FROM MEASUREMENT OF SOFTWARE FUNCTIONAL SIZE TO MEASUREMENT OF COMPLEXITY

D. Tran-Cao[1] G. Lévesque[2] A. Abran[3]
de.trancao@lrgl.uqam.ca levesque.ghislain@uqam.ca abran.alain@uqam.ca
Software Engineering Management Research Laboratory
University of Quebec at Montreal (UQAM)

Abstract

The current measurement methods of software functional size really measure in particular the data movements associated with each process. The data manipulation or algorithmic complexity is not taken into account adequately in none of the most popular methods. In this paper, we recall some well-known methods measuring problem complexity in data manipulation and their interest for a new definition of complexity. This definition has the merit of dissociating the concept of effort, which is always associated with complexity, and refers to the characteristics and intrinsic properties of software. Our objective is to propose some simple and practical approaches to measure some of these characteristics which we consider particularly relevant and to incorporate them in a measurement method of the functional size.

Résumé

Les méthodes actuelles de mesure de la taille fonctionnelle des logiciels ne mesurent réellement que les mouvements de données associés à chaque processus. La manipulation de données ou la complexité algorithmique n'est pas prise en compte convenablement dans aucune des méthodes les plus populaires. Dans cet article, nous rappelons quelques mesures bien connues de la complexité du problème dans la manipulation de données et leur intérêt pour aboutir à une nouvelle définition de la complexité. Cette définition tend à dissocier la notion d'effort, qui a toujours été associée à la complexité, et se réfère aux caractéristiques et aux propriétés intrinsèques du logiciel. Notre objectif est de proposer quelques approches simples et faciles d'application pour mesurer quelques-unes de ces caractéristiques que nous considérons particulièrement pertinentes et pour les incorporer à une méthode de mesure de la taille fonctionnelle.

1. Introduction

Software size in function points is a measure of the product size. It can be used with the goal of evaluating and predicting some aspects of the production process like the effort, the cost and the productivity of software development. There are two main approaches to measuring software size: a posterior approach as LOC [8] and a priori approach as the methods based on software functionality [1, 2, 3, 4, 16, 20, 23, 26]. LOC is the simplest and the earliest method used to measure software size, it is very useful, but it also has been the object of many criticisms [4, 10] on how to define a line of code and how to deal with different types of programming languages. The methods of a priori approach get more and more attention of the software measurement community because they are

[1] Cognitive and Computer Sciences PhD student, University of Quebec at Montreal
[2] Professor of Software Engineering, University of Quebec at Montreal
[3] Professor of Software Engineering, École de Technologie Supérieure

independent of programming languages and they allow early estimation size of the end product which provides, when calibrated to the software environment, a significant index to evaluate the effort of development and the cost of a software product [15]. In fact, Albrecht's Function Point Analysis (FPA) is widely used to measure the software size of Management Information System (MIS). However, the FPA is criticized for not taking into account the complexity in an objective way [23]. Therefore, the FPA is not likely applicable to all types of software [2].

Some attempts were made to adapt FPA to software types that are complex in data manipulation as real time software in order to objectively estimate the software complexity [1, 2, 16, 20, 23, 26]. Among these, Mark-II Function Point Method [23] proposed an approach to calibrate objectively the complexity, but it requires history data for calibrating. So, it may be difficult to apply it to an application type with no history data. Some other extensions deal with the special characteristics of software which express the algorithmic difficulties or the complexity in the process of transforming (or manipulating) data from inputs to produce expected output data. We can recall some well-know methods here. Feature Points [16] weights the algorithm complexity based on its calculation steps and the data elements that need to be manipulated. But there is no guidance to identify an algorithm at an adequate level of abstraction [2]. 3D Function Points [26] proposed another approach to take into account the complexity of data manipulation. This method measures software size in three dimensions: data, function and control (dynamic behaviours). Data dimension is measured by the number of inputs, outputs, inquiries, internal data structures and external logical files. This dimension is measured similarly as the unadjusted function points of FPA. Function dimension is measured by the number of transformations which is the sum of the number of processing steps and the number of semantic statements. And the third dimension is measured by the number of transitions in the state model. The 3D Function Points introduced some indices very interesting to quantify the data manipulation complexity, but it is criticized for lacking of detailed definitions which allows the identification of these indices [2].

As shown above, these extension methods do not provide enough details to be applied successfully and with a satisfactory level of confidence. In general, we do not have a generic software model at the analysis phase. So it may be very difficult to insure that the measurement method is applied to the software requirements document with the same level of abstraction. It is then difficult to use and interpret the result of measure. In the family of function point methods, COSMIC-FFP [1, 22] proposes a generic software model at the analysis phase based on the software requirements specifications. According to this model, software is considered as a set of functional processes that, in turn, is implemented by a set of functional sub-processes of two types: data movement and data manipulation. Then the software size is measured by counting the numbers of data movements. This method does not address the complexity of data manipulations.

In this paper, we propose a research direction to deal with the software complexity of data manipulation that transforms the inputs to the outputs of a functional process. Then this complexity measure could be integrated to the FFP method to provide a more complete index to predict the software development effort and costs. Our objective is to

propose some simple and easy measures to quantify some of the intrinsic characteristics of software that we consider particularly relevant for data manipulation complexity.

2. Software Measurement and software complexity

Software measurement method is a rule for assigning a number or identifier to software in order to characterize the software [9]. Our research focus on proposing a method to evaluate software complexity, which in turn is used later to assess the effort or the cost to develop software. It is also worth to distinguish between the characteristics which we want to measure and the measure by which we evaluate and understand that characteristic. For example, the complexity of code is a characteristic used to describe a piece of code, a design or even a problem specification. This characteristic may be measured by different measures, for instances, Lines of Code [8], Cyclomatic number [18] or Difficulty measure [12]. Therefore, measure of complexity (a measure) is not the complexity (a concept) in itself. Confusingly, in the literature, the term "complexity" is often used to indicate the measure which is used to assess the concept of complexity. Zuse say that *the term complexity measure is a misnomer. The true meaning of the term software complexity is the difficulty to maintain, change and understand software. It deals with the psychological complexity of program* [25]. There hasn't been a consensus on how to define software complexity, but we can find some interesting definitions of this term in the literature.

Basili [6] defines that *software complexity as a measure of resources expended by a system [human or other] while interacting with a piece of software. If the interacting system is a programmer, then complexity is defined by the difficulty of performing tasks such as coding, debugging, testing or modifying the software.* The definition does not distinguish clearly between the complexity (a concept) and measure of complexity. However, the concept of complexity can be interpreted as the difficulty of performing some tasks on the software. Moreover, the difficulty comes from the intrinsic characteristics of software. Sellers [21] propose another definition in which software complexity is a concept that refers to the software characteristics that bring the difficulty to the person who perform a task on it. He defines *the cognitive complexity of software refers to those characteristics of software that affect the level of resources used by a person performing a given task.*

Although the concept of complexity is always interpreted via the difficulty or the effort of performing a task, a measure of complexity is not a measure of effort. The complexity measure may be an index or a parameter of the effort to do something but it is not a measure of effort. Intuitively, we believe that two similar piece of software are at the same level of difficulty or complexity, but in fact, the effort to build them in two different organizations may be different. Therefore, effort does not coincide with the complexity. In our research, software complexity refers to the intrinsic characteristics of software that cause the difficulty of a given task, then a software complexity measure is to assign a number to these characteristics or evaluate these characteristics in term of numbers.,while effort indicates the work-hours (man-month) paid to do that task.

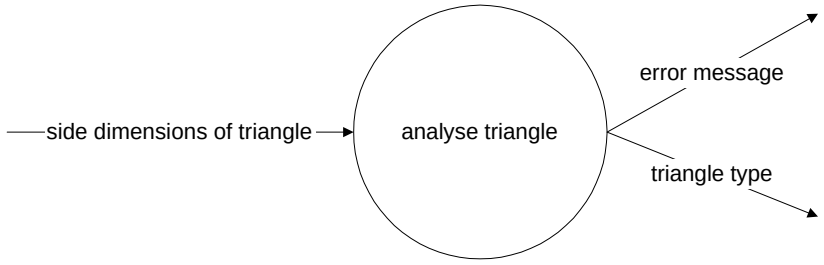
In addition, software complexity differs from the software size. Fenton [10] suggests that software size is a function of three variables: length, functionality and problem complexity. He also argues that there are many types of complexity, such as problem complexity, solution complexity, algorithmic complexity, structural complexity and cognitive complexity. This paper focuses on the problem complexity, which is based on the specification of data manipulations to transform the inputs into expected outputs.

3. Software specification and functional measurement

A software requirement specification is an abstraction of an implemented (or to be implemented) software system. It is the earliest product of the software development process. Therefore, a product measure cannot be applied before the software specifications are already available. In fact, the FPA and its extensions measure the software size based on software functionalities, which are derived from the software specifications. The challenge for measures with the functional approach is to have good specifications, which means, a complete specification at an adequate level of abstraction to enable accurate software size measurement [11]. No method defines such level of abstraction, so users (who use measurement method) are always confused when they compare the size of two software which are derived from two specifications at two different levels of abstraction. Some authors try to overcome this challenge by using a formal specification like using algebraic specification [11], but we found that this approach is not adequate to evaluate the complexity of data manipulation that occurs in the functional process, as the difficulty to transform inputs to outputs. Moreover, the specifications are not often made in algebraic formal, the fact is that, they are often produced in natural language or in Program Design Language (PDL), which is a combination of natural language with a syntax of a structured programming language.

In this research, we develop an approach for a complexity measurement method based on the software specifications at the end of the analysis phase. The specifications describe processes whose content can include a narrative text, a program design language of process algorithm [19, p.330]. To illustrate, consider the analyze-triangle problem whose first specification is a narrative text as described in figure 1.

This specification is detailed later by adding more details to the algorithm to transform input data to output data. In fact, at the end of the analysis phase, we can get a more detailed specification in PDL which is similar to the pseudo-code in figure 2.



The analyze triangle process accepts values A, B and C that represent the side dimensions of triangle. The process tests the dimension values to determine whether all values are positive. If a negative value is encountered, an error message is produced. The process evaluates valid input data to determine whether the dimensions define a valid triangle and if so, what type of triangle- equilateral, isosceles or scalene – is implied by the dimensions. The type is output.

Figure 1: analyze-triangle specification in narrative text[4]

```
PROCEDURE analyse-triangle
        READ side dimensions
        IF any dimension is negative THEN produce an error message
        ELSE IF the largest dimensions is less than the sum of the others THEN
                Determine number of equal sides
                IF three sides are equal THEN type is equilateral
                ELSE IF two sides are equal THEN type is isosceles
                ELSE {no sides are equal} type is scalene
        ELSE triangle does not exit
END PROCEDURE
```

Figure 2: analyze-triangle specification in using PDL

The specification describes only **what** is desired rather than **how** it is to be realized. The specification must be operational, which means, the specification must be completed and formal enough and so that it can be used to determine if a proposed implementation satisfies the specification for arbitrarily chosen test case. It also can be used as a guide for acceptance test to verify if the right thing has been built. For these purposes, we propose that a functional specification of a process should describe not only the input-output data of the process but also the *conditions on input* to produce *different types of expected outputs*. In this view, the specification of a functional process can be seen as rules to transform inputs to outputs. These rules can be expressed in PDL as in the second version of the specification of analyze-triangle (figure 2), or in a decision tree as in figure 3.

To measure the complexity of data manipulation to transform inputs to outputs, first we reuse a generic model proposed by the COSMIC team [1, 22], in which software is a set of functional processes. COSMIC team proposes that each functional process is an

[4] this part is copied from page 331 of [19]

ordered set of sub-processes of two types: data movement type and data manipulation type. In this paper, we propose an approach for a new model to study the complexity of data manipulations. We propose that, *at the end of the analysis phase, the specifications must describe all output types desired and the conditions on inputs to produce those outputs*. Therefore, we can consider that each functional process represents the rules applied on inputs to produce the outputs desired. We also want to represent each functional process or rules as a decision tree, so that the complexity of data manipulations is measured with this tree. After that, the complexity of software under study will be the sum of the complexity of all functional processes.

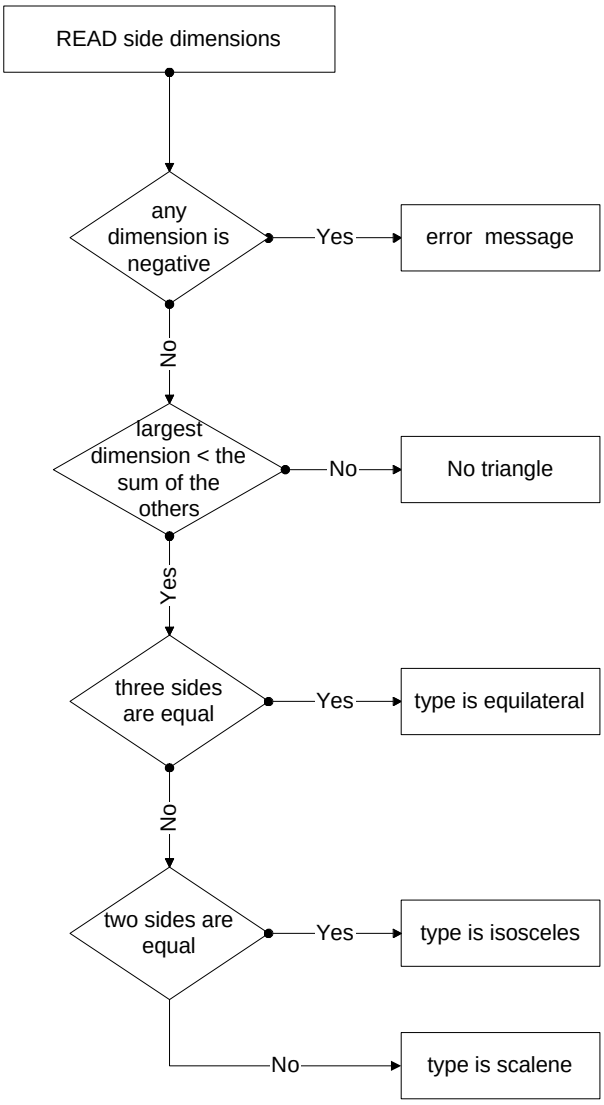


Figure 3: decision tree of analyse-triangle specification

4. Simple indices for complexity measurement

As it has been said in the previous sections, our research goal is to propose a new measurement method which deals with the complexity in data manipulations that occur in functional processes of the software. Our research aims to measure some indices of this complexity from the decision tree. These indices can then be used as parameters to assess the size, the effort or the cost of software. Here we use the decision tree, which describes the conditions on inputs to produce the different types of outputs, as the level off abstraction of a functional process. We distinguish two different types of node on the decision tree. The statement nodes (rectangle node in the figure 3) represent a task that must be done in the process and that can be identified from the user side like the inputs and outputs of the process. The decision nodes (diamond nodes in the figure 3) represent the Boolean conditions causing a branch in the decision tree. A decision node illustrates the conditions on input data, which are the predicate statements in the specifications, to produce an output type.

Some simple indices may be derived from the decision tree:

- The number of inputs and outputs, which can be interpreted as the amount of data that must be processed in the process. This index is used in many function point methods such as FPA, COSMIC-FFP, Mark-II, 3D Function Points. It is always interpreted as a part of the functional size of the process.

- The number of decision nodes, which is known as the "complexity" in the structure of the process. In fact, the Cyclomatic Number of McCabe [18], which is widely used as the complexity metric, is the number of decision nodes plus one. This number is also the number of linearly paths (or branches) in the decision tree. This number is sometimes used as an index for the difficulty of the coverage test of the process [17]. Note that, a decision node may represent a complex logical predicate that is a conjunction or a disjunction of simple predicates. Therefore, we believe that the counting of predicates of all decision nodes may be a potential index of structural complexity. In fact, 3D function Points suggested a counting of all logical predicates while measuring the functional size.

- The depth of the tree, which is interpreted as the maximal number of steps in data processing from input to output. In the literature, 3D Function Points proposed the idea of measuring the number of steps of internal operations required to transform input data to output data. This number is the counting of processing steps and semantic statements, but no detailed definition is provided to identify processing steps or semantic statements in the process. Here, we propose that the depth of the decision tree is interpreted as the (maximal) number of processing steps in the process and it is used as an index of the data manipulation complexity. Intuitively, the sum of length of all paths on the tree may be considered as a candidate for the number of processing steps. But we believed that the depth (the length of the longest path) is more adequate to express the complexity. The sum of length of all paths may well conform to the "length", not complexity of the process. The counting of some simple index of the tree in the figure 3 is shown in the table

below. However which one is more adapted to represent the complexity will be determined in empirical research further.

Number of inputs and outputs	6
Number of decision nodes	4
Sum of predicates of all decision nodes[5]	3 + 1 + 1 + 3 = 8
The depth of decision tree	6
Sum of length of all paths	2+3+4+5+6 = 20

Table: counting of some simple index of the tree in the figure 3

In future research, we will analyse the independence of these indices to find out some independent indices on which we establish a new measure for data manipulation complexity. The first hypothesis is that *data manipulation complexity depend on the numbers of processing steps in the process, that can be quantified by the depth or the sum of length of all paths, and the structural complexity of the process, that can be quantified by the number of decision nodes or the sum of predicates*. The number of processing steps can be interpreted as the index for the mental effort to deal with processing steps from input to output and the structural complexity can be know as the index for the mental difficulty to deal with the structure of the process or the organization of processing steps in the process.

After we have determined the independent indices for data manipulation complexity, for example the depth of decision tree and the number of decision nodes, we will establish experimentally a complexity measure or an effort estimation model or a cost model using these factors. Then external validation will be made by analysing the correlation between the measures from these new models and history data.

5. Validation

In our research, we use an approach for developing structural measures [5]. This approach shows that, first, we have to define an attribute of software based on an abstraction of software document. In this research, the attribute is data manipulation complexity whose definition is based on the decision tree, which is considered as the abstraction of a functional process. Then, we have to define an order of the abstraction and a mapping to the real number that preserves this order. Here we want to define this order and mapping this order to real number by calculating some simple indices of the decision tree as mentioned above. That means, the new method will be experimentally established based on these simple indices. Of course, the new measurement method need to be validated to show that it measures what we want to measure, or at least to show that it is useful to assess an attribute of software.

[5] the predicate in the first decision node is : a<0 or b<0 or c<0; the predicate in the last decision node is: a=b or a=c or b=c.

Baker [5] states that *validation of a software measure is the process of ensuring the measure is a proper characterization of the claimed attribute.* This definition refers to internal validation, that is, it requires consideration of the underlying models used to capture the object and attributes. Internal validation aims to show that we measure the right thing in a right way. But internal validation of a software measure is often difficult because both object and attribute which we want to capture are not clearly defined.

Another type of validation is external validation. *External validation of a measure m is the process of establishing a consistent relationship between m and some available empirical measure data purporting to measure some useful attribute* [5]. This kind of validation does not prove that we measure what we want to measure but it shows that our measure is useful to assess some external attributes. In fact, this type of validation is often used to validate software measure, for example. The Cyclomatic number and Difficulty measure of Halstead are analysed in correlation with the number of bugs in the code. The software size (measured by FPA, Mark-II, FFP,…) is often analysed in correlation with the effort. Note that, FPA does not measure the software development effort but it is used as an index of development effort in an effort estimation model. Then the value of effort (estimated by FPA effort model) is analysed in correlation with the real effort, if the correlation is well, we believe that FPA is useful to predict software development effort. This approach is often criticized because of lacking causal relation between the measure validated with empirical data, but it is the main way to validate a new measure for now.

In our research we will use this approach to validate our measure, that means, first we propose some simple indices for data manipulation complexity like the depth and the number of decision nodes in the decision tree representing the functional process. Then, we establish an effort prediction model based on these indices. After that, the prediction effort, which is calculated by our model, will be analysed in correlation with the actual effort (real history data) or with the effort predicted by another model (COCOMO 2.0, for example). If the correlation is high or better than what is obtained with the software size measured by FPA or FFP, we believe that our measure is useful.

6. Conclusion

So far, we show that the current software size methods don’t take into account adequately the complexity of data manipulations. Some methods (such as 3D Function Points and Feature Points) want to capture this complexity but they lack detailed definitions of attributes measured as well as the abstraction level of the specifications which allow to capture correctly and consistently these attributes.

This paper defines an abstraction level of requirement specifications at which the specifications are detailed enough to express all types of expected outputs of a functional process as well as the necessary conditions on inputs to get expected outputs. At this level, each functional process may be represented as rules to manipulate the inputs to produce the outputs. These rules can be represented in natural language or PDL and it is considered as a decision tree to produce all types of outputs expected.

Based on this decision tree, we identify some simple indices that are considered as potential candidate indices for data manipulation complexity. These indices are chosen on the preliminary hypothesis that data manipulation complexity depends on the number of processing steps to transform the inputs into expected outputs. It also depends on the structure of the process that is represented by the logical predicates.

The main goal of this paper was to propose a research direction to establish a new measure for data manipulation, including a proposition for some indices for data manipulation complexity, the ways to establish a new measure and validate it. We hope that a new measure of complexity based on such simple indices may be easy to use and may satisfy the consistency criteria.

References

[1] Abran, A.; Desharnais, J.-M.; Oligny, S.; St-Pierre, D.; Symons, C., *COSMIC FFP - Measurement Manual, Version 2,1*, Montreal, May, 2001.

[2] Abran A., Desharnais J.M., Maya M., St-Pierre D., Bourque P., *Design of a functional size measurement for Real-Time Software*, UQAM, Research report No 13-23, 1998.

[3] Albrecht A., Gaffney J., *Software Function, Sources Lines of Code, and Development Effort Prediction: A Software Science Validation*, IEEE Transactions on Software Engineering, Vol. SE-9, No. 6, November 1983.

[4] Albrecht A., *Measuring Application Development Productivity*, Present at IBM Applications Development Symposium, 1979.

[5] Baker A.L., Bieman J.M, Fenton N. Gustafson D.A., Whitty R., *A Philosophy for Software Measurement*, System Software, 1990.

[6] Basili V.R., *Qualitative software complexity models: A summary in tutorial on Models and methods for software Management and Engineering*. IEEE computer Society Press, Los Alamitos, California, 1980.

[7] Côté V., ST-Denis R., 1992, *Bridging The Gap Between Case Tools and Project Management Through a Decision Support System Based on Metrics*, iEE Compsac, 300-309.

[8] Conte, S.D., Dunsmore, H.E., and Shen, V.Y., *Software Engineering Metrics and Models*, Benjamin/Cummings, Menlo Park, CA, 1986.

[9] Dunsmore H.E., *Software Metric: An Overview of An Evolving Methodology*, Information Processing & Management, Vol. 20, No 1-2, pp. 183-192, 1984.

[10] Fenton N.E., Pfleeger S.L., *Software Metrics: A Rigorous and Practical Approach*, second edition, 1997.

[11] Hastings T.E., Sajeev A.S.M., *A Vector-Based Approach to Software Size Measurement and Effort Estimation*, IEEE Transactions on Software Engineering, Vol. 27, No. 4, April 2001.

[12] Halstead, M.H., *Element of Software Science*, New York, Elsevier North-Holland, 1977. 0

[13] Henry S.M., Kafura D.G., *Software Structure Metric Based on Information Flow*, IEEE Transactions on Software Engineering, 7, no. 5, September 1981, pp.545-522.

[14] Humphrey W.S., *A discipline for Software Engineering*, Reading Mass., Addison-Wesley, 1995.

[15] Jeffery R.D., Low C.G.; *Function Points in the Estimation and Evaluation of Software Process*, IEEE, Vol. 16, No 1, January, 1990.

[16] Jones C., *Applied software measurement – Assuring productivity and quality***,** New York, McGraw-Hill Inc, 1991.

[17] McCabe T.J., Butler C.W., *Design complexity measurement and testing*, Communications of the ACM, Vol. 32, No. 12, 1989.

[18] McCabe T.J., *A Complexity Measure*. IEEE Transactions of Software Engineering, Volume SE-2, no. 4, pp.308-320, December 1976.

[19] Pressman R.S., *Software engineering: a practitioner's Approach*, fourth edition, The NcGraw-Hill, 1997.

[20] Reifer D.J., *Assert-R: A Function Point sizing Tool for scientific and Real-Time Systems*, Journal of systems software, vol. 11, pp. 159-171, 1990.

[21] Sellers H., *Object-Oriented Metrics – Measures of Complexity*, Prentice Hall, New Jersey, 1996.

[22] Symons C., Grant Rule P., *One size fits all 'COSMIC' – Aims, Design principles and Progress*, Project Control for Software Quality, ISBN 90-423-0075-2, 1999.

[23] Symons C., *Function Point Analysis: Difficulties and Improvements*, IEEE Transactions on Software Engineering, Vol. 14, No. 1, January, 1988.

[24] Zuse H., *A Framework for Software Measurement*, Walter de Gruyter, Germany, Berlin, 1997.

[25] Zuse H., *Software Complexity Measures and Methods*, Walter de Gruyter, Germany, Berlin, 1991.

[26] Whitmire S.A., *3D Function Points: Scientific and Real-Time Extensions to Function Points*, presented at Pacific Northwest Software Quality Conference, 1992.