A Cognitive Model for Complexity Metrics

Tom Klemola

Centre for Object-Oriented Technology Applications and Research,
University of Technology, Sydney
New South Wales
AUSTRALIA
E-mail: klem@socs.uts.edu.au

Abstract. Cognition involves both short-term and long-term memories. Understanding how they are used can lead to software metrics that predict human performance in software development, and can be used to assess and improve the understandability of both text and code. Short-term memory is most affected by overloading over a short period of time. Long term memory is affected by the frequency and diversity of exposure to a concept and degradation over time. Metrics can be defined based on current understanding of both short-term and long-term memory performance to predict where and how often errors may occur.

1 Introduction

When programs were written in assembly language, a line of code was limited in its meaning. Most programmers would acquire an expertise that was similar to that of other programmers since the variation of program segments was small. In this environment, LOC was a good predictor of error rate and development time [Henderson-Sellers96].

High level languages with abstract data types change the range of variations that can appear in a program segment. A given line of code could have several identifiers for instance in a function call with three arguments where each argument is of a different type. The same program in assembly language would be represented over several lines where each line would perform a simple operation. The greater diversity of high level language programs means that two given lines of code are likely to be different, and hence require a different level of involvement from the programmer. This results in vastly different production times and different error rates for programs of a similar length.

High-level language programs tend to resemble more the particular application

domain structure (i. e. inventory control) and solution domain structure (i. e. GUI) in question. In such increasingly sophisticated environments, metrics such as LOC/module are usually less consistent than with assembler code, their greatest potential use being when programs in the same application domain written by the same people are compared.

Consequently, module size becomes an increasingly poor starting point for quality assessment of design and code. As a replacement, the trend in software metrics was to focus on software complexity [McCabe76; Henderson-Sellers96]. Measuring complexity should reflect attributes of human comprehension since complexity is relative to human cognitive characteristics [Cant95]. Here we focus on aspects of short and long term memory, as well as the effect of experience.

Furthermore, as programs begin to look more like text, the need arises for metrics that will be good at predicting text comprehension complexity. Hence, there is a need for a model of comprehension that can be applied to both text and programs, in order to derive metrics to predict human performance at all stages in the software development process.

2 Features of Cognitive Performance

Short-term memory (STM) limits, comprehension by chunking and tracing, long-term memory (LTM) structure, LTM degradation, and acquired expertise are features of cognition. These can be used to model cognitive processes.

STM limitations vary depending on the individual and on what kind of information is being retained [Kintsch98]. For the purposes of understanding text, STM has been measured at 4 concepts [Broadbent75]. This would suggest that any sentence that is using more than 4 concepts to make a point may not be immediately understood.

STM has a duration of 20-30 seconds [Reed96]. When reading new material, one must retain the "old" concepts in STM, using them to define new concepts. When the density of such terms is high, the risk
of error increases [Kintsch98].

Information is processed in the human brain by grouping concepts together into abstract units called chunks [Cant95]. Chunks can also be patterns of association. Chunks can be composed of chunks, so that an entire document or a set of documents can be thought of as a concept or chunk.

Chunks can become part of long-term memory if they are encountered often enough. Once in long-term memory, they can be recognised when they are encountered.

Tracing is the process of searching through text to find other references to the same term or symbol [Cant95]. The significance of a concept is derived by tracing the text

for instances of its usage. Tracing has been observed as a fundamental activity in program comprehension [Boehm-Davis96].

When comprehension involves recognition of something known rather than deriving the solution to a problem, the comprehension process has been measured to be about 10 times faster [Kintsch98]. Long-term memory structure can be described as a network. The structure of this network can be seen as being determined by order and frequency of exposure to particular concepts. When a concept has many references, it is easier to remember. Conversely, when an item is scarcely mentioned, it is quickly forgotten. Long-term memory has been found to degrade with time to about 30% after 3 years and remain fairly stable after that [Conway91].

3 Acquired Expertise

Expertise, or the ability to solve problems on recognition, can be acquired after many years of problem solving in a given domain [Ericsson95]. This can colour a person's perception of which way is best, since the way they know works very well for them. It can also result in the production of text that contains word usage that few other people can easily understand. This suggests that cognitive complexity can be quite subjective.

A study of expert troubleshooters found that they favoured familiar solutions to problems that resembled a familiar problem even when the solution was incorrect. The repeated exposure to the same problem had trained them to apply the same solution when confronted with superficially similar symptoms. Novices were faster at troubleshooting in this situation [Besnard99].

Matching chunks is analogous to pattern matching. If a chunk is matched, then the group of concepts can be treated as a single concept.
When an individual recognises all the patterns of a domain, he has acquired expertise in that domain.

4 Complexity Metrics

Some traditional complexity metrics can be supported by the fact that they are clearly related to cognitive limitations. These include LOC, fan-out, and decision points such as McCabe's cyclomatic complexity [McCabe76].

When a program segment has a high number for LOC, it will have enough features such as identifiers, that it becomes difficult to understand correctly. Simply following up all the uses of a particular identifier can take more than 20-30 seconds and result in error if time is limited.

When a program segment has a high fan-out, the time spent tracing references can increase and comprehension is compromised. Finally, when there are many possible

paths to be taken within a module as when McCabe's Cyclomatic Complexity would get a high value, again tracing becomes demanding and it can be difficult to correctly interpret.

As programmers gain experience, they will avoid repeating mistakes that the complexity of their particular domain will precipitate in a novice. The task of predicting the error such a programmer might make becomes dependent on their past experience and their current project. This consequence of individual performance characteristics poses a challenge to the design of general complexity metrics.

5 Cognitive Complexity Metrics

STM can be thought of as a container, where a small finite number of concepts can be stored. If data are presented in such a way that too many concepts must be associated in order to make a correct decision then the risk of error increases.

Long-term memory has both structure and performance characteristics. It can be considered as a knowledge base of symbol usage. It is convenient to represent it as a weighted multi-graph where a concept is a node and an edge is an association. The edge carries a weight representing how often it has occurred, and a label indicating the kind of association. The extent to which a concept is important has to do with the number of different edges leading to it. How well it is remembered depends on how often and how recently it has been seen.

For the purpose of cognitive complexity metrics, we are interested in unfamiliar word usage, as this will tax STM and lead to error. Determining what is unfamiliar is the principal challenge in assessing STM demands.

In computer programs, identifiers represent concepts. If the program is unfamiliar to the programmer, then the identifier density is a good predictor of error provided time is constrained [Klemola99]. If the object of comprehension is text, then the density of terms used to describe new information is a good indicator of comprehension error [Kintsch98].

In order to adjust for individual differences that experience brings, a strategy can be applied given our understanding of human thinking. When an audience possesses knowledge of many of the concepts being used in a text, those concepts need not be counted when predicting STM load. Thus we can tailor a comprehension metric such as the identifier density to the audience.

The weakest links (faintest details) between concepts are the first to be forgotten. A way of testing for the retainability of a text would be to build a network from the key terms used in the text. Then we would expect that information related to concepts that are weakly linked in the network would be forgotten most quickly and would therefore be the most likely cause of error.

We can consider the comprehension process to be one where text is chunked and matched with chunks stored in long-term memory. In order to measure the STM demands of reading a text it is necessary to determine the set of terms used to define the new information in the text. Once this has been done, then we can measure the density of such terms that should give an indication of its comprehensibility.

Once we have assessed a body of code or text for its cognitive complexity, we can use the measures in different ways. Now we can decide if the cognitive complexity is acceptable. If it is not then we could first assess the material for decomposability. If it is a program, then perhaps it could be further modularised. If it is a text, then perhaps it can be rewritten incorporating more analogies familiar to the audience. If it is not feasible to improve on the cognitive complexity as some domains are inherently complex, we should then allocate enough time for individuals to understand the material without error.

In a situation where performance depends on the comprehension of text, text that is seldom referred to would be more likely to contribute to errors. This may be a problem inherent to software specification, since the computer need only be told once of a concept, yet the human being will soon forget what has been only mentioned a single time.

Conclusion

One goal of cognitive complexity metrics is to be able to predict where the risk of error is high. This can be due to two factors. When STM is overloaded, errors occur. Also, when a weakly supported concept is important, the risk of error is raised. In this way, understandability can be measured as either too much information in a short time or not enough information over a longer period.

Future directions for my research include empirical testing of text comprehension exercises involving text of a technical nature. I also plan to look at how chunking is pattern matching and develop some metrics based on pattern matching.

Acknowledgements

I would like to acknowledge Brian Henderson-Sellers and Robert Rist for helpful suggestions, Rob Rist and Tom Osborne for stimulating discourse, Barry Jay for a discussion on graphs, and Walter Kintsch for validation on cognitive issues and encouragement. I would like to thank COTAR and UTS for funding attendance at conferences and Insearch Institute of Commerce for support. I would also like to thank Brian Henderson-Sellers for his encouragement.

References

[Besnard99] Denis Besnard, Mireille Bastien-Toniazzo, Expert error in trouble-shooting: an exploratory study in electronics. International Journal of Human-Computer Studies (1999) 50, 391-405, Academic Press

[Boehm-Davis96] Deborah A. Boehm-Davis, Jean E. Fox, Brian H. Phillips, Techniques for Exploring Program Comprehension, Empirical Studies of Programmers: Sixth Workshop, pp3-37, Ablex Publishing, 1996.

[Broadbent75] D. E. Broadbent, The magic number seven after fifteen years. In A. Kennedy & A. Wilkes (Eds.), Studies in long-term memory (pp. 3-18). London: Wiley

[Cant95] Cant, S., Jeffery, D.R. and Henderson-Sellers, B., 1995, A conceptual model of cognitive complexity of elements of the programming process, Inf. Software Technol., 37(7), 351-362

[Conway91] M. A. Conway, G. Cohen, and N. Stanhope, On the Very Long-Term Knowledge Acquired Through Formal Education: Twelve Years of Cognitive Psychology, Journal of Experimental Psychology: General, 120, 395-409, 1991, American Psychological Association.

[Ericsson95] K. Anders Ericsson and Walter Kintsch, Long-Term Working Memory, Psychological Review, 1995, Vol 102, No. 2, pp. 211-245.

[Henderson-Sellers96] Brian Henderson-Sellers, Object-Oriented Metrics: Measures of Complexity, Prentice Hall, 1996

[Kintsch98] Walter Kintsch, Comprehension: a paradigm for cognition, Cambridge University Press, 1998.

[Klemola99] Tuomas Klemola, Software Comprehension: Theory and Metrics, Masters thesis, Concordia University, 1999

[McCabe76] McCabe, T.J., 1976, Tom McCabe, A complexity measure, IEE Trans. Software Eng., 2(4), 308-320

[Reed96] Stephen K. Reed, Cognition, Brooks/Cole Publishing

Company, 1996