Is it really a "law" of languages encompassing everything from human to monkey to DNA? Or is it just an empirical curiosity, of no more system-theoretic significance than, say, a spurious correlation?

But the empirical evidence is mounting that when we finally manage to create a theoretical structure within which to speak mathematically about complex systems, power laws like Zipf's will appear as a necessary consequence of such theories. When (and if) they do will be the moment when they pass from interesting empirical curiosities to laws of nature.

—John L. Casti

Introduction

As computers become faster and cheaper, their popularity continues to increase. Whatever we do and wherever we go, we find computers. Computer programs can save time, increase efficiency, and organize our lives, but they can also prove disastrous when they fail. It is, therefore, of the greatest importance that computer programs possess such characteristics as reliability, efficiency, and maintainability. To prevent disasters, we must be able to differentiate between "good" and "bad" computer programs according to these characteristics. In this manner, we must be capable of measuring their quality. Unfortunately, the application of conventional software metrics in performing this task (for example, Halstead's software science metrics, McCabe's complexity, and lines of code metrics (Conte, Dunsmore, & Shen, 1986) have often not been successful (Fenton, 1991; Kokol, 1989)).

On the other hand, "infometric" research (Eghe & Rousseau, 1990) has developed the concept of information production process (IPP) which is applicable in econometrics, demography, scientometrics, informetrics, bibliometrics, and linguistics.

As a consequence of the weaknesses of conventional software metrics and the success of infometric research, we decided to analyze the possibility of using linguistic laws (Rousseau, 1994) in the software quality assessment process. It is the aim of this article to present the first results and to encourage others to advance our work.

Linguistic Laws

Recent research (Eghe & Rousseau, 1990) shows that the concept of information production process can be successfully applied to the field of linguistics research. The general IPP concept introduces:

- Production units consisting of sources that produce items and
- Productions being defined as the number of items produced by a source.

In linguistics, words in natural language are defined as production units and word occurrences as items. A law frequently used to model the IPP in linguistics (a linguistic law) is Zipf's law (Rousseau, 1994).

Zipf's Law

One type of empirical relation that turns up frequently in the linguistic research is the power law. A particularly intriguing (Casti, 1995) illustration of this kind of law is Zipf's law (Bunde & Havlin, 1994; Eghe & Rousseau, 1990; Kokol, 1994; Rousseau & Zhang, 1992). It expresses the relation between word rank \( r \) (rank order of words) and word frequency \( f(r) \) (the frequency of the appearance of words). It states that, to a
very good approximation, word frequency (number of occurrences of a word) in a given text is inversely proportional to its word rank (words are ranked in decreasing order according to their number of occurrences, i.e., the most common word has rank 1, the second most common word has rank 2, and so on). The general form of Zipf's law is (Rousseau, 1994):

$$f(r) = \frac{F}{r^\beta}$$  \hspace{1cm} (1)

where $F$ and $\beta$ are constants. An often used derivation of the general Zipf's law is:

$$f(r) = \frac{1}{r^n \ln(a \cdot R)}$$  \hspace{1cm} (2)

where $R$ is the number of different words in language and $a$ is the language-sensitive constant [for example, $a = 1.78$ for the English language (Casti, 1995)].

Note that Zipf's law has also been recently used in the analysis of biological systems. For example, Fitzpatrick (Hillman, personal communication, 1995) shows that the size of the cortex as a whole and the size of functional areas follows the Zipf's law; and Stanley (Casti, 1995) uses the same law to differentiate between coding and noncoding sections of DNA.

Zipf's Law and Computer Programs

Like ordinary texts written in natural language, computer programs can be also represented as strings of words. They are written according to strict grammatical rules [context-free and regular grammars; see, for example, Floyd and Beigel (1994) or Devlin (1992)]. They have vocabularies of keywords, reserved words, and operators, from which programmers select appropriate keywords during the programming process. In addition, they have numbers and vocabularies of identifiers (names of user-defined variables, procedures, functions, modules, labels, etc.) created by programmers. These are, in general, not language dependent and distinguish computer languages from natural ones. Another great difference between natural and computer languages is that computer languages are much more restricted and formal than natural languages and have much more limited vocabularies.

Methods and Results

Regarding the fact that identifiers and numbers are not language dependent, we have ignored them in our analysis. Using the IPP definitions introduced above, we define reserved words and operators as production units and their occurrences as items.

To be able to analyze a great number of computer programs written in different programming languages, we constructed a computerized tool, called PROMIS (Kokol, 1995). PROMIS (see Fig. 1) enables one to perform various computer program analyses almost automatically. Using it, we have analyzed:

- 130 student FORTRAN programs; the size of individual programs is from 100 to 200 lines of code, and all are programmed to perform similar tasks, solving problems from the technical domain such as circuit analysis, filter design, and numerical methods;
- professional public domain C++ programs; the size of individual programs was from 1,000 to 10,000 lines of
TABLE 1 Parameters, correlation coefficients R, and results of Kolmogorov-Smirnov tests (KS α). \( \text{Min } a \) is the minimal value of a constant \( a \) calculated for a given programming language, \( \text{Max } a \) is the maximal value of a constant \( a \) calculated for a given programming language, and the \( a \) is calculated for the merged programs.

<table>
<thead>
<tr>
<th></th>
<th>( \text{Min } a )</th>
<th>( \text{Max } a )</th>
<th>a</th>
<th>R</th>
<th>KS α</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORTRAN</td>
<td>0.0121</td>
<td>0.051</td>
<td>0.0303</td>
<td>0.964</td>
<td>0.1</td>
</tr>
<tr>
<td>C</td>
<td>0.009</td>
<td>0.062</td>
<td>0.0357</td>
<td>0.930</td>
<td>0.1</td>
</tr>
<tr>
<td>C++</td>
<td>0.008</td>
<td>0.059</td>
<td>0.0323</td>
<td>0.735</td>
<td>0.005</td>
</tr>
</tbody>
</table>

code, and the programs had to perform various tasks from different nontechnical domains such as compilers, operating system routines, user utilities, information systems, and business programs; and

- professional programs written in C, the size of individual programs was from 500 to 2,000 lines of code, performing similar tasks from the programming language domain such as parsers, scanners, code generation routines, and semantic analyses routines.

We analyzed each program individually and each programming language individually. This latter means that we have also merged all programs written in the same language into a single program and then analyzed this merged program.

For curve fitting, we needed to compare the theoretical Zipf’s distribution with the observed word frequencies and rank. Here we used the least square method (simplex fitting) found in a statistical package. To prove the statistical significance of the similarity between observed and theoretical distributions, we employed Pearson’s correlation coefficient and the Kolmogorov-Smirnov test (Eghe & Rousseau, 1990).

FIG. 2. The predicted and observed word and operator occurrences for FORTRAN-merged program.

FIG. 3. The predicted and observed word and operator occurrences for C-merged program.

FIG. 4. The predicted and observed word and operator occurrences for C++-merged program.

Discussion

The results of both analyses (see Table 1, and Figs. 2-4) show that the occurrences of reserved words and operators in computer programs follow Zipf’s distribution. The fit was statistically significant regarding both Pearson’s correlation coefficients and Kolmogorov-Smirnov tests.

It is interesting to note that the constant \( a \) from Equation 2 is very similar for all three computer languages. It
is our belief that this similarity is the consequence of all three languages belonging to the same paradigm of programming languages—namely, the imperative paradigm (Wegner, 1989).

The worst curve coefficient is for the C++ programming language programs. The correlation coefficients for FORTRAN and C programs are much higher. This may be due to the fact that all FORTRAN programs are very similar and from the same domain (also true for the C programs), while the C++ programs are very different and from various domains. Programs from different domains and solving distinct problems seem to require different operators and different reserved words. The consequence is that the set of most common words in program $i$ and the set of most common words in program $i+1$ are very dissimilar. This disturbs the distribution of ranks and frequencies (see Fig. 4).

Possible Application: Zipf’s Law and Software Metrics

From the analysis presented above, we see that the constant $a$ from Equation 2 was different for every computer program (see Table 1 for maximal and minimal values of $a$). This fact opens the possibility of using the constant $a$ as a software quality metric—the Linguistic metric. To prove the usability of the Linguistic metric, we analyze FORTRAN programs to study the relation between $a$ and some empirical software measures, and the relation between $a$ and some conventional software metrics.

The reason for selecting relatively short student FORTRAN programs is that we have developed an objective method for the assessment of their quality, expressed by the program mark. Program mark is an average grade calculated from grades given by five experts. Each expert rates the following properties of the program: Readability (is the program easy to understand without additional documentation), user interface (is the program easy to use even for non-experts), reliability (does the program contain any errors), robustness (does the program calculate the desired output for proper input data, or warn the user for improper input data), and programming style (use of structured programming, content of comments, proper use of indentation, descriptive identifiers, etc.).

Another empirical metric, sometimes used as an indicator of quality, is the number of the final version of a program—program version (version number indicates how many times the program has been changed). If we propose that every program change means an improvement to the program—either removal of an error or any other improvement—we can argue that the higher program version means higher quality.

The comparison between linguistic and conventional software metrics, like Halstead metrics, McCabe’s complexity, and program length (Conte, et al., 1986), in predicting the above, two empirical metrics are shown in Figure 5. We see that the linguistic metric performs much better in this case than conventional metrics. But we must prove that this is true in more general terms, and what will require more extensive analysis in the future.

Conclusion

This study shows that Zipf’s law has potential use in analysis of computer programs. This finding opens the possibility of using linguistic laws as a basis for developing a new class of software metrics (i.e., linguistic metrics). Our study shows that linguistic metrics can perform better in the prediction of some software quality characteristics and are viable candidates for future research in the software metrics area.

References


FIG. 5. The comparison between linguistic and conventional software metrics.