Information Complexity Hypothesis: a Conceptual Framework for Reasoning on Pragmatics Issues

Vladimir E. Zyubin*
* Institute of Automation and Electrometry SB RAS/Novosibirsk State University, Novosibirsk, Russia, zyubin@iae.nsk.su

***Abstract*— This paper presents an overview of recent research on human factors in computer and information systems. In the paper, we discuss the cognitive processes, structure of human memory and human limits for processing information, and formulate hypothesis of information complexity. The result of this work is a set of domain-free principles, which creates a compact conceptual framework for reasoning on pragmatics issues. The principles are applicable to various stages of program lifecycle and design of new programming languages.**

I. Introduction

The questions of human factors, psychology and aesthetics in programming were raised by Andrey Ershov as far back as the beginning of the programming era [1], when profession of programmer was an elite one. These questions have no answers and stay aside from common computer science investigations hitherto.

The *status quo* in computer science shows that the community concentrates its efforts on the mathematical aspects of the human-computer system, while human aspects of programming are mostly disregarded. Some few examples of notable works on the topics of software psychology [2-4], and computer language pragmatics, e.g. [5, 6], are rather the exceptions that prove the rule. But, is programming just a bit manipulation as it looks like from computing mathematics' point of view? Is there any harm for computer if the program it executes consists of the *goto* statements [7]? Can we restrict our investigations to the computer insides only, if programming is a human activity? Can we discover the laws of programming, while we ignore a valuable part of the problem?

The present work is based on the presupposition that programming is a branch of human activity, where human beings create programs exclusively for human beings and use computers for it at a few stages of the process only. So computer science ought to take into consideration not only computer aspects of the process, but comprehensive research of the human-computer system also, where human factors are a valuable part at least. But as a matter of fact, human aspects become the most valuable part of programming when we touch on questions of requirements analysis, specification, design, coding (as a computer language usage), and maintenance.

Historically, computer science has its roots in mathematics, so it is considered as so called *exact science*. Students specialized in computer science have no courses on elements of psychology yet. On the other hand, psychology is considered rather as a part of the humanities and is not interested in the domain of strict formalisms.

The consequences of this tragedy are very painful. Let's just recall the Algol language project, when despite of great intellectual investments the language has collapsed under the weight of its complexity as well as the tower of Babel.

In the case any successful attempt to combine computer science and psychology will give powerful and useful conceptual means that can help us to avoid such a mess at least. The author strongly believes that computer science has no future without common familiarization of the psychology conceptual means and their assimilation in practice and education process.

In this paper, we discuss the cognitive processes, structure of human memory and human limits for processing information, formulate hypothesis of information complexity, and explore some well-known cases from the created outlook. The result of this work is a set of domain-free principles, which allows to reach information complexity reduction with new knowledge level of information complexity origins. The principles are applicable to various stages of program lifecycle and design of new programming languages.

II. Human Aspects of Programming

In computer science, the most familiar work from psychology domain is the old article "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information" by G.A. Miller [8]. The main idea of the paper is within the title: a human being has a limit on processing information, which is specified by the number seven. We have serious mental problems when we deal with semantically separated units, which number exceeds the threshold value. The magical number is widely cited in the computer literature, e.g. see [9], but the citations look like mechanical ones. Firstly, the authors use the exact number, while the original number is a "magical" one, or to be more precise, the threshold value is specified by range because it depends on some circumstances we will discuss below. Secondly, the article reports about valuable exception for the rule. The limit becomes far beyond the bounds of the range if the target group is familiar with the topic the units belong to. Thirdly, the experiments show that the limits can be overcome if the units additionally have semantically different perspectives. G.A. Miller writes 'The better know a little about much than know much about a little'.

Miller's law is very useful for cognitive process understanding, but the aforementioned nuances encourage us to look at the problem a bit deeper and try to draw more detailed picture by examination of the human memory structure [4].

Human memory consists of two main parts (although they have no strict boundary and rather reflects conceptual level than objects or some parts of human brain). These parts are ***short-terms memory*** and ***long-term memory*** [10]. Their features are show in the following table.

TABLE I.
FEATURES OF SHORT-TERM AND LONG-TERM MEMORIES

Type of memory	Labor intensity of load	Time of storage	Semantic capacity
Short-term	Negligible	Up to 30 sec	7 ± 2 associated entities
Long-term	Intense	Static storage	Practically unlimited

The short-term memory demands no effort to load, but limits operation time and quantity of the operands. In order to retain information in short-term memory, it must be constantly rehearsed, which requires effort. Humans experience a feeling of relief when information no longer needs to be rehearsed and can be forgotten. This relief, experienced at the completion of a task, is termed *closure*. The need for closure suggests that it would be preferable to work with small portions of a task at a time and to be able to release the information at completion of that portion. That problem is also known as *closure problem* [4]. The limits directly connected with *information complexity* concept, which can be interpreted as an extent of short-term memory loading.

On the other hand we have long-term memory that has the only limit. It demands hard efforts to load information into. The efforts assume manipulations within short-term memory. So the memories under question can effectively operate jointly only.

The discussed facts lead us to the following interesting conclusions that are very close to Greeno's mental model of problem solving [11].

III. INFORMATION COMPLEXITY HYPOTHESIS

Miller's law describes the limit for processing of unfamiliar information only. If the subject is well known then Miller's law does not work. Knowledge is just a long-term memory that is already loaded with the data. The process of loading the long-term memory is the education process.

Also, we can comply with Miller's law if the long-term memory has information that looks like the target entity. That trick is widely used in programming languages that implement so called artificial metaphors [12], i.e. transfer conceptual means from one specific domain to the target one. For example we can recall relay-ladder diagrams that use the metaphor of relay to program control algorithms. Despite of metaphoric artifacts (i.e., misleading analogies in the user's mind) and other inconveniences of the method, it is very powerful approach because of drastic reduction of mental efforts and time expenses for initial stage of studying.

The other approaches we implement in order to make Miller's law less strict are based on a decomposition of the original problem into weakly connected parts [13]. It can be obtained by:

- removal of inessential details,
- extraction of semantically encapsulated entities,
- decomposition into weakly-connected aspects,
- hierarchical construction.

It is necessary to note that the tricks lead to appearance of a new information structure, where any local mental operation keeps within Miller's law. Process of building such an information structure is also known as model building.

The difference between the original entity and its model is the following. The model consists of elements that have the bounded above number of semantic connections exclusively, while there are no *a priori* reasons for the original entity to be comply with Miller's law.

Also it is necessary to note that model building leads to uncertainty, but the new quality (called *information isolation*) is obtained.

If after elimination of inessential details from a target entity, we still can not build model that complies with Miller's law, we have to eliminate *conventionally inessential details*, i.e. to build an *aspectual model*. So target entity can have several models that structurally contradict each other.

The above-mentioned remarks lead to the following statements:

- Mental operations, which assume concurrent manipulation with a large number of semantic entities, are impossible for human being.
- Dynamic mental operations with a complex information entity demand a model building, which must guarantee compliance with Miller's law.
- pragmatics characteristics of both constructed model and the model description language we use ensue from interaction of the end user's short-term and long-term memories.

This set of statements has been called information complexity hypothesis (ICH).

ICH creates compact, but very powerful conceptual framework for reasoning about programming language features and other pragmatics issues. As it would be shown below, the ICH principles are applicable in various stages of program lifecycle and new programming language design.

IV. WELL-KNOWN FACTS FROM THE ICH OUTLOOK

A. Graphic and Textual Programming Languages

What language is better? Most discussions on the topic we can see around looks like holy wars. Graphic languages are called simple, intuitive, friendly, attractive, etc., etc. WYSIWYG and WIMP are a common approach for user interface design. But the experiments [5] show that superiority of the graphic is debatable. What is the matter?

Inquiring into the question shows that graphics have two very attractive features: plenty expressive means and simple usage of artificial metaphor. The expressive means of graphics are color, tint, size, shape, texture, position, orientation. Although the means are specialized (e.g.,

color, shape, texture are suited for substitution of the qualitative aspects only, and the others are suited solely for quantitative ones), they are very powerful because they allow to combine semantically different perspectives.

Artificial metaphors drastically reduce mental operations on the initial step of education because of extensive usage of the existent long-term memory contents. The side effects of graphics are metaphoric artifacts, uncertainty, difficulties with formal semantics of the languages, the only generating type of grammars, etc. [12, 13].

In contrast to graphics, textual languages provide consistency of external and internal (machine) data representations, easy coding standardization and modification of the textual notation (search, search and replace functions) and translator creation, and laconic statements. They allow to choose arbitrary metaphor, but demand high level of professional skill, or, in other words, specific long-term memory contents.

The two-stage approach for programming has been suggested in [13]. On the first stage, the main means for programming is graphic one. The second stage assumes a textual language. From the ICH perspective, the key advantages of the approach are the following. Graphic means reduce the pressing on the short-term memory when the programmer try to analyze the target task. It allows:

- to reduce mental loading with help of an artificial metaphor,
- to establish connections inside the team,
- to organize a constructive dialogue with the customers.

A non-zeroed level of uncertainty is a reasonable price for the comfortable work opportunity. The faults will have local nature and can be corrected during the second stage, when programmer *will have the long-term memory prepared at the first stage*.

The only problem is a seamless connection between these presentations. The *seamless problem* have to be solved individually, e.g. as it can be conducted for the Reflex language (a C-like language designed for control algorithms specification) [14, 15].

B. Aspect-oriented Programming

The aspect oriented programming (AOP) [16] is a style of programming that attempts to abstract out features common to many parts of the code beyond simple functional modules and thereby improve the quality of software. The goal of the AOP is to make it possible to deal with cross-cutting aspects of a system's behavior as separately as possible. AOP is just a kind of the ICH principles implementation, when the model building is achieved by aspectual decomposition. The authors of AOP state the following problems: Why do aspects contradict with each other? What makes one aspect primary over the others? Can we develop a clear understanding of the different "natural shapes" that different issues have?

From the ICH outlook we can easily answer the questions.

Aspectual decomposition is made by different metaphors (it is not obligatory the metaphors are artificial ones). That circumstance leads to a contradiction in model structures that can be eliminated by hierarchical connection between the aspects only, i.e. there is a main aspect that defines the global structure of the system and there are biased aspects with a local sense. So called "natural shapes" are just aspectual models that comply with Miller's law that can not reflect the target problem independently because of conventionally inessential details.

C. Other Examples and the Grand Challenge

From the ICH point of view, the following discussions could be very interesting:

- Is UML [17] a language or just a set of languages that are called diagrams?
- Why the C++ and Java programming languages resemble the C language?
- What is the secret of the C language that makes the leader of OOP Microsoft Corporation to use it for the operating systems?
- Why industrial automation has chosen means of the IEC 61131-3 standard [18], and ignore a large amount of alternative formal models?
- Can we tell general-purpose language from domain-specific ones? Are the OO-programming languages the real third generation languages? Aren't they domain-specific, although mainstream, ones?
- What mental problems the concepts such as polymorphism, inheritance, and information hiding solve?

But it seems the main topic we have to discuss at the presence is what language shall we use in order to program the multicore processors. It is a real grand challenge we face with. Lack of answer means we have been thrown back in the age of low-level programming when we have to take into consideration the platform topology. Moreover, it can lead to a collapse of the computer mainstream at all [19]. The only way to meet the challenge is to improve our understanding of programming and design the approach that reconciles the human aspects of programming with physical parallelism of the machine architecture.

V. Conclusion

Information Complexity Hypothesis created to enhance quality of current practice in programming has been introduced. Information Complexity Hypothesis consists of three principles that reflect natural human limits for processing information ensued from the short-term and long-term memories interaction.

Information Complexity Hypothesis provides compact, but very powerful conceptual framework for reasons about programming language features.

The author believes that the introduced conceptual means could be a significant value to programming language research and development. Coming to understand the natural criteria for estimation information processes can help to our efforts to develop various kinds of computer and information systems.

The author expects a number of exciting developments using the ICH principles over the next few years that help us to meet the present day challenges and enhance programming process quality.

References

[1] A. P. Ershov, "On human and aesthetic factors in programming," *Programmirovanie J.*, No 1, 1990

[2] J. Seo, and B. Shneiderman, “Knowledge discovery in high dimensional data: Case studies and a user survey for an information visualization tool,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, 2006

[3] B. Shneiderman, “Direct manipulation: A step beyond programming languages,” *IEEE Computer*, vol. 16, 1983

[4] B. Shneiderman, *Software psychology: human factors in computer and information systems.* Winthrop Publishers, Cambridge, MA, 1980

[5] T. R. G. Green, M. Petre, “When visual programs are harder to read than textual programs,” Human-Computer Interaction: Tasks and Organisation, Proc. ECCE-6 (6th European Conference on Cognitive Ergonomics). G. C. van der Veer, M. J. Tauber, S. Bagnarola and M. Antavolits, Eds. CUD: Rome, 1992

[6] K. N. Whitley, “Visual programming languages and the empirical evidence for and against,” *J. of Visual Languages and Computing*, vol. 8, No 1, 1997

[7] E. W. Dijkstra, “GOTO statement considered harmful”, *Communication of the ACM*, vol. 11, No 3, 1968

[8] G.A. Miller, “The magical number seven, plus or minus two: some limits on our capacity for processing information,” *Psychological Review J.*, vol. 63, No 2, 1956

[9] D. Decker at al., *Review Guidelines on Software Languages for Use in Nuclear Power Plant Safety Systems.* NRC, 1997

[10] K. A. Ericsson, W. Kintsch, “Long-term working memory,” *Psychological Review J.*, vol. 102, No 2, 1995

[11] J. G. Greeno, “The structure of memory and the process of problem solving,” *Contemporary Issues in Cognitive Psychology*, R. Solso, Ed.,1973, pp.23-45

[12] V. L. Averbukh, “Metaphors of visualization,” *Programmirovanie J.*, No 5, 2001

[13] V. E. Zyubin, “Text and graphics: what language does programmer need?”, *Open Systems J.*, No 1, 2004

[14] V. E. Zyubin, “Reflex language: a dialect of the C language for programmable logic controllers,” *Mechatronics*, No 12, 2006, pp. 31-36

[15] V. E. Zyubin, “Graphical and textual specifications for complex control algorithms: opposition and cooperation,” Proc. El-Pub2003 (8th Int. Conference on E-Publications), Novosibirsk, 2003

[16] G. Kizales at al., *Aspect-Oriented Programming*, Xerox Palo Alto Research Center, 1999

[17] C. Larman, *Applying UML and Patterns: An Introduction on Object-Oriented Analysis and Design and Interactive Development. 2nd ed.* Prentice Hall, 2001

[18] IEC 61131-3. *Programmable Controllers. Part 3: Programming Languages. 2nd ed.* International Electrotechnic Commission, 1998

[19] V. E. Zyubin, “Multicore Processors and Programming,” *Open Systems J.*, No 7-8, 2005