REVERSE BIFURCATIONS IN SOFTWARE DEVELOPMENT

P. Kokol[1], E. Feuer[2], M. Lenič[1], V. Podgorelec[1], A.I. Cardoso[3], L. G. Crespo[4]

[1] Laboratory for System Design
FERI, University of Maribor, Slovenia
kokol@uni-mb.si

[2] SZT Tszaki, Budapest, Hungary

[3] Mathematical Science Center
University of Madeira, Portugal

[4] University of Lisbon, Portugal

Abstract: *Computer software and the software development process belong to the class of complex systems. As a consequence software development process can be analysed by tools, techniques and concepts used in the chaos theory, especially logistic map. In the present paper we introduce an approach that enables assessment of current state the software development process based on previous software revisions. That further enables us to control the software development process in more efficient ways. For the analysis we used metrics that does not depend on any specific programming language.*

Keywords: bifurcations, alpha metrics, software development.

1. INTRODUCTON

Complex systems share certain features [1-5] like having a large number of elements, possessing high dimensionality and representing an extended space of possibilities. Such systems are hierarchies consisting of different levels each having its own principles, laws and potential structures shortly called emergent properties.
Computer programs, including popular information systems, usually consist of (or at least they should) number of entities like subroutines, modules, packages, classes, functions, etc., on different hierarchical levels. Concerning "laws of software engineering" or the concepts of programming languages the emergent characteristics of above entities must be very different from the emergent characteristics of the program as the whole. Indeed, the claim that 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 – clearly qualify computer programs as the class of complex systems that should be developed using a "complex development process" [10].
Intrigued by above we were interested if same characteristics of chaotic behaviour can be really found in the software development process. For that purpose we modelled it as an Reverse logistic iterative map which shows strong chaotic behaviour and tested the model on some large software development projects.

2. LOGISTIC MAP AND THE SOFTWARE DEVELOPMENT PROCESS

A very popular and interesting concept is the Logistic map defined by the following function

$$X_{n+1} = \pi X_n (1 - X_n) \tag{1}$$

Computer programs are normally developed and maintained by an iterative process – the computer programs evolves trough different successive versions (Figure 1). In the beginning we have to thought about many ideas representing different possible correct or incorrect solutions – creativity is large, entropy is large information content is low. In the next phase some choices are made, and probably the correct and optimal solutions emerge

– entropy and the number of ideas are reduced as the need for creativity, contrary the information content is enlarged. In the last phase we are working with a single idea (probably a very complex one), creativity and entropy reaches the minimum and information content the maximum (Single idea phase). Additionally, modern software development methodologies advise to work on the small modules first and then slowly integrate modules into larger units. Thereafter correlation between variables, concepts, ideas are on the short range in the beginning and then extend toward longer ranges.

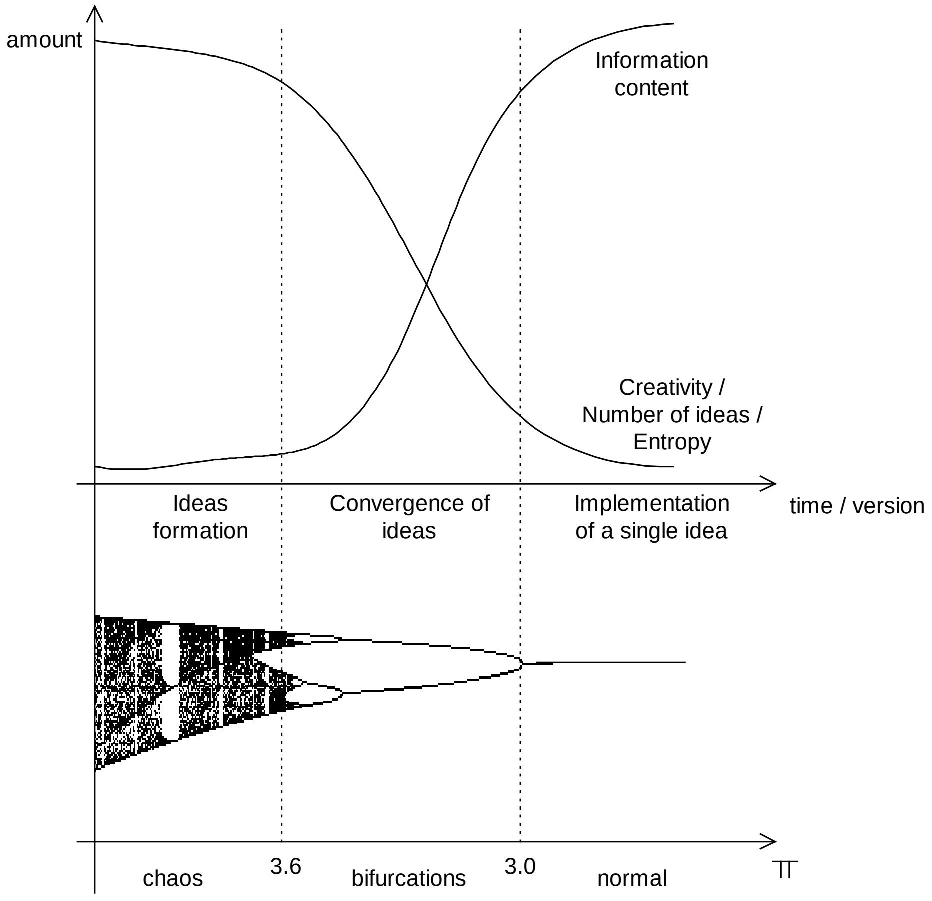

Fig. 1. The software development process and logistic map

Above phases can be matched with the three phases of the Logistic map graph: namely chaos, bifurcation and normal phase (bifurcation's represent the ideas, viewpoints, beliefs – simply ideas in the further text) but in the reverse order – thereafter we called it the Reverse logistic model of software development. Since the phase of a logistic map can be easily identified by the equation control parameter π, we can use the "logistic map ↔ software development process match" to test the hypothesis that our model is correct, assuming that we can measure the information content/entropy and correlation of each single program version. In the paper we use the alpha metric presented in next chapter for this manner.

3. ALPHA METRICS AND THE LOGISTIC PARAMETER

Alpha metric [6-9] is based on the long-range correlation calculation. In this paper we used the so-called CHAR method described by Kokol [8]. A character is taken to be the basic symbol of a computer program. Each character is then transformed into a six bit long binary representation according to a fixed **code table** (i.e. we have 64 different codes for the six bit representation – in our case we assigned 56 codes for the letters and the remaining codes for special symbols like period, comma, mathematical operators, etc) are used. The obtained

binary string is then transformed into a two-dimensional Brownian walk model (Brownian walk in the text which follows) using each bit as a one move - the 0 as a step down and the 1 as a step up. With such approach the program is transformed into signal and can be therefore analysed with tools and techniques used in the chaos theory. Another aspect of the conversion is programming language independence; therefore same method can be used on projects, which contain different programming language. It also enables to analyse specifications and other documents concerning the project if they are available in digital character based form.
An important statistical quantity characterising any walk is the root of mean square fluctuation *F* about the average of the displacement. In a two-dimensional Brownian walk model the *F* is defined as:

$$F^2(l) \equiv \overline{[\Delta y(l,l_0)]^2} - \left[\overline{\Delta y(l,l_0)}\right]^2 \tag{2}$$

where

$$\Delta y(l,l_0) \equiv y(l_0+l) - y(l_0)$$

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

  $$F(l) \approx l^{\alpha} \text{and } \alpha \neq 0.5.$$

The power law is most easily recognised if we plot *F(l)* and *l* on a double logarithmic scale. If a power law describes the scaling property then the resulting curve is linear and the slope of the curve represents α. In the case that there are long range correlation in the program analysed, α should not be equal to 0.5.
The α metric measures the information content of the program. Large difference of α value form 0.5 means more information content and less entropy and less creativity. As a consequence α values near 0.5 indicates phases near the chaotic phase. To make these assumptions more visible, we normalised α using the following equation:

$$\alpha_{nor} = 2|\alpha - 0.5| \tag{3}$$

In such manner the α_{nor} values lies between 0 and 1, 0 indicating the maximal entropy. To be able to calculate the phase using α values we should calculate π and thus rearrange the Logistic map into

$$\pi = \frac{X_{n+1}}{X_n(1-X_n)} \tag{4}$$

While the bifurcations in the Logistic map in our case represent the number of ideas and while the α_{nor} is inversely related to that number we have to calculate the normalised number of ideas with

$$I = 1 - \alpha_{nor} \quad (5)$$

3.1 α and development phases

When using presented approach we've noticed several distinguished states of projects. According to the trend of α values we were able to identify 7 different behaviours shown in table 1.

α trend	**Logistic Map Phase/ Phase trend**	**Development process Phase/ Phase trend**	**Current Phase**
Steady	Normal	Single idea	Normal/ Single idea
Falling	Toward chaos	Toward idea formation	If π less then 3 normal/ Single idea
Increasing	Toward normal	Toward single idea	If π greater then 3.6 chaos/ idea formation
Oscillating	Bifurcation or Chaos	Idea convergence or Chaos	If π greater then 3.6 chaos/Idea formation

Table 1. Phases and states of software development

4. CASE STUDY

The theory presented in the previous sections has been tried out on real world projects consisting of many modules and followed trough many versions. Some results of interesting modules are shown on figures 2 to 9.
Figures 2. trough 5. present the no - normalised alpha, size in lines of code an bytes and the π coefficient of the logistic map for a more or less normal development of a module. The chaotic behaviour is shown in the beginning (π is larger then 3) and goes trough to normal – we can conclude that the process has stabilised and that only some minor maintenance operations are still performed. The project documentation confirmed our conclusion.
Figures 6. trough 9. present the no - normalised alpha, size in lines of code an bytes and the π coefficient of the logistic map for a chaotic development of a module. As we can see during the whole period some pieces of code are added, deleted (Figures 7 and 8), information content (expressed by alpha) is also oscillating, and that chaotic behaviour is expressed in PI that is above 3 almost all the time. We can conclude that the process has not been stabilised yet and that the mayor development is still under the way. Going trough the project documentation showed that actually two development teams have done development with wholly different ides competing for resources and trying to prevail over the other party.

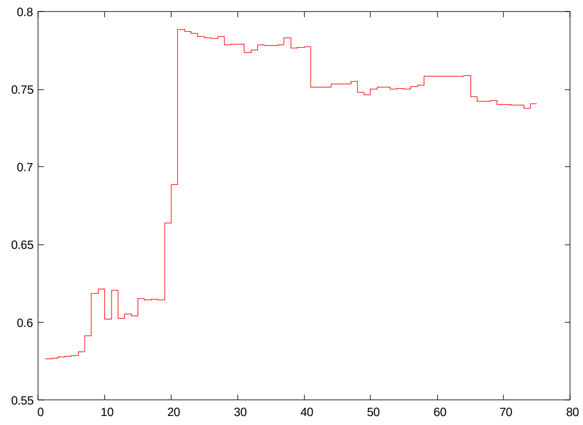

Fig. 2. Alpha for a module expressing normal behaviour

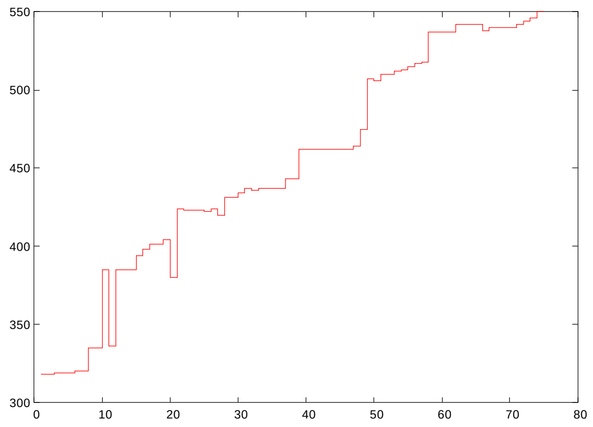

Fig. 3. LOC for a module expressing normal behaviour

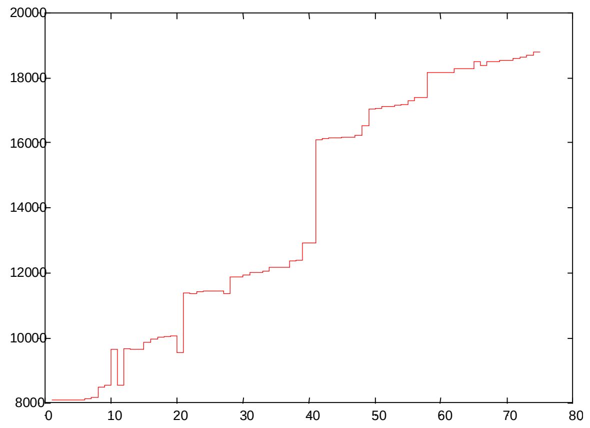

Fig. 4. Size in bytes for a module expressing normal behaviour

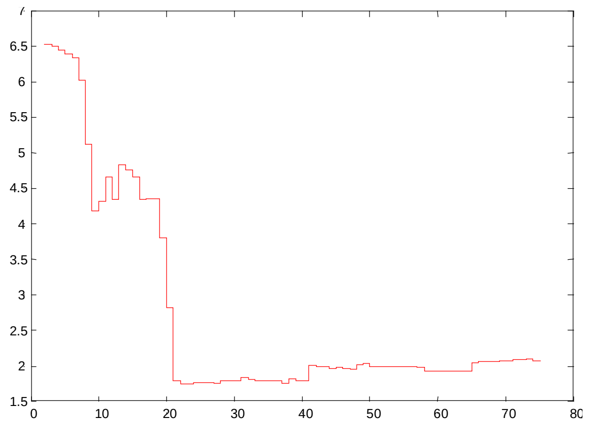

Fig. 5. π for a module expressing normal behaviour

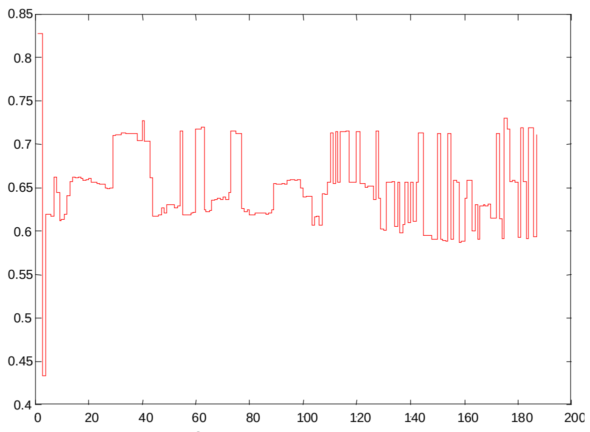

Fig. 6. Alpha for a module expressing chaothic behaviour

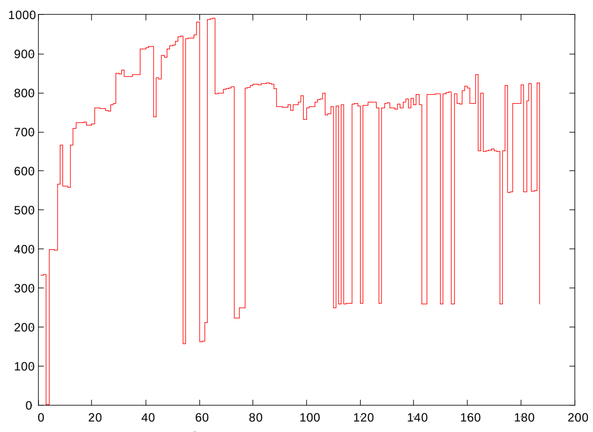

Fig. 7. LOC for a module expressing chaothic behaviour

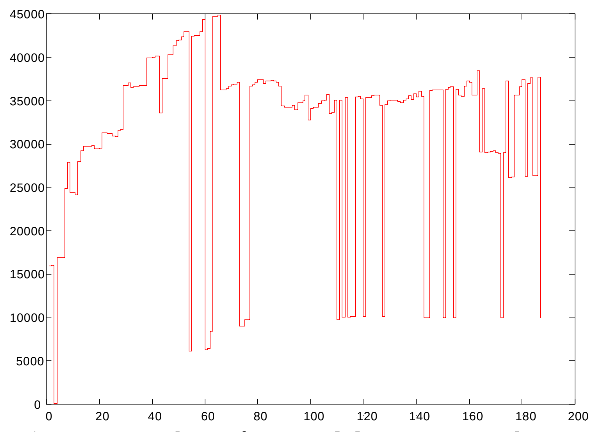

Fig. 8. Size in bytes for a module expressing chaotic behaviour

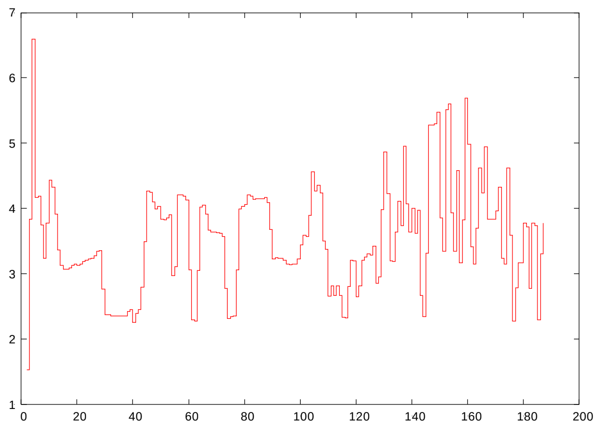

Fig. 9. π for a module expressing chaotic behaviour

5. CONCLUSION

In the present paper we presented a hypothesis that the software development process can be modelled as a reverse logistic map. We tested the hypothesis on more then 1000 modules of several large projects in several different programming languages and shown that the great extent the hypothesis can be confirmed. Very important feature of presented approach is its programming language independence.

6. REFERENCES

[1] Bar Yam, Y., "Dynamics of Complex Systems", Addison Wesley, 1997.

[2] Tucker, A.B. (ed.), "The Computer Science and Engineering Handbook", CRC Press, 1997.

[3] Kitchenham, B., "The certainty of uncertainty", Proceedings of FESMA (Eds: Combes H et al), Technologish Institut, 1998, pp. 17-25.

[4] Morowitz, H., "The Emergence of Complexity", Complexity 1(1), 1995, pp. 4.

[5] Gell-Mann, M., "What is complexity", Complexity 1(1), 1995, pp. 16-19.

[6] Kokol, P., Kokol, T., "Linguistic laws and computer programs", Journal of the American Society for Information Science, 47(10), 1996, pp. 781-785.

[7] Kokol, P., Podgorelec, V., Brest, J., "A wishful complexity metric", Proceedings of FESMA (Eds: Combes H et al), Technologish Institut, 1998, pp. 235–246.

[8] Kokol, P., Brest, J., "Fractal structure of random programs", Sigplan Notices, June 1998, pp. 33-38.

[9] Kokol, P., Podgorelec, V., Zorman, M., Pighin, M., "Alpha - a generic software complexity metric", Project control for software quality (Eds: Rob J. Kusters et al.), Maastricht : Shaker Publishing BV, 1999, pp 397-405.

[10] Cardoso, A.I., Crespo, G., "Is the software process a cahotic one ?", Working paper of Mathematical Science Center, Madeira University, 1999.