How Difficult Is It to Comprehend a Program That Has Significant Repetitions: Fuzzy-Related Explanations of Empirical Results

Christian Servin, Olga Kosheleva, and Vladik Kreinovich

Abstract In teaching computing and in gauging the programmers' productivity, it is important to property estimate how much time it will take to comprehend a program. There are techniques for estimating this time, but these techniques do not take into account that some program segments are similar, and this similarity decreases the time needed to comprehend the second segment. Recently, experiments were performed to describe this decrease. These experiments found an empirical formula for the corresponding decrease. In this paper, we use fuzzy-related ideas to provide commonsense-based theoretical explanation for this empirical formula.

1 Formulation of the Problem

Why should we measure comprehension complexity. Some programs are easier to understand, some are more complex and thus, take more time to understand. In teaching computing, it is desirable to be able to estimate how much time it will take for students to understand a given program.

Similar estimates are useful for managing teams of professional programmers. When they write new code, we can gauge their productivity, e.g., by the number of lines of code. However, it is well known that in many cases, programmers do not write code "from scratch": in the process of writing code programmers often use

Christian Servin
Information Technology Systems Department, El Paso Community College (EPCC)
919 Hunter Dr., El Paso, TX 79915-1908, USA, cservin1@epcc.edu

Olga Kosheleva
Department of Teacher Education, University of Texas at El Paso, 500 W. University
El Paso, Texas 79968, USA, e-mail: olgak@utep.edu

Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso, 500 W. University
El Paso, Texas 79968, USA, e-mail: vladik@utep.edu

available code snippets and modify them so that they can be appropriately incorporated into the newly designed code. To be able to do it, the programmer needs first to understand the available code. To gauge the programmers' productivity – and to properly estimate the time needed to complete the corresponding task – it is desirable to estimate the time needed to comprehend the given code segment.

How comprehension complexity is measured now: MCC. Several measure have been designed to gauge the comprehension complexity. Judged by the number of citations, the most widely used measure of comprehension complexity is so-called McCabe's cyclomatic complexity – MCC, for short [6]. Crudely speaking, the complexity of a simple no-branching no-loops program is 1, and each if-statement, each loop adds one to this complexity.

Limitation of MCC. For a program whose parts are very different from each other, MCC provides a very good measure of comprehension complexity.

However, many programs contain parts which are very similar. This makes perfect sense: there are only so many different clever ideas and ingenious tricks, so in a reasonable long program, where lots of these ideas have been applied to make this program more efficient, inevitably we will have the same idea used several times.

This is similar to the well-known pigeonhole principle often used to prove results in theory of computation: if N pigeons are all in cages, and the overall number of cages n is smaller than the number of pigeons, then there must be at least one cage than contains several pigeons. Similarly, if we have N parts using clever ideas, and the number n of used ideas is smaller than N, there must be at least one idea that is used in several parts of the code.

And when the same idea is used in different parts of the code, these parts become similar. The problem is that MCC does not take this similarity into account: whether we consider a two-part code consisting of completely different parts or a two-part code with two similar parts, the MCC is the same in both cases – the sum of MCCs of both parts.

Of course, in reality, similarity between the parts makes the code easier to understand. It is therefore necessary to take this into account.

Experimental data. To analyze how the comprehension complexity decreases with repetition, researchers measured the time that it takes to understand the part of the code that for the second, third, etc., times uses the same idea; see [2] for details. They concluded that, on average, the comprehension complexity C_i of the i-th repetition is related to the complexity C_1 of the first repetition by a formula $C_i = q^{i-1} \cdot C_1$ for $q \approx 0.6$.

Remaining challenge and what we do in this paper. As we have mentioned, the main reason for the study [2] was to better gauge comprehension complexity. This can help both in teaching computing and in gauging the productivity of programmers. From this viewpoint, while empirical formulas are helpful, it is usually more reliable if a formula has some convincing theoretical explanation. This way, we can more sure that this formula – derived based on a few cases – can be safely applicable to other cases as well.

This is what we do in this paper: we provide a fuzzy-related explanation for the above empirical formula.

2 Analysis of the Problem and the Resulting Explanation

Our approach. In our analysis of the problem, we will use natural commonsense ideas about this situation. Such ideas are usually described by using imprecise ("fuzzy") words from natural language. So, if we want to come up with numerical dependencies, we need to translate these commonsense descriptions into precise terms. This need was first well understood in the 1960s by Lotfi Zadeh, who called such translation techniques *fuzzy*, and who developed successful techniques for control (and similar situations); see, e.g., [1, 4, 7, 8, 9, 11]. In this paper, we will use somewhat different but related techniques, also inspired by Zadeh's original ideas.

Let us start our analysis. In general, if we have a code segment with comprehension complexity C, then, if we encounter a similar code segment further on, the comprehension complexity of the consequent sequent should be smaller. Of course, the comprehension complexity of this consequent code segment depends on the complexity of the original code segment:

- if the original code segment was difficult to understand, the consequent segment will also be not very simple, while
- if the original code segment was rather simple, the consequent segment will be even simpler.

In other words, the comprehesion complexity of the consequent code segment depends on the comprehension complexity C of the original code. Let us denote comprehension complexity of the consequent code segment by $f(C)$.

Based on common sense, what can we say about the function $f(C)$? First, we know that $f(C) < C$. Also, we are talking about reasonably small code segments, segments that are, eventually, easy to understand by an average programmer. This is especially so in the educational environment, when we start with simple code. So, the values C that we are interested in are relatively small – we are not talking about complex codes with hidden logic that programmers from competing companies try to reverse engineer.

Situations when we are interested in the dependence $y = f(x)$ between two physical quantities x and y and we know that x is small are common in physics; see, e.g., [3, 10]. In such cases, a usual technique is to take into account that for small numbers x, their squares, cubes, etc., are much smaller than the original number. For example, for $x = 0.1$, its square is $x^2 = 0.01 \ll x = 0.1$, and its cube is even smaller. Thus, a reasonable idea is to expand the unknown dependence $y = f(x)$ in Taylor series $y = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots$ and ignore terms which are quadratic or of higher order in terms of x – since these terms are much smaller than x. As a result, we get a linear dependence $y = a_0 + a_1 \cdot x$.

It is important to notice that by "small", physicists mean small in the physical sense – much smaller than possible large values, it is not always correlated with the numerical value being small. For example, in terms of changing a human state 1 second is very small, but if we describe the same amount in nanoseconds, we get one billion – mathematically it is a big number, but from the physical viewpoint, the corresponding period is still small.

Since the value C is small, it makes sense to apply a similar idea to the dependence $f(C)$, and thus, conclude that $f(C) = a_0 + a_1 \cdot C$ for some a_0 and a_1.

When the code segment is very simple, i.e., when $C \approx 0$, a similar consequent segment should also be simple. So, we have $f(0) = 0$. Thus, in the linear formula, we have $a_0 = 0$ and $f(C) = a_1 \cdot C$.

What we can now explain and what still needs to be explained. So, we have $C_2 = a_1 \cdot C_1$, $C_3 = a_1 \cdot C_2$, ..., and, in general, $C_{i+1} = a_1 \cdot C_i$. By induction, we can conclude that for all i, we have $C_i = a_1^{i-1} \cdot C_1$. This is exactly the observed dependence, with $q = a_1$.

So, we explained the general shape of the formula. What remains to be explained if why we have $q \approx 0.6$. To explain this value, let us continue our analysis.

Let us continue our analysis. An empirical fact is that the time needed to comprehend the next segment is significantly smaller than the time needed to comprehend the original segment. This decrease is caused by the fact that the consequent segment is similar to the previous one.

A consequent similar fragment is similar to the previous one, but these two segments are not almost identical: if two code segments were almost identical, we could probably combine them. So, it is reasonable to conclude that there is significantly more difference between the two segments than there is similarity.

How can we gauge this? The decrease in time is causes by similarity. If we start with time C needed to comprehend the original segment, then:

- the similarity causes the decrease $q - C \cdot q = (1 - q) \cdot C$ from C to $q \cdot C$, while
- the non-similarity leads to the need to still spend the time $q \cdot C$ on comprehending the new segment.

Thus, the fact that there is more difference than similarity means that the value $(1 - q) \cdot C$ is significantly smaller than $q \cdot C$.

Here we have another natural-language term – "significantly smaller". How can we describe it? Similarly to what we did earlier, we can try to assign, to each numerical value x, a value y that is typical among all the values which are significantly smaller than x. In other words, we are looking for a function $y = g(x)$ that would assign such typical value y to each x. Similarly to our first idea, we can conclude that the dependence $y = g(x)$ should be linear, i.e., should have the form $y = a \cdot x$ for some value a.

To find this value a, we can take into account that now, we have two cases of a quantity being significantly smaller than the other:

- first, the time $q \cdot C$ needed to comprehend the consequent segment is significantly smaller than the time C needs to comprehend the original segment, and

- second, the time $(1-q)\cdot C$ corresponding to similarity between the two segments is significantly smaller than the time $q\cdot C$ corresponding to the difference between the two segments.

If we apply the above formal description $y = a\cdot x$ of the statement "x is significantly smaller than y", then:

- from the first case, we conclude that $q\cdot C = a\cdot C$, i.e., that $a = q$, and
- from the second case, we conclude that $(1-q)\cdot C = a\cdot q\cdot C$, thus $1-q = q^2$.

This quadratic equation is easy to solve, so we conclude that

$$q = \frac{\sqrt{5}-1}{2} = 0.618\ldots \approx 0.6.$$

Thus, we have explain the numerical value of the parameter q as well. So, the empirical formula (1) is fully explained.

Comment. The above value is known as the *golden ratio* or *golden proportion*. It is worth mentioning that there are other fuzzy-related arguments that lead to this ratio; see, e.g., [5].

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), HRD-1834620 and HRD-2034030 (CAHSI Includes), EAR-2225395 (Center for Collective Impact in Earthquake Science C-CIES), and by the AT&T Fellowship in Information Technology.

It was also supported by a grant from the Hungarian National Research, Development and Innovation Office (NRDI).

References

1. R. Belohlavek, J. W. Dauben, and G. J. Klir, *Fuzzy Logic and Mathematics: A Historical Perspective*, Oxford University Press, New York, 2017.
2. D. R. Feitelson, "From code complexity metrics to program comprehension", *Communications of the ACM*, 2023, Vol. 66, No. 5, pp. 52–61.
3. R. Feynman, R. Leighton, and M. Sands, *The Feynman Lectures on Physics*, Addison Wesley, Boston, Massachusetts, 2005.
4. G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.
5. M. Koshelev, "Fuzzy logic explains the Golden Proportion," *International Journal of Intelligent Systems*, 1997, Vol. 12, pp. 415-417.
6. T. McCabe, "A complexity measure", *IEEE Transactions on Software Engineering*, 1976, Vol. 2, No. 4, pp. 308–320.

7. J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, Cham, Switzerland, 2017.
8. H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.
9. V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.
10. K. S. Thorne and R. D. Blandford, *Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics*, Princeton University Press, Princeton, New Jersey, 2021.
11. L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.