The Computer: A Human Brain Assistant

The current methodology for using computers to solve problems can be summarized as a generalization of Polya's four-step methodology [5] used in teaching mathematics, and consist of the following actions performed by the computer user:

1. Formulate the problem;
2. Construct an algorithm that solves the problem;
3. Program (code) the problem-solving algorithm into a program in the computer's language;
4. Execute the program obtained in (3) by providing it with data representing an instance of the problem to be solved;
5. Validate the result.

This methodology was used with the first computers to solve problems in engineering and mathematics. In the meantime, computers have evolved and become instruments used in solving problems from all aspects of human activity, with algorithms being executed by the computer, thus adding the step of programming. Unfortunately, current computer usage relies on software whose complexity increases proportionally with the number of computer application domains. Since the computer's application domains encompass all areas of human activity, software complexity threatens to destroy the computer itself [4]. This phenomenon leads to a contradiction between the tool's (computer's) complexity and the difficulty of using it (user). Therefore, to support current computing technology, it is necessary to develop a problem-solving methodology that uses

the computer as a brain tool for executing algorithms from all areas of modern life.

2. Limitations of the Current Methodology

The contradiction generated by using the computer is currently resolved by developing software tools that allow users to avoid using the computer language in the problem preparation process (programming). But the language of these software tools complicates the problem preparation process for the computer even more. Due to the spiral of knowledge, the number and complexity of new computer application domains increases exponentially with the successes of computer applications, thus leading to the exponential growth of software complexity [3]. In turn, this leads to an increased demand for professional expertise in using computers, thus posing significant educational challenges [2]. This contradiction can be eliminated by employing a computer usage methodology based on the user's natural language, similar to the methodology for using all tools developed by humans to improve their activity.

3. A New Computer Usage Methodology

Let's now try to resolve the fundamental contradiction of computer usage differently. To do this, let's assume that each domain of computer usage is equipped with a virtual machine characteristic of the domain, capable of computationally interpreting the domain's concepts. This means that each concept of the domain is characterized by a term of the natural language, which is associated with a computational model specifying the term's meaning. For example, a cell in biology is characterized by

the term "cell" associated with a data model, most often a graph, which expresses the term's meaning using terms for the cell's components, such as nucleus, membrane, genes, etc., each in turn specified by a natural language term and a data model explaining the term's meaning. In this way, a natural language sentence becomes associated with a data model specifying its meaning.

3.1 The Algorithmic Language of the Domain

Following the idea presented above, an expert in an application domain will use the computer to solve her problems by expressing problem solutions in terms of the domain's concepts. That is, the domain expert will communicate with her computer using the domain's natural language, called here the Domain Algorithmic Language (DAL). Through school education, domain experts integrate domain concepts and their meanings into their students' brains. However, only the term is used in domain language sentences, and the phrase's meaning remains an abstraction. Thus, the domain's natural language is part of the natural language, which is ambiguous and therefore difficult to use as a computational object. However, by associating domain language terms with data models representing their meanings and making these models visible, the domain language's ambiguity is resolved, and DAL can be easily manipulated with the computer. For example, the concept "cell" in biology is not confused with the concept "cell" in telephony or automata theory unless the user is not a domain expert. But in this case, the used concept has the meaning of the user's domain (student and/or expert).

The hypothesis we use here is that during the school learning process, domain concepts can be memorized both in the student's brain, as natural language terms, and on an information support specific to the domain, as a computational representation of the term's meaning. For example, Pythagoras' theorem in geometry is recorded on the information support as a linguistic expression associated with its proof, recorded on the same information support, as follows:

Begin Concept:

Term: Pythagorean Theorem;

Statement: In a right triangle $\triangle ABC$, where A is the right angle, BC is the hypotenuse, and AB and AC are the legs, the following equality holds: $AB^2 + AC^2 = BC^2$.

Proof: By construction. We draw the altitude from A, thus obtaining the similar triangles $\triangle ABD$, $\triangle ADC$, and $\triangle ABC$. From the similarity of $\triangle ABC$ with $\triangle ABD$, we get $BC/AB = AB/BD$, hence $AB^2 = BC \times BD$. Similarly, from the similarity of $\triangle ABC$ with $\triangle ADC$, we get $AC^2 = BC \times DC$. Summing $AB^2 + AC^2 = BC \times BD + BC \times DC$. But $BD + DC = BC$, so factoring BC in the previous equality, we get $AB^2 + AC^2 = BC \times (BD + DC)$, hence the result $AB^2 + AC^2 = BC^2$.

End Concept.

In the article [7], we called this learning style the Computational Emancipation of the Problem Domain (CEAD). CEAD-ing a domain can be carried out during school education in collaboration with domain and computer science experts. The domain education

expert provides the tuple (term, meaning), and the computer science expert provides the model and tools for recording the thus-learned concepts. The primitive concepts of DAL can be manually recorded on an information support called the Domain Ontology (OD). Concepts generated in the learning and usage process are expressed in terms of already recorded (learned) concepts and thus can be automatically represented in the Domain Ontology during the learning and usage process. Consequently, the DAL of a domain dynamically grows with the education and usage process. Software tools, such as XML, RDF, URI, SPARQL, WSDL, etc., currently used for advanced studies in the Semantic Web, can be effectively used for this purpose during the learning and usage process.

Consequently, in any field of expertise, during problem-solving, the computer can be used as a brain tool through a natural communication process between the computer and its user. Thus, software complexity is removed from the computer usage process, becoming an activity executed by computer science experts. The learning process is no longer complicated by mixing application domain concepts with computer science concepts.

3.2 Problem Solving Using the New Methodology

To solve a problem in the application domain D, we use a methodology similar to the current one. But now, we assume that domain D is computationally emancipated and its ontology is OD(D). Additionally, domain D is provided with a virtual machine whose instructions are terms denoting the concepts of domain D, and the calculation represented by these instructions is specified by the data model associated with the concept, stored in the

domain ontology D, and accessible on the Internet as a web service. The virtual machine dedicated to domain D (DDVM) is equipped with an abstract processor with a register called Concept Counter (CC) and operates according to the following program:

CC := Web Service (OD(D)};

While CC is not End

{

Execute Web Service indicated by CC;

CC := Next(CC));

}

Validate the result

Next(CC) is a sofware tool that searches OD(D) for the next concept stored in CC and executes the identified calculation process. This algorithm is similar to the program executed by a real computer, with the difference that:

• The program is an expression of the DAL language, i.e., a phrase of the domain's natural language;

• The computer memory storing instructions and data is the domain ontology OD(D), containing the user's knowledge.

If domain D is real computer programming, then DAL is a regular programming language, and the domain phrase is the program

representing the problem solution in the real computer's memory. Thus, for using the computer to solve domain D problems, the user performs the following actions:

• Describe an algorithm that solves the problem. This algorithm is a phrase of the domain's natural language;

• Make the CC register of the DDVM virtual machine contain the address in OD(D) of the first concept of this phrase;

• The rest is automatically executed by DDVM(D).

Notice that programming has disappeared, naturally along with its complications. We demonstrated this problem-solving model with solving equations in the domain of Algebra [1, 8]. Among the new software tools required by this computer usage methodology are:

• A software description language (Software Architecture Description Language, SADL) [9]. SADL is a universal language developed as an XML application whose primitive constructs are tuples of the form

⟨DALterm⟩ Web-Service ⟨/DALterm⟩.

• A SADL interpreter that executes SADL algorithms (programs) on a real computer;

• A translator that transforms DAL phrases into SADL algorithms;

The computer user develops problem-solving algorithms using the domain's algorithmic language, invokes the DAL translator, which translates DAL phrases into SADL algorithms, which are then executed by the SADL interpreter on the Internet using

concrete computers that are available. Software tools involved in this activity are developed and implemented by domain experts in collaboration with computer science experts.

4. Future Research

The research reported here was initiated many years ago, with *its* embryo in Chapters 6 and 7 of the book [10]. The evolution of software over the past 10 years validates the "karma" of this research: to support the evolution of the computer, we must develop software that facilitates using the computer with the user's natural language. Until now, software technology has been dedicated to the computer's technology. This research shows how to construct software dedicated to the computer user. All research and development reported here can only be carried out through collaboration between domain experts and computer science experts. Therefore, research is required in two new directions:

• Educating domain experts on how to computationally emancipate the domain of their expertise and record learned concepts in the domain's ontology;

• Designing and implementing a general software support for using the computer to solve problems by means of the user's natural language.

References

[1] C. K. Bui. *An evolutional domain oriented approach to problem solving based on web service composition*. PhD thesis, The university of Iowa, Department of Computer Science, Iowa City, IA 52242, May 2013.

[2] P. Denning. The profession of IT. *Communications of the ACM*, 58(9):34–36, 2015.

[3] P. Horn. Autonomic computing: IBM's perspective on the state of the information technology. http://www.research.ibm.com/autonomic/manifesto, 2001.

[4] J. Markoff. Killing the computer to save it. *ACM TechNews*, October(31), 2012.

[5] G. Polya. *How To Solve It*. Princeton University Press, second edition edition, 1957.

[6] T. Rus. Computer integration within problem solving process. In *Proceedings of RoEduNet 11-th International Conference*, Sinaia, Rumania, 2013.

[7] T. Rus. *Computer-Based Problem Solving Process*. World Scientific, 2015.

[8] T. Rus and C. Bui. Software development for non-expert computer users. In *Proceedings of the International Conference on Cloud Computing and Virtualization*, pages 200–207, Management University, Singapore, 3-5 May 2010.

[9] T. Rus and D. Curtis. Application driven software development. In *International Conference on Software Engineering Advances, Proceedings*, page 32, Tahiti, 2006.

[10]. T.Rus. Mechansime Formale Pentru Specificarea Limbajelor, Editie revizuita si adaogita. Presa Universitara Cljeana 2023.