Analyzing software with the complexity theory

Peter Kokol, Mitja Lenič
Laboratory for system design, Faculty of Electrical Engineering and Computer Science
University of Maribor, Maribor, Slovenia

Abstract

Computer software and the software development process belong to the class of complex systems. Continues monitoring and analysis of development process is crucial to minimize risk. For that reason many metrics and monitoring tools were developed. Most of metrics are programming language dependent. In the present paper we introduce a tool for analyzing program complexity using chaos theory, especially the idea of the logistic map. It enables assessment of current state of the software development process based on previous software revisions. Analysis is programming language independent and also enables analysis of object and compiled code.

Key Words
Software development, chaos theory, bifurcation, alpha metrics

1. Introduction

Software development process is a complex process, which results in computer software that clearly qualifies as the class of complex systems. Therefore such systems should be developed using a “complex development process” [10]. 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.

2. Chaos and software

Intrigued by above we were interested if same characteristics of chaotic behavior can be really found in the software development process. For that reason we have to transform computer software in a form suitable for tools for chaos analysis. For that reason a computer software component (module, class, etc.) is viewed as a signal – sequence of bytes or characters – and can be as such analyzed with tools used for chaos analysis. Using different tools multiple qualitative measures can be obtained. On other hand such measurements result in measured value that usually change for components throughout development process. Output is time sequence of measured values that can be analyzed using same idea as in computer components. For that purpose we modeled it as an Reverse logistic iterative map which shows strong chaotic behavior and tested the model on some large software development projects. In this paper we present a way to analyze software component using chaos theory. We also introduce framework for combining different transformation of software component to prepare it for tools for chaos analysis.

2.1 Transforming software to signal

Computer software can exist in binary or source code format. From the aspect of complexity both representations should correlate in same degree of complexity. Compiled software in binary format only adds some aspect of compilation and optimization (therefore it should definitely be more complex), but basically should have same properties as source code format. From that point of view it is not important in which format computer software module is analyzed, but it should be taken into account when transforming it to signal for the further analysis with other tools. Basically we can directly interpret computer program as sequence of bytes and convert it directly to a signal that can be then feed to other tools for chaos analysis. But on other hand, in some cases some information can be
redundant or not relevant for the analysis, what is especially true for source code. If we look at the signal produced from (binary/source) code treated as signed byte we can almost always observe some sort of noise (Figure 1).

Figure 1: Module of java source code directly interpreted as signal.

Signal clearly has chaotic characteristics and is definitely interesting for further analysis. On other hand conversion might introduce some inexistent order or correlation in the signal, so transformation must be carefully chosen. Commonly used tool for this conversion is the transformation to 1D Brownian motion. There are many different ways to implement such transformation, based on how the input source is interpreted. One simple way and probably most appropriate for binary input is to simply interpret input as a sequence of bits and if the value 1 move Brownian walk up otherwise move it down. On figure 2 the same module is presented with binary Brownian motion.

Figure 2: Same module with binary Brownian motion

It can be clearly observed, that converted signal now has some trend, although most interesting part for analysis with chaos is “noise” on trend.

Alternative to binary Brownian motion is simple conversion to Brownian motion, where each byte is treated as signed byte. Brownian motion is then constructed as a move up or down dependent on the input value. Binary and simple Brownian transformation are suitable for binary modules, but in the case of source code its use has to be reconsidered. Not all parts of the source code do represent an instruction used for compilation of software component. Those parts of code do not represent instructions and do not impact logic in module and are usually located in source code for easier understanding (comments) and better overview (formatting). Although they are very important part of the source code and can contain some important information about the state of the code, they impact measured values. In some cases we want to analyze just some points of interest in source code and have to be able to filter out “unwanted” data.

For that reason we also introduced char based transformation to Brownian motion. For each character a code for transformation to binary values has to be given in order to produce Brownian motion. This coding has to have some characteristics [citat], such as symmetric encoding and full coverage of binary code. Characters not included in coding table are simply discarded as irrelevant data. This approach can be sued to remove repetition of spaces and impact of code formatting style to produced measurement.

Another more sophisticated approach is to recode source code. Therefore new coding table is introduced based on regular expressions. To each expression symmetric code is assigned (from \(-N/2\) to \(N/2-1\)). New signal can be than processed using other (for example Brownian motion) transformation. That approach was used by Cordosa [citat], to filter out reserved words in compilers and measure alpha metrics to determine correlation between alpha and program structure.

We can go step further and can extract program structure using language specific compiler. Resulting signal is symmetrically encoded sequence of production numbers. It enables analysis of program structure and dramatically reduces signal size that can have very positive effect on computational intensive operations. Main disadvantage is availability of language specific compiler. All described methods except parsing are programming language independent and can be used on projects that use different programming languages. It also enables to analyze specifications and other documents concerning the project if they are available in digital character based form.

Next step after parsing is semantic processing. In case of compiler the binary representation of module is created, that can be then directly used in analysis. A comparison of source and binary code is given in [citat].
2.2 Signal to measure

After successful conversion of program to signal we have to select appropriate tools to analyze complexity. There has been some experiments with alpha metric that were quite promising. Alpha metric [6-9] is based on the long-range correlation calculation. In previous work a CHAR method was used to analyze programs. It is basically char based Brownian motion fixed on six bit encoding (i.e. 64 different codes for the six bit representation – 56 assigned codes for the letters and the remaining codes for special symbols like period, comma, mathematical operators, etc)

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:

$$F^2(l) = \left[ \Delta y(l, l_0) \right]^2 - \left[ \Delta y(l_0) \right]^2$$

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

The $F(l)$ can distinguish two possible types of behavior:

- 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 recognized 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 $\alpha$. In the case that there are long-range correlation in the program analyzed, a should not be equal to 0.5.

The metric measures the information content of the program. Large difference of a value form 0.5 means more information content and less entropy and less creativity.

Another interesting metric is the Hurst exponent that measures the smoothness of fractal time series based on the asymptotic behavior of the rescaled range of the process. Rescaled range analysis is a means of characterizing a time series or a one-dimensional (1-D) spatial signal that provides simultaneously a measure of variance and of the long-term correlation or “memory”. From that point of view it has similar characteristics as alpha metrics. Interesting effect using Hurst metrics is that when using it in combination with Brownian motion, there are small variations of measurement near 1.0.

Both presented metrics measure some sort of randomness in signal. Therefore we also decided to include serial correlation and Lyapunov exponent.

All metrics can be used to find some correlations (order) in software.

2.3 Measure to assessment

Development of software usually involves many people. In the process of communication usually some revision control systems are used to synchronize multiple snapshot of developed software. In such way every commition of software component creates current snapshot of the software. Trough development cycle software components evolve, leaving behind revisions/evolvement history. That history can be very useful source of information to assessment of development state of the project. The idea is to treat measurements as a signal and analyze it in the same manner as software using chaos theory.

We made use of the idea of logistic map. Signal has to be normalized to reflect its characteristics and then logistic map can be calculated. Since the phase of a logistic map can be easily identified by the equation control parameter $\pi$, we can use the “logistic map software development process match”.

State of software development can be matched with the three phases of the Logistic map graph: formation of ideas (chaos), convergence of ideas (bifurcations) and single idea (normal part of logistic map).
3. ProComplex

To analyze project complexity and assess software development state we’ve developed tool called ProComplex. It has the ability to automatically analyze a whole project and its revisions from version control system. Currently it uses concurrent versions system (CVS) [citat]. It also enables analysis of snapshots of project in pre-specified directory structure. That enables very transparent use on different types of project using different version control systems. As already described each method for measuring complexity is viewed as transformation (filtering) of input signal and measuring its complexity with some complexity metric (processor).

\[ \text{method} = (\text{filters, processor}) \]

Each filter or processor can have some specific arguments. In following description arguments are written in squared brackets \([\]\). For example CHAR alpha metric presented in [citat] can be viewed constructed:

\[ \text{CAHR alpha} = (\text{Char Brownian [6bit code]}, \alpha) \]

Filters can be sequentially combined in the same manner as using decoration pattern. For example using scanner filter to filter out reserved words in C and analyses alpha on its Brownian motion:

\[ \text{C reserved word alpha} = (\text{Simple Brownian (Scanner Coder[C reserved words]), alpha}) \]

Considering program transformation into signal and then its filtering/expansion/conversion using different filters, resulting signal can increase in length. For example calculation of alpha metrics is in range of \(O(n^3)\). In case of conversion of signal to binary Brownian motion, the computational requirement increases for 512 times. To address this problem we introduced two possibilities to reduce execution time. First reduces calculation precision of metrics. For example alpha metrics is not calculated by definition. It exploits characteristic of alpha measurement and dynamically adapts calculation step to maximal specified limit.

Reduction of calculation precision introduces some sort of noise that can be misinterpreted as chaotic effect in further analysis, but on the other hand, calculation precision is limited by a number of representations that also introduces additional noise.

Second solution was to distribute processing environment with distribution of workload between multiple computers. This solution can be also used in combination with reduced calculation precision since projects consist out of huge number of files and it multiple revisions.

Figure 4: ProComplex in action
4. Conclusion

Software development clearly classifies as complex process and should be therefore analyzed with tools for chaos analysis. Proper conversion of program in binary or source code format to signal is important and might have great impact on measurements. With introduced concept of filters and processors in tool ProComplex we plan to make extensive research on multiple freely available and preparatory projects from industry to confirm or reject our hypothesis, that project complexity and assessment of its state can be made using tools available from chaos theory.

4. Acknowledgement

This work was a part of SQUFOL project, founded by EU IST-2001-37003.

References