiency. Representational complexity considers the tradeoffs between graphical and textual notations for unambiguous representations of system model, system interactions and system behavior. Structural complexity is viewed in terms of coupling and cohesion, without considering the individual complexity of the components.

Functional complexity characterizes the dynamic performance of the system seen as a sequence of events required to fulfil a certain functionality of the system. For the purposes of functional complexity measurement in the COSMIC-FFP context, we consider the different types of events to which the system must respond in some time interval, as specified in the scenarios of usages of the software. Intuitively, the greater the variety and number of these events, the more complex the functionality. Functional complexity is quantified in terms of the entropy of an amount of information handled by the events in one usage of the system, and is aimed at complementing the COSMIC-FFP functional size method.

There is almost no cross-referencing in the software engineering literature between these two fields of knowledge, that is, functional size measurement and measurement of entropy, from the field of information theory. With the recent publication of work on the measurement of functional complexity based on entropy concepts, there is now an opportunity to investigate the candidate linkages, and candidate contributions, across the two fields. In this paper, section 2 presents the key elements of entropy measurement, and section 3 the key elements of COSMIC-FFP. Section 4 discusses the mapping of concepts across both measures, and, finally, section 5 identifies research directions including the use of both types of measures for scenario-based black-box testing and reliability estimation purposes.

2 Entropy Measurement

Information theory-based software measurement [18] [12] is used to quantify functional complexity in terms of an amount of information based on some abstraction of the interactions between software components [14]. However, what does information mean in this context? Shannon, the father of information theory, has stated that information causes change, and if it doesn't it isn't information [17]. In other words, we say that we have gained information when we know something now that we didn't know before, when 'what we know' has been changed. Under the assumption that the complexity of the product is associated with the information content of the software product, the quantification of the amount of information will be used to assess the functional complexity of the software system and the required quality improvement[2]. The average amount of information is quantified by the entropy of a set of events happening in one usage of the software. Below, the notion of entropy and its applicability to the functional complexity measurement are introduced.

2.1 Entropy

Entropy is one concept in information theory, and it was introduced by Shannon [17] as a quantitative measurement of the uncertainty associated with random phenomena. It is said that one phenomenon represents less uncertainty than a second one if we are more sure about the result of experimentation associated with the first phenomenon than we are about the result of experimentation associated with the second one. A random phenomenon must be described as a mathematical model, referred to as a probability space, in order to use mathematical reasoning to investigate questions about the phenomenon. For example, in throwing a die, the probability of the appearance of 1, 2, 3, 4, 5 or 6 is 1/6 for each. Much uncertainty is associated with throwing a die, since the expected outcome of the experiment is uncertain. Considering any set of n events and their probability distribution $\{p_1, \ldots, p_n\}$, the quantification of this uncertainty quantity is calculated using the following entropy formula:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i \quad \ldots\ldots (1)$$

2.2 Entropy Measurement in Software Engineering

In software engineering, Hamming has introduced entropy as a measure of the average information rate of a message or language [9]. A message means a string of symbols drawn from an alphabet of symbols $s_1, \ldots, s_q$. The field of information theory deals with the measure of the amount of information contained

in a message [5]. Information, in this context, is finding out something you did not already know; that is, when a symbol occurs that we were not expecting, we have more information than if a symbol we were expecting occurred. Rate, in this context, means the frequency of occurrence of each symbol [5].

Thus, the amount of information conveyed by a single symbol in a message is related to its probability of occurring: $I_i = \log_2 p_i$ …… (2).

Hartley (1928) was the first to propose the use of logarithms. The logarithm guarantees that the amount of information increases as the number of symbols increases.

Information is additive [5]; that is, the amount of information conveyed by two symbols is the sum of their individual information contents. It follows then that an entire alphabet of symbols s_1, …, s_q would on average provide the amount of information calculated in formula (1), with a bit as the unit of information per symbol. It can be shown that the maximum amount of information per symbol is provided by an alphabet with symbols that all occur with an equal probability. The average amount of information conveyed by each symbol in such an alphabet is $\log_2 q$ for an alphabet having q symbols, each with an equal probability of occurring. The minimum amount of information is conveyed by an alphabet in which one symbol occurs with a probability of one, and all others occur with a probability of zero. Such an alphabet is said to have a language entropy of zero.

2.3 Entropy-based functional complexity

In this paper, we propose a new method for quantifying functional complexity from a software behavior description. Our functional complexity measure characterizes the performance of the system as specified in the scenarios. Functional complexity is quantified in terms of the entropy of an amount of information based on an abstraction of the interactions among software components. Assuming that each message represents an event, therefore, entropy-based software measurement is used to quantify the complexity of interactions between the software and its environment and within the software (between software classes) in terms of the information content of the interactions, based on some abstraction of the interactions [17], [6], [10].

In OO development, the only vehicle for information interchange between the software and the environment, and within software modules (classes or packages), is the event, also called message. The environment is communicating with the OO software via external messages: input (to communicate a request from the environment for a service/usage) and output (to communicate the answer from the software to the environment). In order to fulfil the requested functionality, the objects are collaborating via message interchange. In the

Rational Unified Process (RUP) [11] for OO software development, the interactions (interchange of messages, that is, events, between the environment and the modules, and between the modules) are described as scenarios – written stories of a system's functionality related to one specific usage. In UML, the scenarios are modeled using interaction diagrams. Therefore, each scenario can be abstracted as a sequence of events.

For the purpose of measuring functional complexity, each scenario is mapped to a timed sequence of events, where each event is considered as a unit of information. The scenario described in Figure 1 has the sequence of events (messages) *e1.e2.e3.e2.e5*. To quantify the amount of information contained in this scenario, we can apply the entropy formula (1). We need, therefore, the set of events (that is {*e1, e2, e3, e5*}) and their probabilities {1/5, 2/5, 1/5, 1/5}.

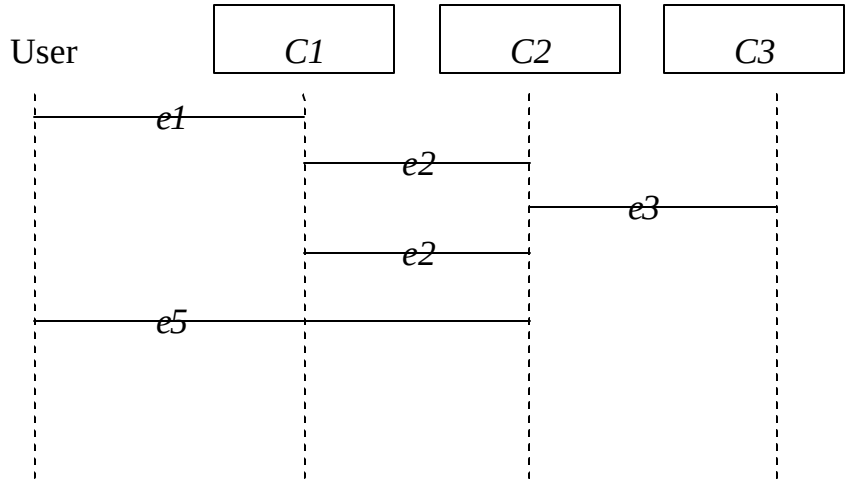

Figure 1: Scenario example

The functional complexity for system implementation [2] is defined as an amount of work output performed in a time slice by the system. The amount of work performed, in this context, means the quantity of information processed in that period of time, and the number of functions necessary to perform the work. The events represent the functions necessary to perform the work in one usage of the system, i.e. in one scenario.

The concepts of information theory [2] are applied to measure the amount of work output performed in a time slice by the system in terms of the amount of information in the event sequence. That measure is based on an empirical distribution of events within a sequence.

The probability of the i-th most frequently occurring event is equal to the percentage of the total number of event occurrences it contributes and is calculated as $p_i = f_i / \mathrm{NE}$, where f_i is the number of occurrences of the i-th event

and NE is the total number of events in the sequence. The classical entropy calculation quantifies the average amount of information contributed by each event. Therefore, the functional complexity in a time slice is defined in [2] as an average amount of information in the corresponding sequence of events and is computed as follows:

$$FC = -\sum_{i=1}^{n} (f_i / NE) \log_2 (f_i / NE) \ldots\ldots (3)$$

Functional complexity (FC) is the quantification for the amount of information interchanged in a given interaction (scenario). The functional complexity in a period of time with a higher average information content should, on the whole, be more complex than another with a lower average information content. That is, the FC measure is intended to order the usages of system in a time period in relation to their functional complexity.

3 COSMIC-FFP Measurement Method

3.1 COSMIC-FFP Overview

The functional size measurement method developed by the Common Software Measurement International Consortium (COSMIC) has now been adopted as an international standard (ISO 19761 [8]) and is referred to as the COSMIC-FFP method [1]. This measurement method has been designed to measure the functional size of management information systems, real-time software and multi-layer systems. Its design conforms to all ISO requirements (ISO 14143-1 [7]) for functional size measurement methods, and was developed to address some of the major weaknesses of the earlier methods – like FPA [3], the design of which dates back almost 30 when software was much smaller and much less varied. COSMIC-FFP focuses on the "user view" of functional requirements and is applicable throughout the development life cycle, right from the requirements phase to the implementation and maintenance phases.

In the measurement of software functional size using the COSMIC-FFP method, the software functional processes and their triggering events must be identified [1] [8].

In COSMIC-FFP, the unit of measurement is a data movement, which is a base functional component which moves one or more data attributes belonging to a single data group. Data movements can be of four types: Entry, Exit, Read or Write. The functional process is an elementary component of a set of user requirements triggered by one or more triggering events either directly or indirectly via an actor. The triggering event is an event occurring outside the

boundary of the measured software and initiates one or more functional processes. The sub processes of each functional process are sequences of events, and comprise at least two data movement types: an Entry plus at least either an Exit or a Write. An Entry moves a data group, which is a set of data attributes, from a user across the boundary into the functional process, while an Exit moves a data group from a functional process across the boundary to the user requiring it. A Write moves a data group lying inside the functional process to persistent storage, and a Read moves a data group from persistent storage to the functional process. See Figure [2] for an illustration of the generic flow of data attributes through software from a functional perspective.

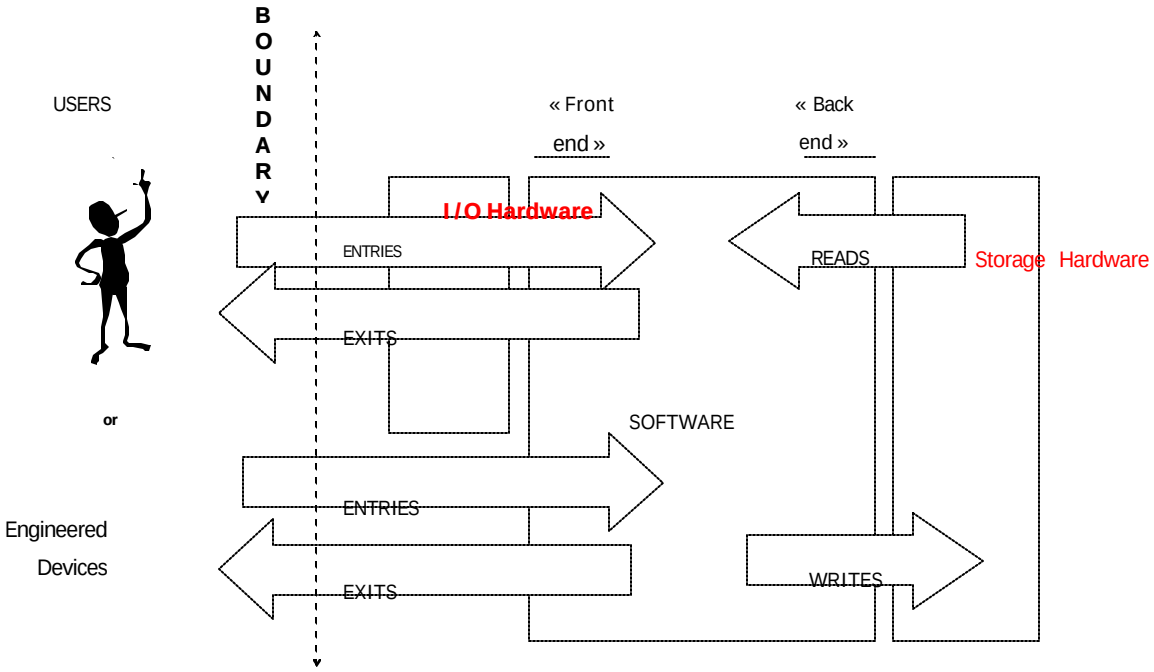

Figure 2: Generic flow of data attributes through software from a functional perspective

3.2 COSMIC-FFP and Functional Size

A general procedure for measuring software functional size with the COSMIC-FFP method is proposed, as in Figure 3. The measuring process is performed through five steps. First, the boundary of the software to be measured is identified by the measurer based on the requirements and the specifications of the interaction between the hardware and software. Secondly, the measurer identifies all possible functional processes, triggering events and data groups from the requirements. These are considered as candidate items at this stage. Thirdly, the candidate items (i.e. functional processes, triggering events and data groups) are

mapped into the COSMIC-FFP software context model (Figure 3) based on COSMIC-FFP rules. In this mapping, each functional process must be associated with a triggering event and to the data group(s) manipulated by it. This mapping also allows identifying layers. Fourthly, COSMIC-FFP subprocesses (i.e. data movements of the following types: Entry, Exit, Read and Write) will be identified within each functional process. The COSMIC-FFP measurement function will be applied to the identified sub processes to determine their respective COSMIC-FFP Cfsu size measure. Finally, the measurer will compute an aggregate of the measurement results to obtain the total functional size of the software being measured.

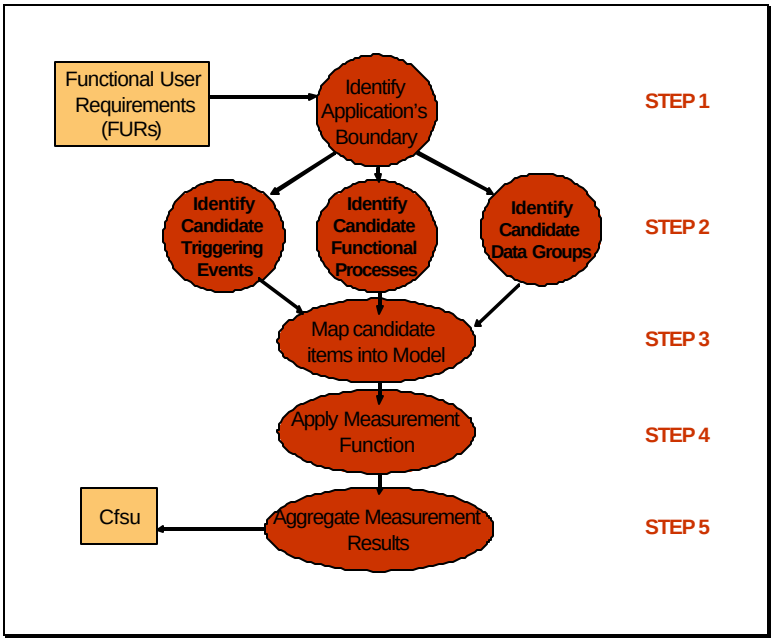

Figure 3: General procedure for measuring software size with the COSMIC-FFP method – ISO 19761

4 Analysis of Similarities and Differences across Measures

The method used to analyze the compatibility between Functional Complexity-based Entropy and COSMIC-FFP consists of comparing the generic software models, their software model components and their software measurement processes [16]. For such a comparison, it is necessary to identify the concepts behind the terms used in both measures: in fields that are not yet mature and where the terminology is not fully standardized, such as software engineering, different terms will sometimes refer to the same concepts, and at others, the same term will refer to different concepts.

4.1 Models of Software Comparison

In their generic view of software from a functional perspective, the two measures share a similar generic modeling of how to recognize the functionality of software.

Information theory-based software measurement quantifies functional complexity in terms of an amount of information based on some abstraction of the interactions between software modules, and more specifically the complexity of interactions between the software and its environment and within the software in terms of the information content of the interactions.

COSMIC-FFP shares a similar generic model of software functionality, which is defined as the interactions between the software and its environment and within the software, as illustrated in Figure 2. In COSMIC-FFP, the environment is represented by the users interacting with the software, such software users being either humans, engineering devices or other software applications. Within the software, the interactions deal with the data read or send them to persistent data storage.

4.2 Models of Software Component Comparison

The second step consists in identifying and comparing the software model components used by each method required for the instantiation of the generic software model of functionality. In the RUP context, the functional processes used in COSMIC-FFP can represent the set of scenarios for the software. For example, in the Hotel Reservation System [RURA1 & RURA2 in 23], the user can create a reservation. This process of allowing the user to add a new reservation is considered as a functional process, and is triggered by selecting the user for this option. Similarly, creating the reservation is a scenario containing a sequence of events between the user and the system, and this scenario contains a sequence of events within the system. Therefore, for each functional process, its sub processes and its triggering events are sequences of events (events).

Within the same RUP context for functional complexity measurement, the entropy formula can likewise be calculated on one process (scenario), as described above.

See Figure 4 for the functional process for creating a reservation that can be the scenario of a set of events interchanged between software components. The set of alphabets (events) comprises {1: select “create”, 2: display, 3: type required information, 4: store information, 5: display error message if it occurs} and their probabilities: {1/5, 1/5, 1/5, 1/5, 1/5}.

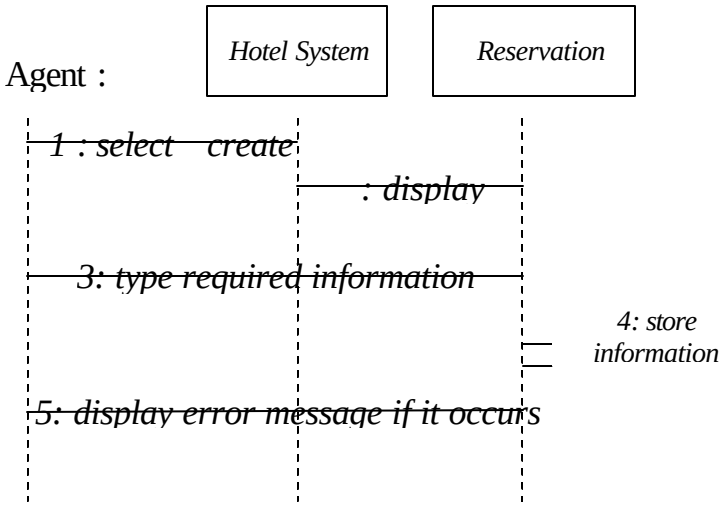

Figure 4: Create reservation sequence diagram

It is to be noted next that the measurement unit defined in COSMIC-FFP is a data movement, that is, a base functional component which moves one or more data attributes belonging to a single data group. It can now be observed that an event as the unit of Functional Complexity based-Entropy has the same meaning as the data movement, the unit of COSMIC-FFP.

A summary of the terms used in both COSMIC-FFP and Functional Complexity based-Entropy having similar meanings is presented in Table 1. This table shows the same conceptual level for both COSMIC-FFP and Functional Complexity-based Entropy; however, the terms used in the data movements of COSMIC-FFP and in the interactions of Functional Complexity-based Entropy have different labels. For example, in COSMIC-FFP, data movements are classified into four categories. The term corresponding to the data movements and its categories that is used in Functional Complexity-based Entropy is the event, but without classification. In addition, other terms used in both measurement methods, such as those interacting with the software, the software boundary and the set of user requirements, have the same meaning even though some have different labels. For example, software users are actors while at the same time they are interacting with the software. As a result of Table 1, the models of software of these measures are compatible. The question mark that appears in the table for Functional Complexity-based Entropy means that it is not yet known what the measurement unit is and what its symbol is. This can be analyzed more extensively in future papers dealing with scale type issue.

Concepts	COSMIC-FFP (Data Movement) terms	Functional Complexity-based Entropy (Interactions) terms
Humans or things interacting with the software	Software users	Actors
Between the environment and the software	Software boundary	Software boundary
Set of User Requirements	Functional Process	Scenario (Sequence of Events)
Data which are part of the interaction	Data groups	Set of data attributes
External Input (From Environment)	Triggering event	Event
External Input (From Environment)	Entry data movement	Event
Output (to the environment)	Exit data movement	Event
Entity being taken into account in the measurement	Data movement	Event
Measurement Unit	1 data movement	?
Measurement unit symbol	Cfsu	?

Table 1: Similarities of concepts between COSMIC-FFP & Functional Complexity-based Entropy

4.3 Software Measurement Process Comparison

Even though each type of measure takes into account similar concepts (with different terms) for the model of the software to be measured, each type of

measure, when its measurement processes are compared, defines different measurement functions (e.g. formula) to combine the information into a 'measure' for purposes which are obviously different. For example, the COSMIC-FFP formula is used to measure the functional size of software, that is, the amount of functionality of the software through the addition of data movements. COSMIC-FFP recognizes only data movement type sub processes, and it contains an approximation assumption that each data movement is associated with an average amount of data transformation.

By contrast, Functional Complexity measurement based on Entropy is used to measure the amount of information in the interactions between the software and the environment, and within the software modules; this formula associates the functional complexity of software to the frequency of function occurrences through a logarithmic function of probability distribution of the events (see formula 3). It can be easily observed that Functional Complexity-based Entropy extracts more information than does COSMIC-FFP about the events, their frequencies and their probabilities.

For illustrative purposes, let us consider two events: e1 and e2. In Figure 5, if either e1or e2 is certain (p1 =1 or p2 = 1), then FC is zero. The same thing will happen when p = 0 for either event. However, when p1 = 0.5 or p2= 0.5, both events are just as probable and FC is 1, which is the maximum. Therefore, FC is maximum if all probabilities are equal, and it is minimum if one event has a probability equal to 1. However, in COSMIC-FFP, when two different data movement types are required to perform the functional process, then the number of Cfsu is 2. The Cfsu number is also the same when one data movement is required twice and the same data group is accessed in order to execute the functional process. It is to be noted that two functional processes may end up with the same functional size regardless of the type of data movement.

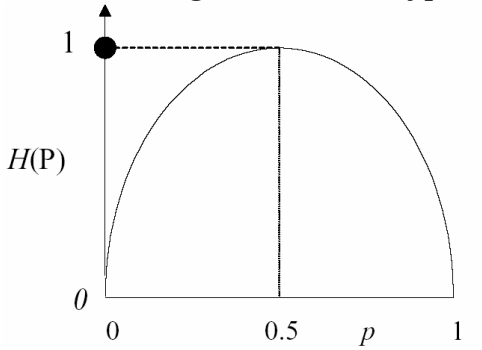

Figure 5: Functional Complexity-based Entropy

5 Discussion and Next Steps

COSMIC-FFP and Functional Complexity-based Entropy are both measures of software functionality, but their purposes are different, one being to measure size, and the other the complexity of one usage (functionality) of the software. This paper has explored the similarities and differences between the two measures. Even though both measures use significantly different terminology, a comparative analysis of the concepts behind these distinct terminologies reveals important similarities in how they view and represent software from a functional perspective. Findings include, in particular, significant similarities in the way both measures view software, and in their generic model of software functionality when they consider the interactions of the software with its environment, and interactions within the software itself. There are also important similarities, but not full equivalence, within the software components they take into account in their respective measurement processes. Finally, each measure obviously has different measurement functions (that is, the formula for transforming the information into numbers): while COSMIC-FFP is strictly an additive aggregation of data movements, the functional complexity measure is much more complex and is based on the concept of entropy, itself derived from information theory.

Further research is required to investigate in greater detail the similarities of concepts within the elements handled within each of these measures. Such an investigation will be useful for cross-fertilizing the two types of measurement method.

On the one hand, COSMIC-FFP measurement concepts and procedures are well documented and, through the method's international acceptance as an ISO standard, it has achieved international recognition as a measurement method supported by the international community specializing in measurement of any kind. However, the field of software functional size itself has very limited depth in terms of research and theoretical support to draw upon. Its use is therefore fairly limited, extending only to productivity studies and estimation, with almost no reported use in quality and reliability analysis.

On the other hand, entropy has been used extensively in many fields, including entropy-based measures which have been used, for instance, for performance [19] and reliability estimation [15], both with very strong theoretical and empirical support. For instance, the reliability of the software can be estimated following some input reliability model on a certain input domain.

Of particular interest would be to investigate the use of COSMIC-FFP in both testing effort and reliability estimation, the COSMIC-FFP method being applicable at the early development phase where we only know about the specifications of the software: for instance, it would be of interest to use the COSMIC-FFP measurement details in the scenario-based black-box testing

strategy [21], [11], and to investigate the applicability of entropy measurement in COSMIC-FFP for reliability estimation purposes.

Another research topic which needs to be looked at and analyzed is the scale types of these two measurement methods, in order to add new depth to other areas in the software measurement field.

References

1. A. Abran, J.-M. Desharnais, S. Oligny, D. St-Pierre and Symons, C. COSMIC-FFP - Manuel de mesures - Version 2.2, Université du Québec à Montréal, Montréal, 2001.

2. Alagar, V.S., Ormandjieva, O. and Zheng, M., Managing Complexity in Real-Time Reactive Systems. in *the Sixth IEEE International Conference on Engineering of Complex Computer Systems*, (Tokyo, Japan, 2000).

3. Albrecht, A.J. and Gaffney, J.E. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Trans. Software Eng.*

4. B. Henderson-Sellers *Object Oriented Metrics: Measures of Complexity*. Prentice-Hall, New Jersey, 1996.

5. Hamming, R. *Coding and Information Theory.* NJ: Prentice-Hall, Englewood Cliffs, 1980.

6. Harrison, W. An Entropy-Based Measure of Software Complexity. *IEEE Transactions on Software Engineering*, *18* (11).

7. ISO 14143-1. Functional size measurement - Definitions of concepts. in, International Organization for Standardization - ISO, Geneva, 1988.

8. ISO/IEC 19761. Software Engineering - COSMIC-FFP - A functional size measurement method. in, International Organization for Standardization - ISO, Geneva, 2003.

9. J. F. Peters and Pedrycz;, W. *"Software Measures" in Software Engineering: An Engineering Approach*. J. WILEY, 2000.

10. John Stephen Davis and Leblanc;, R.J. A Study of the Applicability of Complexity Measures. *IEEE Transactions on Software Engineering*, *14* (9). 1366-1372.

11. Larman, C. *UML and Patterns: An introduction to Object Oriented Analysis and Design and the Unified Process*. Prentice Hall, 2002.

12. N. F. G. Martin and England, J.W. *Mathematical Theory of Entropy*. Addison-Wesley Pub. Co., 1981.

13. Norman E. Fenton and Pfleeger;, S.L. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing, 1998.

14. O. Ormandjieva. Deriving New Measurement for Real Time Reactive Systems *Department of Computer Science & Software Engineering*, Concordia University, Montreal, 2002.

15. S. N. Weiss and Weyuker;, E.J. An Extended Domain-Based Model of Software Reliability. *IEEE Trans. Software Eng.*, *14* (10). 1512-1524.

16. Serge Oligny and Abran;, A., On The Compatibility Between Full Function Points And IFPUG Function Points. in *10th European Software Control and Metrics Conference (ESCOM SCOPE 99)*, (Herstmonceux Castle, England, 1999), 23.

17. Shannon, Claude E., Weaver and Warren *The Mathematical Theory of Communication*. the University of Illinois Press, Urbana, Chicago, 1969.

18. Taghi M. Khoshgoftaar and Allen;, E.B. Applications of information theory to software engineering measurement. *Software Quality Journal*, *3* (2). 79-103.

19. V.S.Alagar, O.Ormadjieva and J.Shen;, Scenario-Based Performance Modeling and Validation in Real-Time Reactive Systems. in *the First Software Measurement European Forum*, (Rome, Italy, 2004).

20. Whitmire, S.A. *Object Oriented Design Measurement*. John Wiley & Sons, 1997.

21. X. Bai, W. T. Tsai, K. Feng and Yu;, L., Scenario-based Modeling and Its Applications to Object Oriented Analysis Design and Testing. in *IEEE Words*, (2002).

22. Zuse, H. *Software Complexity Measures and Methods.* Walter de Gruyter, Berlin

New York, 1991.