COMPLEXITY-BASED METRICS FOR THE EVALUATION OF THE PROGRAM ORGANIZATION

Rui Gustavo Crespo

Instituto Superior Técnico
Av. Rovisco Pais,
1049-001Lisboa, Portugal
Tel:+351-218418398
R.G.Crespo@digitais .ist.utl.pt

Ana Isabel Cardoso

Engª . Sistemas e Computadores
Universidade da Madeira
9000-390 Funchal, Portugal
Tel:+351-291740320
aic@dragoeiro.uma.pt

Peter Kokol

Laboratory for System Design
University of Maribor
Smetanova Ul. 17, 2000 Maribor, Slovenia
Tel:+386-62-2207450
kokol@uni-mb.si

Abstract: *Programs are, nowadays, considered to be complex systems. Entropy and Correlation are the most widely used metrics available for the analysis of complex systems. This paper compares the application of these sorts of metrics, in the evaluation of the program organization. We verify that, the metrics based on the correlation are the most valuable for the identification of the program organizations.*

Key words*: Alpha metrics, Complex systems, Program organization*

1. SOFTWARE AS A COMPLEX SYSTEM

The complex systems share certain features like having a large number of elements, possessing high dimensionality, sensibility dependent on the initial conditions and representing an extended space for evolution [1,2]. Such systems are hierarchies consisting of different levels, each level having its own principles, laws and structures. There is no unique definition of Complex Systems through different fields, namely Economics, Management, Biology and Chemistry. However, there are some widely accepted notions, as the idea that the behaviour of a complex system cannot be understood from the sum of the individual behaviour of its parts.
A computer program is intended to fulfil one given task, its specification. Computer programs usually consist of a number of entities, such as modules and functions, on different hierarchical levels [3,4].
Concerning the laws of software engineering, the emergent characteristics of the entities listed above, are very different from the emergent characteristics of the program as the whole. Indeed, programming techniques as stepwise refinement, top-down design, bottom up design or more modern object oriented programming are only meaningful if different hierarchical levels of a program have distinguishable characteristics. Therefore, we need to adopt ways to assess global complexity, by using fractal metrics or entropy based metrics [5,6].
The increase of the complexity in the development and enhancement of software products has been announced informally long time ago [7]. If a project team does not control the product complexity, the product becomes more difficult to maintain and it usefulness decreases. Therefore, the complexity control is an important problem during all product life.

2. THE PROGRAM ORGANIZATION

The specification, the program and other software artefacts, are expressed in a representation scheme. Whatever representation scheme is selected, every software artefact can be translated to a set of symbols characters in the textual way. For example, the design graphical UML diagrams may be stored in XML textual form.
The set of symbols are organized in two levels, syntactical and semantic. The syntactical organization imposes order in the symbol sequences and the semantic organization imposes a limited set of meanings for each symbol sequence.
The syntactical organization is a well-know field of study and easily checked by language processors, such as compilers. The semantic organization, despite many efforts, is still unknown.

Pressures are always exercised in the evolution of software systems, for many reasons such as error correction, hardware updates, environment changes and the evolution of system perception upon the computer use [8]. The system development and evolution requires the understanding of the system and its implementation, and its cost must be minimal. Therefore, the higher is the semantic organization, the easier is the software evolution. Until now, the semantic organization level has been highlighted only through the hardness of changes. There is a need to quantify the semantic organization level, in order to control the software evolution.
The quantification of the semantic organization should be collected, as early as possible, in the process of software development. Hence, our work focuses on the evaluation of the program source complexity [9]. Due to the limitations, depicted later in this paper, the analysis of selected regularity is also extended to the object code.

3. METRICS FOR THE REGULARITY ASSESSMENT

In this section, we describe the metric types for the measure of the program regularities.
Our program sources are text files. The symbols to be analysed, can be reserved words, separation symbols or user entities (such as variables, constants and functions). We decided to restrict our analysis to the reserved words, because they are the only entities that express actions with meaning.

3.1 Entropy Metrics

The entropy metrics measure the regularity of the distribution of symbol frequencies [10]. We considered two different sorts of entropies, the Shannon and the Rény.
For any string $s=s_i^+$, with each $s_i \in S$ having the probability p_i, the Shannon entropy is defined as

$$H = -\sum_i p_i \log p_i \tag{1}$$

And the Rény entropy is defined as

$$H_r = \frac{1}{(1-\alpha)} \sum_i p_i^a \tag{2}$$

In the Rény entropy, the α value allows to overcome the greater (if greater than one), or the lesser (between zero and one) frequencies. In the Shannon entropy, the greater and the lesser frequencies contribute equally to the result.
We observed the programs with the same functionalities, defined by the same specification and implemented by different project teams, have different symbol distributions of programs with different set of functionalities. However, we are unable to differentiate the Shannon and the Rény entropies between a program with a given set of functionalities and the randomly generated programs (i.e., without meaning) with equal word distribution. The entropy metrics are, then, unsuitable to quantify the semantic organization. Therefore, we redirected our analysis to the correlation metrics.

3.2 Correlation Metrics

The correlation metrics measure the connection between the same symbol elements in different string positions. The short-range correlation, such as verified in the Markov chains, relates elements positioned very closed. The long-range correlation [11,12], named α metrics, filters short-range fluctuations and reveal the basic structure of the string. Therefore, the α metric is the most natural candidate for the assessment of the program structures.
There are several methods to evaluate the long-range correlation. We selected the commonly Brownian cumulative random walk, also known as Pheng's method [13].
The Pheng's method starts to identify the set of symbols, that forms the text, and encode them with a balanced numeric code (i.e, the sum of all codes is equal to zero). A graphic, named Brownian walk, is build drawing the point from the previous value, adding the current numeric code of the symbol. The Brownian walk is characterised by the root of mean square fluctuation F about the average of the displacement, defined as:

$$F^2(l) \equiv \overline{[\Delta y(l, l_0)]^2} - \overline{[\Delta y(l, l_0)]}^2, \ \Delta y(l, l_0) \equiv y(l_0 + l) - y(l_0) \quad (3)$$

where

l-is the distance between two points of the walk on the X axis

l_0-is the initial position (beginning point) on the X axis where the calculation of *F(l)* for one pass starts

y-is the position of the walk (the distance between the initial position and the current position on Y axis)

and the bars indicate the average over all positions l_0.

The *F(l)* can distinguish two possible types of behaviour:

- If the string sequence is uncorrelated (normal random walk) or there are local correlations extending up to a characteristic range, i.e., Markov chains or symbolic sequences generated by regular grammars, then $F(l) \approx l^{0.5}$
- If there is no characteristic length and the correlations are "infinite" then the scaling property of *F(l)* is described by a power law: $F(l) \approx l^{a}$ and $\boldsymbol{a} \neq 0.5$.

The F(l) characteristic function can be depicted in a logarithmic scale, and the α value is extracted from the slope and removing the lowest and the highest l values. The lowest l values are useless, because the exponential component and the other components, considered here as noise, have similar influence. The highest l values must be discarded, because the index difference is close to the sample size.

Figure 1 depicts one example of the F(l) characteristic function.

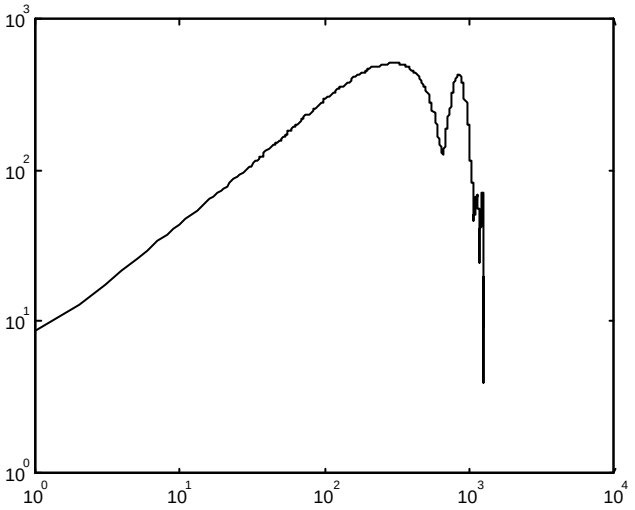

Fig.1 - F(l) characteristic function

The evaluation of the α value requires $O(N^2)$, where N is equal the string size. To have statistical valid results, the string size must be equal or greater than $2^{(C-1)}$, where C is half of the number of code symbols.

We have evaluated the α metrics to several hundreds of source and object code files, written C and Java programming languages, and the results show α values greater than 0.5. Randomly generated programs, with equal symbol distributions, show α values very close to 0.5.

Studies also reveal that, the α value becomes close to the source α value, with a syntactically correct generated program with equal symbol distribution and equal number of functions [14].

The Zipf metrics [O] can not be considered as a truly correlation metrics. They display number of all symbol occurrences, like the entropy metrics. Also, they identify the distribution law for all symbol occurrences.

The Zipf metric [15] focuses only on the syntactic, not on the semantic organization. Hence, it does not distinguish randomly generated programs from programs, with a meaning and equal number occurrences of each symbol.

4. WHERE CAN THE ALPHA METRICS BE EVALUATED?

The α metrics reveal some constraints on the string dimensions.
In the object code, C is equal to two. However, the string is large, because the compiler translated the source file represented in a high-level language, to the object file represented in a low-level language.
In the source code, C is larger (such as 16 for C and 23 for Java), but the source file is smaller. For example, the C program files should be larger than 2 KLOC.
We have identified the α values for statistically significant set programs, which revealed a very strong correlation between the long-range correlation of the source and the corresponding object files.
Therefore, the constraints announced above may be partially released by the selection of the file type.

5.CONCLUSIONS AND FURTHER WORK

The α metrics differentiate programs with, and without, meaning. We also verified, that the α metrics may give the same results in different phases of the software development process. Yet, there is a need to check, if the correlation between the source and the object files is extended to the design files.
It seems the α metrics is a promising framework for the control of multiple versions of the computer applications. Currently, we are analysing the model of the evolution for the α values in large projects with hundreds of versions.

REFERENCES

[1] Gell-Mann M: What is complexity, Complexity 1(1), 1995, 16-19.
[2] Morowitz H: The Emergence of Complexity, Complexity 1(1), 1995, 4.
[3] Cohen B, Harwood W T, Jackson M I, The specification of complex systems, Addison-Wesley, 1986.
[4] Watt, D. A.: Programming Language Concepts and Paradigms , Prentice-Hall, 1990.
[5] Harrison, W.: An Entropy-Based Measure of Software Complexity, IEEE Trans. on Software. 18(11), 1025-1029.
[6] Samadzadeh-Hadidi, M.: Measurable Characteristics of the Software Development Process Based on a Model of Software Comprehension. PhD Dissertation, University of Southwestern Louisiana, USA, May 1987
[7] Lehman,M.M., Programs. Life cycles and the laws of software evolution, Proc. IEEE, 15(3), 1980, 225-252.
[8] Lehman,M.M. and Belady; Program Evolution, Academic Press, London,1985
[9] Cardoso,A.I, Crespo,R.G, Kokol,P. How to measure the complexity of a program text?, Proc. ESI2002, 116-119, 2002.
[10] Li, W, On the Relationship Between Complexity and Entropy for Markov Chains and Regular Languages, Complex Systems 5(4), 1991, 381 - 399.
[11] Kokol, P., Brest, J., Žumer, V., Long-range correlations in computer programme, Cybernetics and systems 28(1), 1997, 43-57.
[12] Schenkel, A. and Zhang, J. and Zhang, Y, Long range correlations in human writings, Fractals 1(1), 1993, 47-55.
[13] Peng C K et al., Nature 356, 1992, 168.
[14] Cardoso,A.I. and Crespo,R.G, and Kokol,P., The alfa metric values for random and fully organized programs, in-printing
[15] Zipf,G.H.; Human behaviour and the principle of least effort: an introduction to human ecology. Addison-Wesley, 1949.