Human Cognition of Complex Thought Patterns

Harry M. Sneed*

Abstract

How much is our perception of the present determined by our experience of the past?

1 Theories on Human Cognition

The Scottish philosopher David Hume was one of the first to point out that our ability to distinguish between cause and effect is not all due to an objective perception of reality, but rather to build in notions of what we expect [1]. Hume, being a sceptic, wanted to take issue with the empiricist John Locke who believed humans could actually perceive objective reality and therefore determine causality [2]. Hume disputed this assertion, claiming instead that causality has no objective basis. It is not derived from our empirical perception. Instead it comes from reasoning and reasoning is subjective. In other words, Hume questioned our ability to see things as they are, an opinion later picked up and expounded upon by the modern philosopher John Kemeny in respect to modern science [3].

Immanuel Kant, the great German philosopher assumed Humes position and brought it to a logical conclusion in his famous essay on "Die Kritik der reinen Vernunft". According to Kant the so called natural order of the universe only exist in our minds. The human mind is the source of all order. Kant states "Since it is our thought which forms the order of reality, one can not say that our perception is determined by the objects we perceive, rather one must say that the objects we perceive are determined by our notion of how they should be " [4] This criticism of our objective reasoning power was in direct contradiction to the prevailing philosophy of the times as propagated by the French philosophers based on the teachings of Descartes and others [5].

The basic thesis of Kant is that there is no such thing as an objective perception of reality. All perception is formed to fit patterns of thought which exist a priori in our minds. Thus, all perception is by nature subjective. Assuming this to be true, the next question is where do these thought patterns come from, how do they originate?

*Institut für Wirtschaftsinformatik, University of Regensburg, Germany, e-mail: Harry.Sneed@T-Online.de

Kant answered this question on a religious ethical basis, claiming that mankind is endowed with God given thought patterns which allow him to perceive reality as it should be. Kant believed in natural law and in a universal reasoning power which allows us to distinguish between right and wrong. Today, biologists would claim that this reasoning power is inherited and behaviouralists would claim that it is formed by our environment. Most likely it is influenced by both the experiences of our ancestors as well as by our own early experiences, as pointed out by Bertrand Russel in his famous essay on the limits of human knowledge [6].

A good example of priori thought patterns is natural language. In describing reality man must assign what he perceives to words he knows. If there is no word, then the object may exist, but it can not be expressed. We only distinguish objects if we are able to assign them to a pre existing notion and pre existing notions are provided by our language. Therefore, our language determines how we perceive things. In case of simple objects, all languages are similar. There is usually a word which one attaches to the object. However, in case of complex objects, languages can differ extremely in how they assign meaning to these objects. They use different thought schemes, which makes it so difficult to translate such concepts. [7] It is here where environment comes into play. The development of natural languages is influenced by environment in which they evolve. It is evident that the language of Eskimos will differ from the language of natives of a tropical island. It is also evident that the language of an agricultural community where people are working in the fields will differ from the language of an industrial community where people are working in factories. Children inherit word and notions, i.e. thought patterns from their parents. As they grow up and have their own experiences these inherited thought patterns are enhanced or even overridden by new context dependent thought patterns formed by another environment. However, one is never free of the original thought patterns one inherits. They are part of our apriori cognitive process and influence everything we perceive. This is really what Kant refers to as our God given power of reasoning [8].

A modern advocate of Kant's theory of cognition is Paul Feyerabend. However, Feyerabend does not believe in any God given power of reasoning not even in a natural law. Instead, Feyerabend claims that natural laws are the result of human conventions which have evolved over time, i.e. something like Locke's social contract. Feyerabend vehemently objects to anything like an objective perception of reality. Nothing can really be proven, since all proofs are based on conventions and all conventions are man made. Thus, all cognition is relative to the prevailing mode of perception. There is no such thing as a single universal view of complex subject matter. Even the language of mathematics is a matter of convention. There are no self-evident truths, as demonstrated by the assertion disproved by the German mathematician Goldberg. Thus, not even mathematical propositions are self-evident, let alone computing algorithms or programming language constructs. The Feyerabend school is referred to as "Constructivism" [9].

The fact that cognition is a result of predetermined thought patterns does not necessarily imply that there is no influence of our impressions upon our cognitive abilities. The theory of evolution also applies to human perception. Our minds are conditioned by the environment in which we live. The ability of animals to perceive their environment is the result of a long evolutionary process and it differs from species to species. Therefore, bats and cats have totally different mental representations of their environment just as do men and dogs. Dogs have evolved to react to what they hear and smell. Men have evolved to react to what they think they see. In this respect our minds have been conditioned to see things as we would believe them to be. Plato compares human cognition with the perception of shadows on a wall. It is not reality we perceive, but only images of this reality which we have been conditioned to recognise [10].

2 Programmer Cognition of complex Programs

Now what might the theory of human cognition and the school of "constructivism" have to do with program comprehension? The answer to this question should be obvious. Programs are representations of complex thought patterns. These patterns are determined to a great extend by the languages or methods in which they are expressed. In fact it would be impossible for them to be expressed without a language. On the other hand, they can only be comprehended by minds conditioned to discern thought patterns in that language. Just as different animal species have developed different means of perceiving their environment, different programmer species have been conditioned to comprehend programs in different ways [11]. A COBOL Programmer will perceive a problem in a different mode than will a C Programmer. Ergo, there is no such thing as a universal approach to program comprehension as long as the minds of programmers are being conditioned by different languages and different methods. The thought patterns of programmers come from their experience in programming, especially from their early experiences. This is where the universities have a tremendous responsibility in implanting cognition patterns similar to the responsibility of mothers in instilling behavioural patterns.

A good example of this is the transition from procedurally oriented to object-oriented programs and designs. A programmer who has been conditioned to think in procedural terms will not been able to comprehend an object oriented program or program design no matter how well it is documented. The programmer will not be able to relate the constructs he perceives with the apriori constructs in his mind. The same of course will hold in the inverse direction. Minds conditioned to comprehend object-oriented constructs will have difficulty in comprehending procedural ones. This is why it is so difficult to get young college trained programmers to maintain old programs. They simply cannot relate to the concepts contained therein. These problems have been documented by Fayad and others [12].

Unfortunately it is still believed by many, especially by persons in management, that if programs are documented well enough, they should be comprehensible to anyone including themselves. This is one of the many myths introduced to the world by great simplifiers like James Martin who also advocated that programming can be done without programmers [13].

A complex Assembler program can only be comprehended by a person familiar with the semantics of Assembler. Even if the program is converted to a higher level language, it will still retain its Assembler semantics. It simply becomes an Assembler program with a COBOL or C syntax and it will take someone conditioned to think as an Assembler programmer to fully comprehend it. No amount of diagramming techniques will ever suffice to make the program comprehensible to anyone without pre existing Assembler thought patterns [14].

The same applies to higher level languages. A program written in a 4GL language such as NATURAL reflects the thought patterns determined by that language with constructs such as map driven and database driven loops, constructs which do not exist in standard 3GL languages. Thus, they can only be comprehended by minds conditioned to think in these terms, i.e. minds which have been exposed to 4GL languages.

The deluge of diverse programming languages and design methods has led to a highly fragmented programming community, where hardly anyone is able to comprehend the work of others. The tower of Babel is being reconstructed in the Software community. Documentation and annotation is of no help unless of course the subject is familiar with the solution domain. Even if one is familiar with the problem, he will still have difficulties comprehending the solution if he is not familiar with the language and method used, e.g. one conditioned to think procedurally will not be able to comprehend an object-oriented solution. Ergo, as long as languages and solution approaches are continually evolving, the program comprehension problem will increase regardless of the quality of documentation.

The contention here is that documentation, whether it be static or dynamic, tabular or graphic, is part of the solution domain. Therefore, it cannot really promote comprehension unless the subject has preconditioned thought patterns which can order and interpret the information contained in the documentation. Human minds must be conditioned to think in terms of the methods used to solve problems in order to understand the problem solutions. Thus program comprehension is really a matter of thought patterns installed in the minds of programmers and analysts by training and experience. These thought patterns are shaped by the languages and methods to which they have been exposed.

Von Mayrhauser states that "program comprehension uses existing knowledge to acquire new knowledge that ultimately meets the goals of a code cognition task. This process references both existing and newly acquired knowledge to build a mental model of the software under consideration" [15]. The contention here is that the existing knowledge, i.e. the apriori mental concepts, must be well above 50 % of the total knowledge required to understand complex systems. The maintainer must already have expectations as to how a system should be constructed and how it should behave in a given context. That implies that he has been exposed to both the solution and the problem domain.

The emphasis here is on exposure. Anyone who has ever dealt with programming knows that it is never learned passively by just talking or reading about it. Thought patterns can only be installed by actively applying them. This applies to natural languages as well as to programming languages. Thus, complex programs can only be comprehended by persons who have experienced the method and language with which the programs have been implemented. Documentation may help to locate and understand the role of program artifacts, but only if they are in the domain to which the subject has been conditioned to comprehend [16].

3 Conclusions to be drawn

What conclusions can be derived from these observations. The first is that we in the program comprehension community should not over estimate the role of documentation. It can be helpful, but it is no substitute for preconditioned knowledge. An experienced C++ programmer will always understand a complex C++ program better than an experienced COBOL programmer no matter how much documentation the latter has. Documentation can not make up for the lack of apriori thought patterns.

Another conclusion is that one will not understand what a program is doing unless one is familiar with the subject area. He may understand the solution but he will never understand the problem. Also here documentation is of limited help, since documentation assumes that the reader is familiar with the terminology and frames of reference used to describe that subject area. Thus someone unfamiliar with aerodynamics but experienced in C++ may understand how a C++ program for calculating the affects of wind currents is functioning but will never understand why. This lack of knowledge will be detrimental if he is called upon to change the algorithm.

The final conclusion is that there is no substitute for apriori knowledge when it comes to comprehending programs. The maintenance programmer should be familiar with both the problem area and the solution domain, in order to be effective. This required degree of familiarity can only be obtained by conditioning the mind through practical experience, i.e. to exposure to the subject area.

4 Actions to be taken

This being the case, there are several ways in which industry can improve the software maintenance situation. The first is through standardisation. There should be less languages and less methods and these should be accessible to a wider range of programmers. Proprietary languages should be strictly avoided if at all possible. The productivity gain made by using Oracle Forms or Power Builder in development is later negated by the problems of maintaining or reusing such solutions. UML - Unified Modelling Language - from the OMG is certainly a step in the right direction just as is CORBA-IDL and ANSI C++.

The second way is by retaining experienced personnel in software maintenance. Persons with knowledge of both the subject area and the solution space are invaluable assets to the organisation. They should be cultivated and valued. If they must move on to other work, there should be a long transition period for them to pass their work knowledge on to their successors.

A third way is by getting greater user involvement in the software maintenance process. If the programmers are not so familiar with the application domain, it is essential that they work together with an end user who has this knowledge and can pass it on. This also means that the user representative is able to help in locating errors and in assessing the impact of changes.

A fourth way is through continuity. By continually changing their languages and design methods user organisations only disrupt the learning process. An organisation should be extremely careful in the selection of a design method and programming language. Once it has committed itself, it should then stick to that language and design approach as long as possible so as to profit from existing thought patterns. Software Managers must learn like Odysseus to close their ears to the wails of the modern day sirens – the Software Case vendors – who would have them believe that all of their problems could be solved by introducing a new development technology.

A fifth and final way is through training. New maintenance programmers should be trained both in the subject area and in the languages and methods used by the organisation. This training should go beyond formal presentation. The subjects must have the opportunity to condition their minds through participation in exercises designed to install thought patterns. This may be expensive and time consuming, but it will be well worth it in improving maintenance productivity. Older programmers should even be given a year of absence to condition their minds to a new technology such as distributed objects.

In summary, the point presented in this paper is that program comprehension is determined more by mind conditioning than by any means of documentation or annotation. Program documentation and annotation is helpful, but only if the thought patterns represented in the documentation are familiar to the subjects. Knowledge may be extracted from software artifacts, but is will only be accessible to persons conditioned to recognise that kind of knowledge. Thus, the thought patterns used in the programs must exist apriori in the minds of the programmers in order for them to associate the two, i.e. what they think they see with what they think they know. This type of pattern watching is the essence of human cognition and program comprehension is after all just another cognition problem.

References

[1] Hume,D. : "An Enquiry Concerning Human Understanding" Open Court Publishing, La Salle, J II., 1946

[2] Locke, J. : "An Essay Concerning Human Understanding" Aldine Press Letchworth G.B., 1961

[3] Kemeny, J. : A Philosopher Looks at Science", van Nostrand, Princeton, 1959

[4] Kant, I. : "Die Kritik der reinen Vernunft" in Geschichte der Philosophie, Ed. Hans String, Bertelsmann Verlag, Stuttgart,1962

[5] Descartes, R. : "Discours de la methode" in History of Materialism, Ed. F. Lange, Simon and Schuster, New York, 1961

[6] Russel, B. : "Human Knowledge - its Scope and Limits", Simon and Schuster, New York, 1948

[7] Stevenson, C.L. : "Ethics and Human Language" Yale University Press, New Haren, 1944

[8] Kant, I. : "Metaphysik der Sitten" in Werke von I. Kant, Ed.E. Cassirer, Herter Verlag, Berlin, 1912

[9] Feyerabend, P. : "Against Method-outline of an anarchistic theory of Knowledge" Atlantic Highlands, Princeton, 1974

[10] Platon, G. : "Der Staat", Artemis Verlag, Zrich, 1960

[11] Dijkstra, E. : "On the Cruelty of really Teaching Computer Science" in Comm. of ACM. Vol. 32, No.12 Dec. 1989

[12] Fayad, M./Tsai, W./Fulghim, L. : "Transition to Object-oriented Software Development", Comm. of ACM. Vol.39, No. 2, Feb. 1996

[13] Martin, J. : "Application Development without Programmers", Prentice-Hall, Englewood cliffs, 1981

[14] Martin, J. : "Diagramming Standards for Analysts and Programmers", Prentice-Hall, Englewood cliffs, 1987

[15] von Mayrhausen, A./Vans, M. : "Program Comprehension during Software Maintenance and Evolution" in IEEE Computer, Aug. 1995

[16] van der Meulen, M./Hage,J. :"Human Factors in Software Maintenance" Report of ESPRIT Project MACS, Maastricht, 1992