Transactions on Modelling and Simulation vol 10, © 1995 WIT Press, www.witpress.com, ISSN 1743-355X

Fractal program metrics: a new way to measure the characteristics of computer programs

P. Kokol, J. Brest, M. Mernik, V. Žumer
The Faculty of Technical Sciences, University of Maribor, Smetanova 17, 62000 Maribor, Slovenia

ABSTRACT

The application of measurement technology in software engineering was not recognised enough in last decades, but recent trends show that software metrics are becoming more and more important. Indeed the development and use of adequate measurement of software and its design process are essential to the production of cost-effective and quality software. Actually software metrics are not only significant as the analysis instruments, but regarding the present iterative software design paradigms, they are likewise becoming powerful synthesis mechanisms. The main objective of this paper is to introduce the fractal software metrics as an alternative to conventional software metrics.

INTRODUCTION

In general there are three main reasons why software metrics are becoming more and more important:

1. Measurement technology serves as a foundation for all scientific and engineering disciplines. Almost all advances in fields such as physics, biology, chemistry, medicine, etc., have occurred through interaction between measures of objects and events in the real world and their abstractions in the world of models and explanations. On the other hand, the application of measurement technology to software engineering and science is relatively new, and therefore its benefits are not recognised enough. But increasing cost of software development, testing and maintenance are the primary factors for the increasing popularity of the software measurement, which is also reflected in recent research [Stolen[7]] which has found an increase in empirical research during last few years also in the field of software design.
2. It is only logical and in fact more than necessary that software controlling computers run as effectively, efficiently and reliably as possible. In that manner it is reasonable to define quantitative properties of software and ways

hove to measure them. Thereafter software metrics are becoming extremely important for understanding software behaviour in every phase of the software life cycle. Actually they are not only significant as the analysis instruments, but regarding the present iterative software design paradigms, they are likewise becoming powerful synthesis mechanisms.

3. Albeit there are various definitions of software quality and various ways how to achieve it, the most usual approach is on carrying through more effective control of the software design process, emphasising the need for improved project management and the introduction and subsequent enforcement of standards. As a consequence, and taking into account the fact that the majority of conventional software metrics have failed to performed the measurement job in many cases (conventional metrics are to complex, don't measure the defined attributes, are not constructed as a part of an engineering process etc.), most effort has been devoted for the search of new metrics to monitor complexity, errors, reliability, usability, maintainability etc.

Unfortunately conventional software metrics have failed to performed the measurement job in many cases. Conventional metrics are to complex, don't measure the defined attributes, are not constructed as a part of an engineering process etc. overcome the weaknesses of conventional metrics and their design and use we followed the Schneidewind model to design a new class of metrics called, according to their theoretical base, fractal metrics [Kokol[5]].

SOFTWARE METRICS DESIGN

The framework on which our metrics approach is based consists out of some definitions adapted from [Schneidewind[4]] namely *behaviour factor, behaviour metric* and *validated metric.*

Behaviour factor F is an attribute of software which provides a *direct measure* of software behaviour like number of faults found, development time etc.

Behaviour metric M is a function whose input are software data (elementary software measurements such as number of edges, number of lines, number of variables etc.), and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute F. M is defined as an indirect measure of software behaviour. This means that M may be used as a substitute for F when F is not available.

Validated metric M' is one whose values $M_1, M_2, \ldots M_i$ have been shown to be statistically associated with corresponding factor values $F_1, F_2, \ldots F_i$. Since M is validated with respect to F it is necessary that F is valid. Thereafter it is assumed that F is valid by definition as a result of wide acceptance or historical usage.

Metrics Life Cycle Model

Metrics life cycle model was introduced by Kokol [Kokol[1]] in an article describing the use of spreadsheet software in software metrics development. The improved version of this model is shown in Figure 1. It enables a more systematic presentation of the basic activities in the metrics life cycle. Figure 1. shows that the process of developing and using new metrics consists out of following phases: modelling, implementation, validation, taking direct measures (empirical measurements), taking indirect measures (software analysis), practical use, data collecting and making improvements.

Modelling. In this phase a metrics model is formulated with help of mathematical, statistical, or other appropriate techniques for explanatory or predictive purposes. We distinguish three kinds of models [Conte[8]]: (1) *theoretical models* based on hypothesised relations among variables; (2) *data-driven models* based on statistical analyses; and (3) *combined models* in which intuition is used to determine the basic shape of the model and data analysis is used to determine the model's constants.

Implementation. In this phase the metrics model is implemented as an analysis tool usually as a computer program.

Taking direct and indirect measurements, and validation. During these phases the model constructed is validated using for example the validation methodology proposed by Scheidewind [Scheidevind[4]]. It consist out of six mathematically defined criteria: association, consistency, discriminative power, tracking, predictability and repeatability. During the validation process indirect measures obtained with the SAT are compared with direct measures, using six criteria given above. The direct measures represent a quantified base, and it is its purpose of validation to determine how well the model can predict this base and the degree of relationship between direct and indirect measurement. After the model is validated it is usual to compare it with other similar metric models.

Practical use, data collecting and making improvements. In these phases the validated model is used by software engineers, project managers, researchers and other interested professionals who can use it as a tool for performing their everyday work. Useful information that can lead to model improvements or enlarging data collections (data-bases) is also gathered during this phase.

Computer Support to Metrics Life Cycle Activities

In an earlier paper [Kokol[1]] it has been shown that spreadsheet software can successfully support many metrics life cycle activities and makes metric development and use process much more effective and efficient. During last few years the possibilities to extend this support with other tools has largely

extended (see Figure 1.). The statistical, spreadsheet and mathematical packages can be used during many metrics life cycle activities (activities in *italic* in Figure 1). The only exceptions in which the use of above tools is predominantly limited are measurement activities. But many other tools (provided by computer system or developed by interested users) can be used to partially automate the direct measurement process. In contrary the indirect measurement process can be fully automated using suitable software packages like compiler compilers.

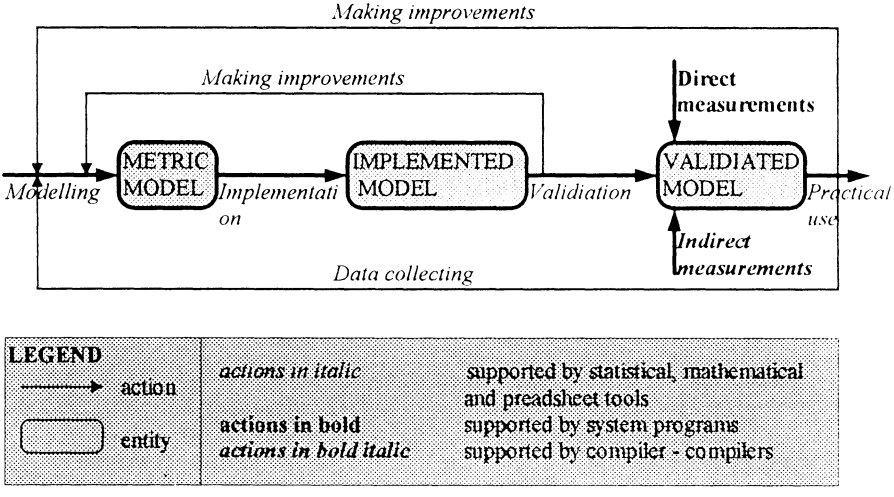

Figure 1. The metrics life cycle model

FRACTAL METRICS

We have successfully used computer supported design and metric life cycle models in the design, implementation and subsequently use of new software metrics. The most notable new metrics are so called ***Fractal Metrics*** [Kokol[5]]. They were developed in the manner to overcome weaknesses of conventional software metrics and as a consequence that the concept of fractals has caught the imagination of many scientist from many various fields. They are based on fractals and chaos theory [Peitgen[6]]. As possible fractal metrics we used: concentration metrics, character counts in identifiers, Zipf, Bradford, Voletera and similar word counting lows, level nesting according to Markov chains, percolation etc. Because of space limitations we would like to present an example in which the concept of self similarity is concerned with the characteristics of computer programs, namely structure of variables.

Brief Definition of Fractals

The concept of fractals has caught the imagination of many scientist from many various fields [Peitgen[6]] including medicine, engineering, philosophy, mathematics etc. The basic unifying concept underlying fractals and related disciplines of chaos and power lows is self-similarity. Self similarity, or invariance against changes in scale or size, is an attribute of many laws of nature and innumerable phenomena in the world around us.

A fractal is broadly defined as [Petigen[6]]

a shape made of parts similar to the whole in some way.

In that manner the fractals must give up some invariance: invariance to rotation, change of scale, translations, division, etc.

The Variable Names as Fractals

A variable name in the computer program can be in general defined with the following regular expression:

$$\{\text{Letter}|\text{Digit}|\text{Underline}\}\{\text{Letter}|\text{Digit}|\text{Underline}\}^{*} \quad (1)$$

Taking into account above equation it is easy to see that the variable names are invariant to:

1. **rotation**: we can rotate the letters of the variable name and still get a variable name;
2. **division:** we can take any part of a variable name and still get a new variable name; and
3. **concatenation:** we can concatenate two variable names and the result is another variable name.

It is shown [Schroeder] that such kind of words can be modelled as a Cantor set with the following fractal dimension

$$D=1/(1-\log(1-p_0)/\log N) \quad (2)$$

where N is the number of legal letters used in the construction of variable names and p_0 is the probability of delimiter symbols appearance.

VALIDATION OF FRACTAL METRICS

Fractal metrics were first validated using the model proposed earlier on the sample of FORTRAN programs constructed by students during a controlled experiment. The obtained data together with direct and indirect metrics are shown in Table 1. Then we used the Schneidewind validation model to validate

the fractal metrics. First two criteria namely association and consistency were performed with the computation of Pearsons correlation's matrices [Moore[10]] shown in Table 2. For the consistency validation we used the actual values (magnitude test) and for the association the rank values (rank test). All computations were accomplished with the help of a statistical software package. Because of the space limitation other criteria are not shown and will be presented elsewhere [Kokol 1995].

Table 1. The Direct and Indirect Metrics

DIRECT MEASURES					*INDIRECT MEASURES*			
Development Time(Minutes)	Number of Versions	Program Mark	Programmer Capability	Programmer Capability	LOC Count	McCabe Metric	Halstead Metric	Fractal D - Metric
120	39	9	4	5	94	5	143306	0.865231
135	33	6	4	4	40	2	40808	0.793324
105	1	6	5	5	57	6	114165	0.873196
105	14	6	3	3	55	3	36843	0.882412
60	9	6	2	3	50	2	33975	0.829128
90	42	6	3	2	39	5	12152	0.853917
90	47	8	3	3	51	2	28477	0.803905
105	40	7	5	5	36	4	25363	0.866015
45	15	10	5	4	32	2	27865	0.827969
40	23	10	5	4	88	4	182010	0.898221
135	1	7	3	3	51	3	168072	0.89286
135	21	10	3	3	43	4	121480	0.864104
150	11	6	2	3	51	4	176402	0.858746
120	28	8	5	5	62	4	146348	0.876053
120	10	6	2	3	40	3	14282	0.877913
120	17	6	4	4	32	3	10646	0.882387
30	17	10	5	4	25	4	11272	0.874753
90	9	9	4	4	39	4	57544	0.73736
45	7	10	5	4	40	5	37986	0.73736
135	29	7	3	3	37	5	17124	0.849642
135	45	6	3	3	41	4	33224	0.84132
60	14	10	4	4	36	3	11005	0.876917
120	4	6	3	4	43	6	26799	0.887532
30	9	10	5	5	21	4	13117	0.907395
105	20	9	4	4	29	3	18238	0.900803
135	63	9	3	3	23	4	9731	0.770989
120	19	6	3	3	29	4	18345	0.875095
90	33	8	3	3	27	5	20689	0.766121
90	33	9	4	4	30	3	26962	0.783282
120	21	8	4	5	39	3	81962	0.721057
120	6	7	3	4	33	3	66290	0.820767
90	14	8	4	4	31	3	12387	0.774292
60	41	10	4	4	38	3	41514	0.776728

A close look to tables shows that most correlation are relative weak but some correlation between direct measures and fractal metrics are on the 10% significant level. But it is the most important that that fractal metrics performs better then conventional ones.

Table 2. Correlation Coefficients

	TIME	VERSION	MARK	LOC	COM-C	GEN-C	McCabe	Halstead	Fractal D
TIME	1	0.172193	-0.61126	0.090538	-0.53918	-0.23173	0.109662	0.250846	0.046738
VERSION	0.172193	1	0.105437	-0.00445	-0.04588	-0.18096	-0.03107	-0.19361	-0.25867
MARK	-0.61126	0.105437	1	-0.04863	0.55636	0.335398	-0.06299	0.037964	-0.20387
LOC	0.090538	-0.00445	-0.04863	1	0.052668	0.166968	0.173177	0.747374	0.260574
COM-C	-0.53918	-0.04588	0.55636	0.052668	1	0.770553	0.1213	0.054876	-0.00088
GEN-C	-0.23173	-0.18096	0.335398	0.166968	0.770553	1	0.094582	0.184606	-0.00652
McCabe	0.109662	-0.03107	-0.06299	0.173177	0.1213	0.094582	1	0.162762	0.148722
Halstead	0.250846	-0.19361	0.037964	0.747374	0.054876	0.184606	0.162762	1	0.204673
Fractal D	0.046738	-0.25867	-0.20387	0.260574	-0.00088	-0.00652	0.148722	0.204673	1

CONCLUSION: FRACTAL VIA TRADITIONAL METRICS

The statistical comparison of Fractal Metrics with conventional metrics showed that fractal metrics are complementary to conventional and thereafter measures other program attributes. The statistical tests of the obtained results and the comparison with some empirical measurement made during experiments (development times, number of errors, number of faults, the experience, grade, quality etc. of programmers etc.) showed that these attributes closely relate to program quality. Thereafter Fractal metrics represent a basis for revealing important program characteristics, but their true meaning and value are currently not yet known and are undoubtedly possible candidates for intensive research.

REFERENCES

1. Kokol P., Application of spreadsheet software in software engineering measurement technology, *Information and Software Technology*, 31-9, 477-489, sep. 1989.
2. Srivastava A., Eustace A., ATOM: A system for building customized program analysis tools, *SIGPLAN* 29-6, 196-205, jun. 1994.
3. Levine J.R., Mason T., Brown D., *Lex & Yacc*, O'Reilly & Associates, Inc. Second Edition, USA, oct. 1992.
4. Schneidewind N .F. - Methodology for Validating Software Metrics, *IEEE Trans Soft Eng* 18 5, pp. 410-422.
5. Kokol P., Searching For Fractal Structure in Computer Programs, *SIGPLAN* 29-3, mar. 1994.
6. Peitgen R. et al, *Chaos and Fractals - New Frointers of Science*, Springer Verlag, 1993.
7. Stolen J. -The development of IS faculty: toward a maturing MIS field, *Data Base* 24-3, aug. 1993.
8. Conte S S. D., Dunsmore H. F., Shen V. Y., - *Software engineering metrics and models*, Benjamin/Cummings, Menlo Park, 1986.

9. Rousseau R., Bradford curves, *Information Processing & Management* 30 - 2, feb. 1992.
10. Moore D S, McCabe G P. - *Introduction to The Practice of Statistics,* Freeman, New York, 1993.
11. Kokol P. *Paper in preparation,* 1995.