Programming without Programming Languages. (New Graphic Poliglot Concept)

I.V. Velbitsky
Ukraine, ivelbit@gmail.com

Annotation: For the first time since 1947 it is proposed to use a simpler and mathematically more rigorous concept of programming with the graphs loaded only through horizontal arcs characters, expression and functions of elementary mathematics. Such graph is a polyglot, it has ISO 8631/1989 standard, and is the only one that can be used effectively throughout the life cycle of program design and use. Conventional programming languages are not needed. The new concept has 100+ times better characteristics with regard to visualization, simplicity, and compactness, as well as the speed of entering into the computer. Processes of error-free design of algorithms, programs, and data structures, evidence of their correctness, self-documenting and documenting of motivation of decisions made are significantly simplified, improved and accelerated. The resulting programs are more effective on the memory footprint and execution time. The larger and the more complex is the program project, the greater is the effect of applying the new concept. The new concept is so simple that it makes it possible to program for ANYONE, not just for programmers. This article describes the history of developing and proving the new concept, its description, advantages, implemented graphical programming environment, and perspectives for its application.

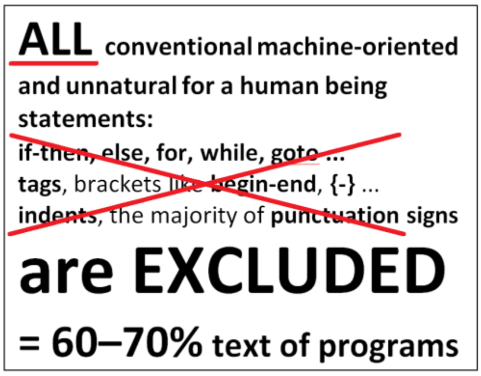
Keywords: **- Graphical programming; Graphs loaded through arcs; Logical and abstract programming schemes; Polyglot, 3D and multi-dimensional programming; Color, drawing and network schedules in programming; Self-documentation and documenting of decision motivation; Evidence and error-free style of programming; cyber security, Big Data**

1. INTRODUCTION

We introduced the term “programming technology” back in the 1960s, as an intuitive comprehension of the fact that not only the language (at the time it was Algol-60) is important, but also how to use it properly and correctly, *easily* and *just*, [1,2]. At present, technology is included in all IT curricula. But...programming remains complicated and expensive, because the programming concept existing since 1947 remained unchanged and became outdated.

Development of programming in graphs *) started in the 70-s when, in the course of 15 years, large-scale control systems for all major missiles of former Soviet Union [3] were built and increasing awareness of the importance of formal documenting of their development process to facilitate rapid introduction of permanent improvements and bug fixes on the one hand and with the works of Dijkstra [4] on the other, who was the first one to demonstrate the lack of mathematical rigor and redundancy of the conventional programming. As a result, for the first time since 1947 a new, very simple concept of graphical programming, which is uniform for the algorithms, software, data and network diagrams for their design throughout the lifecycle of program development and operation was completely formed by 2016. Its comprehensive analysis, tentative implementation (!) and operation of the new graphical programming environment (GPE) were carried out [5-8].

***) In the graph theory in mathematics there are only TWO equivalent types of graphs that can be used in programming. Those are graphs loaded through the vertices (for example, the well-known block diagrams, Moore machines) and graphs loaded through the arcs (for example, the well-known network diagrams, Mealy machines).**



ALL conventional machine-oriented and unnatural for a human being statements:

~~if-then, else, for, while, goto ...~~

~~tags, brackets like begin-end, {-} ...~~

~~indents, the majority of punctuation signs~~

are EXCLUDED

= 60–70% text of programs

Fig. 1. Programming concept existing since 1947

In the current programming concept all conventional machine-oriented operators, which are unnatural for a human, like **if-then, for, goto ..., tags,** square brackets like **begin-end, {-} ...** , indents, most of the punctuation marks, etc. **(all of them!) are excluded** in the new concept (Fig. 1). They became obsolete. There are too many of them – *more than a half* in the existing program texts. They are complicated, empirical (not strictly defined), have low capacity, and provide a primitive handicraft technology in programming. People put too much effort to *neutralize* those drawbacks by means of creating many new special languages, methods and programming environments that "supposedly make things easier," but, in fact, make programming more complex, confusing and not available for everyone.

For the first time the new concept proposes not to write but to draw programs in graphs loaded through horizontal arcs with characters, expressions and functions of elementary mathematics. It is many times easier, faster, better visualized and more compact compared to writing a program using the existing programming concept (using special text languages). It uses the same keyboard, mouse or touch screen and ensures that the resulting drawing complies with ISO 8631/1989 [5]. Such graphs and records on them are polyglots, which are understandable to all without definitions. They are in line with the known, strict mathematical principles of representation and processing of information, which approximate to the principles of operation of the human brain. Therefore, all of the special languages, such as FORTRAN, C ++, JAVA, and Python become low-capacity and unnecessary, though their libraries are retained. Any program from the library in any language can be automatically transferred to the new graphic form of presentation. It results in: 1) simplified transition to the new concept, 2) programming experience accumulated since 1947 will be preserved and multiplied.

The new concept is universal. It allows writing any algorithm and any program by any method known from structural to deep intellectual analysis, providing cyber security and using ready libraries, which can be combined into a single graphical hyper-library. For the first time the new concept introduces into programming the evidentiary and error-free style, self-documenting and documenting of motivation of decisions. The new concept allows each person drawing easily and visually the logical scheme of any problem and solving it manually or with a computer. The principles of the new concept are so simple and natural to a man, that for the first time programming becomes available to everyone (and not just for programmers) and it also becomes an element of universal literacy and culture of the society.

2. ESSENCE OF THE NEW PROGRAMMING CONCEPT

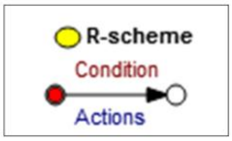


Fig.2. Axiom of the new programming concept.

The new concept uses only *one* horizontal arc of the graph, the so-called *Logical R-scheme or axiom*, for recording of any algorithms, data and programs, Fig. 2, where the *Condition* is written on the top of the arc, and the *Actions* to be performed if the condition is "true" are written at the bottom. Any language, including English, Russian, Chinese, etc., characters, expression and functions of the classical mathematics, and relevant fragments of any programming language with, in the first place, the function of reference to their libraries can be used for writing the Condition and Actions in one or several lines, Fig. 3.

Any number of arcs directed to the right and/or to the left may radiate from one graph node. For recording of the "loop" graphs either an empty arc is used indicating an unconditional transition

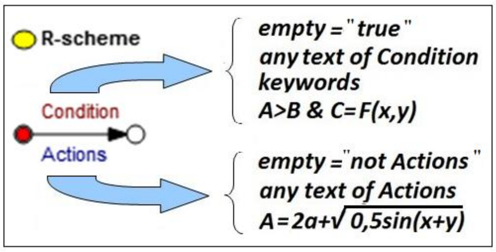


Fig. 3. Recording of Conditions and Actions on the arc in one or several rows.

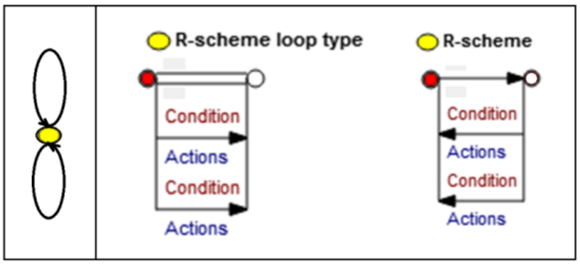
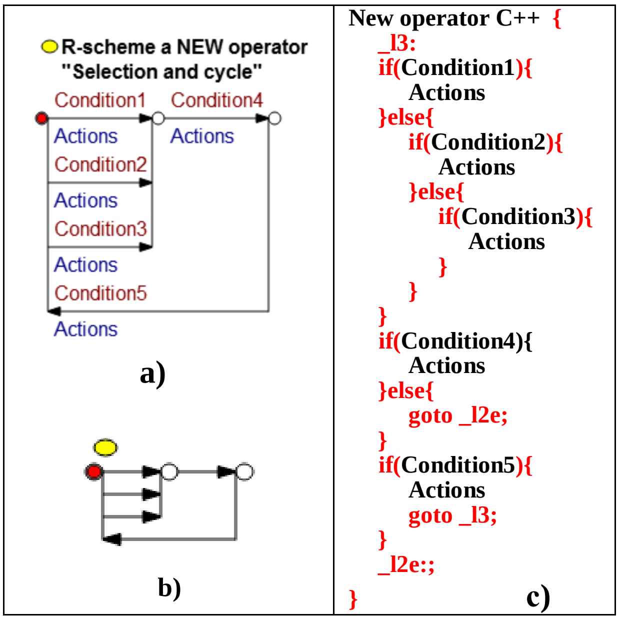


Рис. 4. R-схемы графов типа «петли».

without performing any action, or a special double horizontal arc of the "equal sign" between the nodes, Fig. 4. The arcs radiating from any node are analyzed top down sequentially. The first arc is executed, whose Condition is "true". At the same time all Actions under this arc are carried out and transition is made through the arc's arrow to the new state of the graph (algorithm or program). If all the arcs radiating from the node have "false" Conditions, the transition is made.



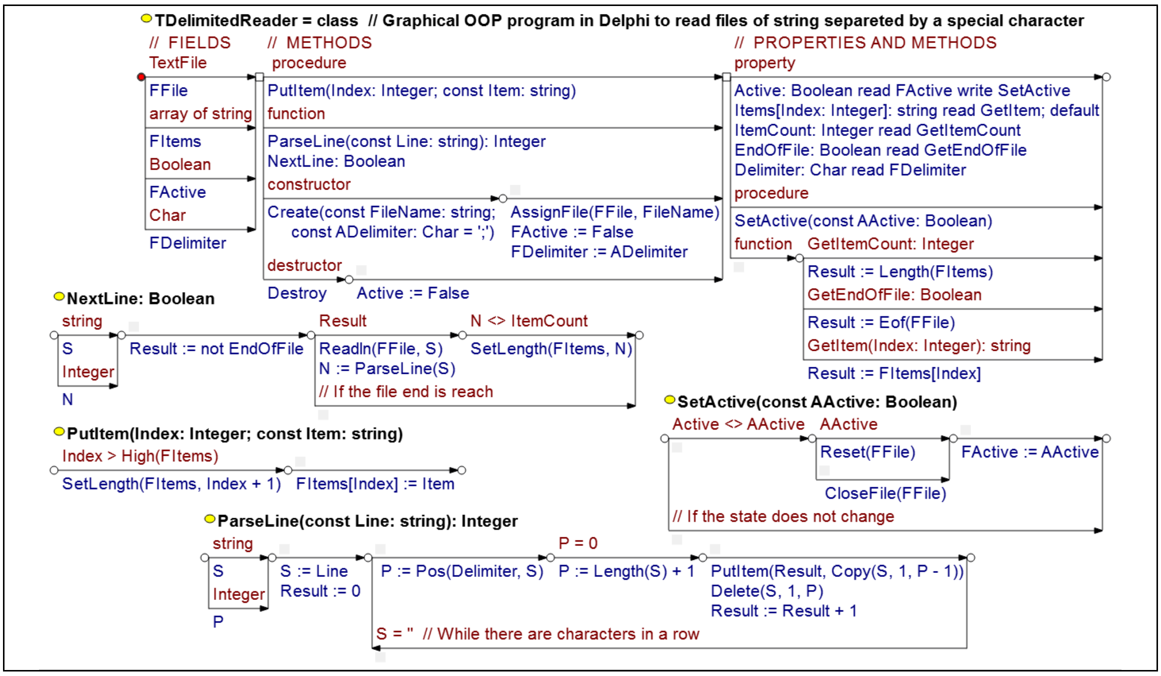
```
New operator C++ {
  _l3:
  if(Condition1){
     Actions
  }else{
     if(Condition2){
        Actions
     }else{
        if(Condition3){
           Actions
        }
     }
  }
  if(Condition4){
     Actions
  }else{
     goto _l2e;
  }
  if(Condition5){
     Actions
     goto _l3;
  }
  _l2e:;
}
```

Fig. 5. Recording of the new statement of "Selection and Cycle" in RR* and C++

Fig. 5 shows an example of defining the new statement: "Of 3 (can be any number) Conditions and Cycle". Fig. 5a shows its R-scheme (14 lines), Fig. 5b – its *abstract* R*-scheme (5 lines) or record of the R-scheme (algorithm, program) without the implementation details, with no records on the arcs, and Fig. 5c – the equivalent conventional recording in C++ (24 lines) where the *superfluous* symbols for the R-scheme in Fig. 5a are shown in red color. The number of such extra characters on Fig. 5c is 77 or, including the indents and line feed, which are also redundant, **- 140** (**= 62%**). The record of R-scheme is **2** times (=24/14), and R*-scheme is **5** times (=24/5) more compact compared to their recording in C++, or R=2 and R*=5, respectively. Entering of one (out of 10) statement in C++ (Fig. 5c) takes an average of 14 (= 140/10) key strokes, and entering of the scheme arc in Fig.5a takes **0.8** (=4/5) strokes, which is **18** (=14/0.8) times less and faster, eR=18. The order of figures is stored in the appropriate translator of R-schemes in the GPE operator **while** at the output. The maximum currently known program RR*-schemes have the following characteristics: R=**133**, R*=**400** and eR=**578**. This means that their compactness is 133, 400 times higher (they have 133, 400 times less lines), and the speed of entering into a computer is 578 times higher compared to the conventional concept of recording of those programs in C++.

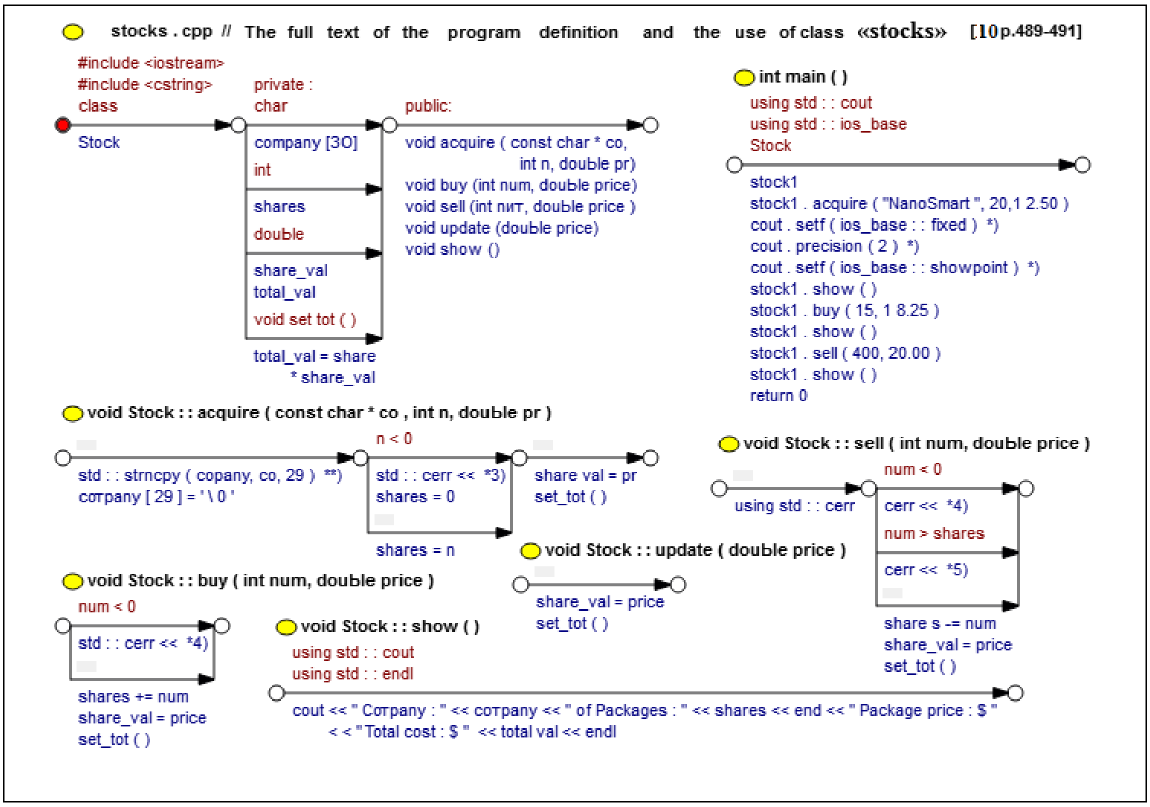
Entering of each horizontal arc of the graph is done with one mouse or keyboard click or finger touch on the touch screen. Multiple new horizontal arcs can be entered with a single click. Vertical arcs and graph nodes are not entered into computer and are drawn automatically in GPE, which greatly simplifies and speeds up entering of graphic statements (arcs) as compared with entering the existing statements in conventional programming languages.

Such graph is assigned a name written on the top near the yellow ellipse, Fig. 2-7. In mathematics that name matches the name of function defined graphically in the new programming concept. The name can be with or without parameters, see. Fig. 6,7 of real graphics software [9,10]. *The Program* in the new concept is set with any number of such graphs interlinked by the name. The first graph (R-scheme) of such a program is called *an axiom*. The first left apex of program's axiom is always highlighted in red.



```
TDelimitedReader = class  // Graphical OOP program in Delphi to read files of string separeted by a special character
// FIELDS          // METHODS                                               // PROPERTIES AND METHODS
TextFile           procedure                                                property
FFile              PutItem(Index: Integer; const Item: string)             Active: Boolean read FActive write SetActive
array of string    function                                                Items[Index: Integer]: string read GetItem; default
FItems             ParseLine(const Line: string): Integer                  ItemCount: Integer read GetItemCount
Boolean            NextLine: Boolean                                       EndOfFile: Boolean read GetEndOfFile
FActive            constructor                                             Delimiter: Char read FDelimiter
Char               Create(const FileName: string;   AssignFile(FFile, FileName)   procedure
FDelimiter             const ADelimiter: Char = ';')  FActive := False      SetActive(const AActive: Boolean)
                                                    FDelimiter := ADelimiter    function   GetItemCount: Integer
                   destructor                                                         Result := Length(FItems)
                   Destroy    Active := False                                         GetEndOfFile: Boolean
NextLine: Boolean                                                                     Result := Eof(FFile)
string             Result          N <> ItemCount                                     GetItem(Index: Integer): string
S          Result := not EndOfFile   Readln(FFile, S)   SetLength(FItems, N)          Result := FItems[Index]
Integer                            N := ParseLine(S)
N                                  // If the file end is reach
                                                              SetActive(const AActive: Boolean)
PutItem(Index: Integer; const Item: string)                   Active <> AActive   AActive
Index > High(FItems)                                                     Reset(FFile)   FActive := AActive
SetLength(FItems, Index + 1)   FItems[Index] := Item                     CloseFile(FFile)
                                                              // If the state does not change
ParseLine(const Line: string): Integer
string                                         P = 0
S         S := Line    P := Pos(Delimiter, S)  P := Length(S) + 1   PutItem(Result, Copy(S, 1, P - 1))
Integer   Result := 0                                               Delete(S, 1, P)
P                                                                   Result := Result + 1
                       S = ''  // While there are characters in a row
```

Fig. 6. R-scheme of OOP program is 7 times **more compact** than the conventional recording of that program in Delphi [9].



```
stocks . cpp // The full text of the program definition and the use of class «stocks» [10p.489-491]
#include <iostream>                                                      int main ( )
#include <cstring>     private :                                         using std : : cout
class                  char              public:                         using std : : ios_base
Stock                  company [30]      void acquire ( const char * co,  Stock
                       int                             int n, double pr)  stock1
                       shares            void buy (int num, double price) stock1 . acquire ( "NanoSmart ", 20,1 2.50 )
                       double            void sell (int nит, double price ) cout . setf ( ios_base : : fixed ) *)
                       share_val         void update (double price)       cout . precision ( 2 ) *)
                       total_val         void show ()                     cout . setf ( ios_base : : showpoint ) *)
                       void set tot ( )                                   stock1 . show ( )
                       total_val = share                                  stock1 . buy ( 15, 1 8.25 )
                          * share_val                                     stock1 . show ( )
                                                                          stock1 . sell ( 400, 20.00 )
                                                                          stock1 . show ( )
                                                                          return 0
void Stock : : acquire ( const char * co , int n, double pr )
                                              n < 0                      void Stock : : sell ( int num, double price )
std : : strncpy ( copany, co, 29 )  **)   std : : cerr << *3)   share val = pr               num < 0
corpany [ 29 ] = '\ 0 '                   shares = 0            set_tot ( )   using std : : cerr   cerr << *4)
                                                                                                   num > shares
                                          shares = n      void Stock : : update ( double price )   cerr << *5)
void Stock : : buy ( int num, double price )
num < 0                                                   share_val = price                        share s -= num
std : : cerr << *4)      void Stock : : show ( )          set_tot ( )                              share_val = price
                         using std : : cout                                                        set_tot ( )
shares += num            using std : : endl
share_val = price        cout << " Corpany : " << corpany << " of Packages : " << shares << end << " Package price : $ "
set_tot ( )                 < < "Total cost : $ "  << total val << endl
```

WHERE

*) // format #. ## with 2 symbols after dot and display of zeros in the result
**) // truncate co to place into company
*3) "Number of packets cannot be negative; for "<<" is set to 0. \ n"
4) "Number of purchased (sold) packets cannot be negative." 6)
5) "You cannot sell more than you have!" 6)
*6) << "Transaction cancelled. \n"

Fig. 7. R-scheme of OOP program is **4.1** times **more compact** than the conventional recording of that program in C ++, R=**4.1** , R* = **32**.

Fig. 6 shows a graphical OOP- program in Delphi [9], and Fig. 7 – another OOP-program in C++ [10, page 489]. This means that in those programs the Conditions and Actions are written in Delphi and C++ respectively. In Figures 6 and 7 the first R-scheme (an axiom with the red dot) specifies the formula and architecture for determining the appropriate OPP, namely, sets the Data (FIELDS for Delphi and privat for C++) and related Methods. Then the definition is given for four methods for Delphi and 5+1 public-functions for C++, respectively. The graphic OOP-program in C++ given in Fig. 7 illustrates the method for recording comments and macro-definitions after the adverb **WHERE**, which for many centuries has been used in mathematics and therefore is more natural, compact and effective compared with their recording in the conventional programming concept.

Fig. 8, 9 shows the Abstract R*-schemes of programs in Fig. 6, 7, respectively. An *Abstract* R*-scheme is a record of the corresponding R-scheme without implementation details, with no records on the arcs, e.i. it is as compact as possible containing only the image of mathematical logic (essence) of the program. This is a new theoretical concept in programming, which has a great future for compact recording, *analysis* and *synthesis(optimization)* of programs. The logical – R and Abstract – R* schemes of the new concept represent a new Unified graphical shell (standard, template) to record programs and program libraries in all languages. This is NOT possible in the conventional programming. It opens up new opportunities for integrating the intellect accumulated in libraries of all the existing programming languages and for their evolutionary development together with the man and the corresponding development team.

Both examples in the Fig. 6-9 are interesting because they illustrate the organization of OOP in a variety of the most advanced OOP-oriented languages – C++ and Delphi. Both examples are written by professional programmers, who used all the subtleties and features of those languages for recording of the programs. Graphic recording of these programs demonstrates the effectiveness (advantage) of RR*-schemes in terms of simplicity, clarity, visualization, compactness and self-documenting of program graphs. This advantage as well as others we have confirmed in all the examples of 1184 pages of the Textbook [10]. This suggests the overall effectiveness of the proposed new graphical programming polyglot concept compared to the conventional programming concept, which remains unchanged since 1947. The new graphic concept also allows *an objective* (!) and quantitative comparing of two new languages and answering the question "what language is better?", for example, Delphi, Java or C++. In addition, let us compare quantitative characteristics of an abstract R-program of 10x10=100 arcs in Fig.9, which is more compact compared to C++ R=**15** and R*= **45** times and entered into the computer eR=**138** times faster, and R-scheme of 100x100 arcs has R = **133**, R* = **400** and eR = **578**. What is important, the new concept is a *polyglot,* because it uses international mathematics and graphics, which are better perceived by man than the existing artificial programming languages.

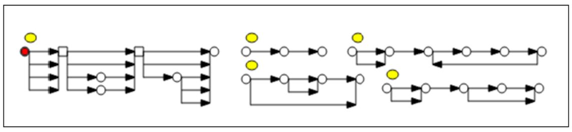


Fig. 8. R*-scheme of program recording without its implementation details is **36** times more compact than its recording in Delphi.

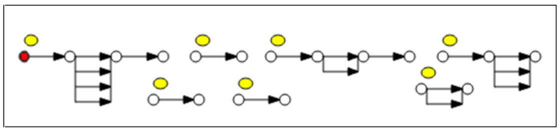


Fig. 9. R*-scheme of program recording without its implementation details is **32** times more compact than its recording in C++.

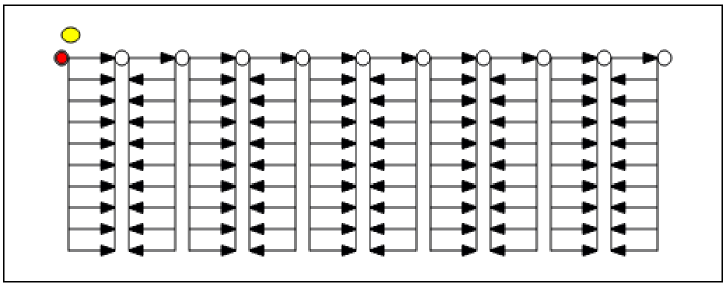


Fig. 10. Entering of R-scheme (program) from 10x10=100 arcs: R=**15** , R*=**45** , eR=**138** versus C++.

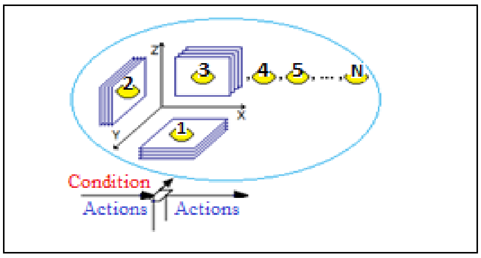


Fig. 11. Organization of 3D programming and documenting of motivation of the decisions made.

To define data structures and classes, OOP techniques, setting functions, etc. the new concept uses *key* words such as: **int, array, class, function**, **#include <iostream> ... ,** which are recorded on the top of the arc and are always true, Fig. 6,7. These arcs generally *do not have alternatives*, they are independent of each other and can be run *in parallel* in accordance with the keyword on the arc. The information they use is, as a rule, auxiliary and serves to neutralize the disadvantages of conventional programming concepts and to simplify the processes of translation in it. In graphical programming this information can be simply obtained from the context of R-scheme and, therefore, can eventually be excluded from the graphical programming in whole or in part, as it was done in mathematics. It is still preserved on the Fig. 6-7. The principle of preservation is visually described in Fig. 7 and in the conventional recording of this information in the C++ given below [10 p. 489]:

```
// stocks.cpp – full program text
#include <iostrem>
#include <cstring>
class Stock // class declaration
{
private :
      char company[30];
      int shares;
      double share_val;
      double total_val;
      void settot () { total_val = shares * share_val;
public :
      void acquire (const char * co, int n, double pr);
      void buy (int num, double price);
      void sell ( int num, double price);
      void update (double price);
      void show ();
};             R=(17/13)  R*=(17/4)=4,3
```

Complex data structures may be set directly in the R-scheme graphics, Fig. 12. In this case, the logic element or data syntax will be recorded on the top of the arc, and corresponding semantics – at its bottom [2]. RR*-schemes can be effectively used for recording algorithms "from the data", such as translators.

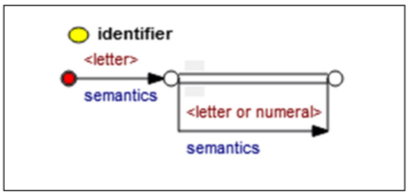


Fig.12 . Setting of data structures.

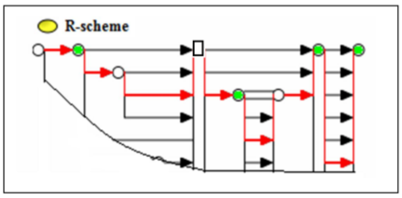
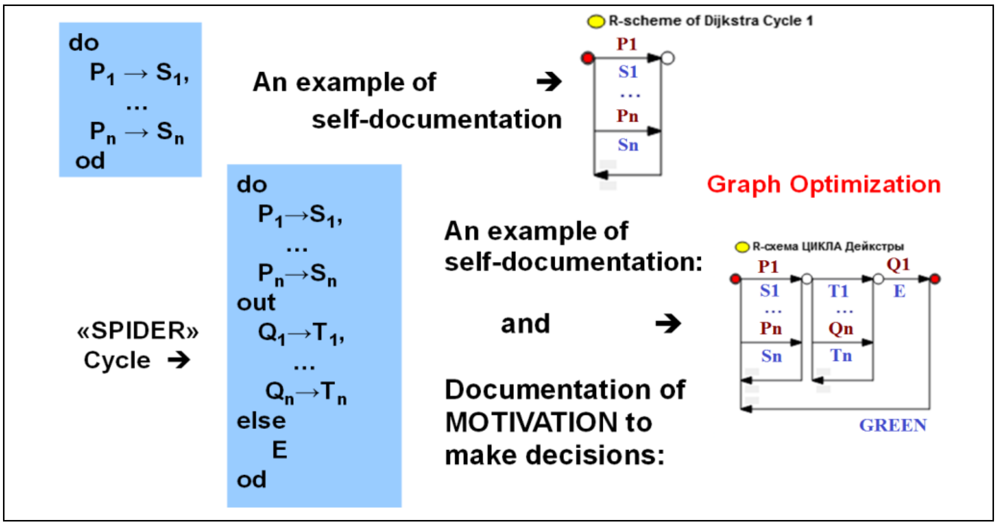


Fig.13 . Use of color to define nodes and arcs

The graph's vertices and arcs may be of various *configuration* and *color,* Fig. 4-14. Fig. 6, 8 demonstrates arcs marked with square vertex that can be executed in parallel, Fig.11 depicts a vertex in the shape of parallelogram explaining the principle of 3D-multidimensional performing of graphic programs, and the rectangular vertex (of float type Fig. 13), for example, can store the number of references to it and have a sensitivity threshold. Fig. 4,12,16,20 uses the arc configuration in the shape of "equal" sign to visualize the image of a "loop" graph, etc. Vertex color is widely used to implement of traffic light in the program, Fig. 13,14, and color of arcs – for visual notation, for example, of routes for automatic generation of program tests, Fig. 13, etc. The color of vertices and arcs can easily be set (specified) in the program, for example, function **GREEN=green (...),** Fig. 14. This is a new feature, which is not available in the conventional programming, or which is hard to model by conventional means. For the first time it becomes possible to naturally and simply document motivation of the decisions taken, program various project pieces in parallel, automatically generate a functionally complete set of tests, and debug programs simultaneously with their development, etc. Fig, 14 demonstrates another new possibility (lacking in the conventional programming) of *automatic optimization of* R*-schemes by their design procedure, which is suboptimal but resulting from a simpler R-scheme: « **SPIDER Cycle ➔**».



```
do
  P1 → S1,
    ...
  Pn → Sn
od
```

An example of self-documentation ➔

```
do
  P1→S1,
    ...
  Pn→Sn
out
  Q1→T1,
    ...
  Qn→Tn
else
    E
od
```

«SPIDER» Cycle ➔

An example of self-documentation: and ➔

Documentation of MOTIVATION to make decisions:

Graph Optimization

Fig. 14. Protected commands of Dijkstra machines.

It is worth mentioning that the theory of constructing the computer hardware on logic devices uses Mealy and Moore machines, which are defined by graphs loaded on arcs Fig. 15,16. This opens up new prospects for the construction of computers with new flexible architecture defined by the appropriate graphic programs (R -schemes) and with the hardware implementation of the relevant libraries. All of that can greatly simplify programming and raise the *level of technology and automation* in developing graphics programs in computers with such architecture.

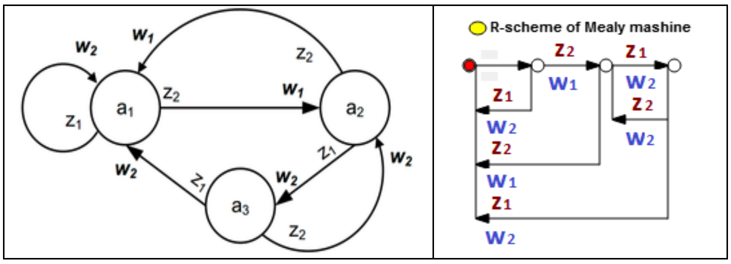


Fig. 15. R-scheme of Mealy machine.

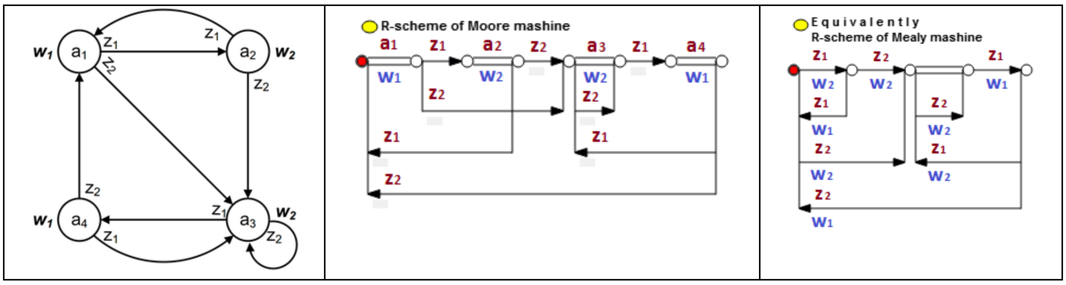


Рис. 16 R-схема автоматы Moore and Mealy.

Throughout the whole life cycle of the new concept a drawing of program project coincides with the program, its documentation and *network schedule* for its development, Fig. 17. **For the first time** the project manager can write in special brackets the developer’s name and estimated time of for its completion near every project on the R-scheme arc (draft program) and also **check** the work quality. That does not exist either in the conventional programming or in the industry. In any industry, for example, a drawing (documentation) of a vehicle *differs* from the vehicle itself and the network schedule for its development.

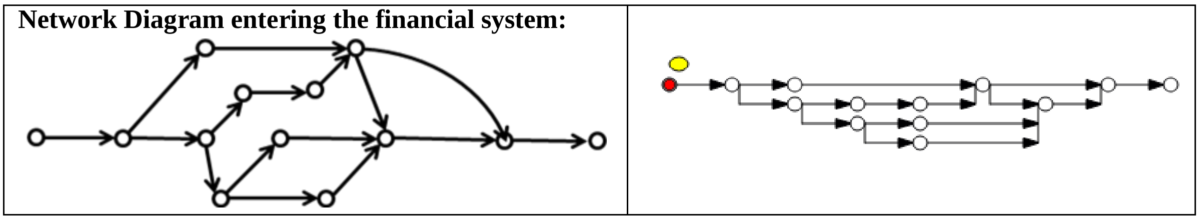
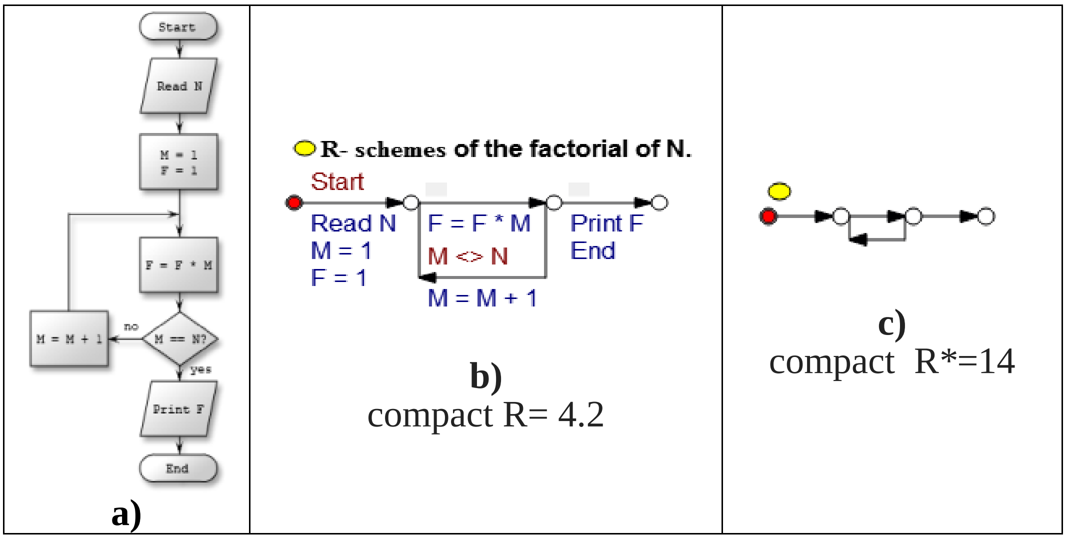


Fig. 17. Recording of network schedule in R-schemes.

The future development of graphic form of recording is promising. Color may be used for recording vertices, arcs and making records on them. The vertices and arcs arrows can have various configuration; the arcs can be double, wavy, dashed, etc.

3. NEW CONCEPT AND GRAPHICS IN CONVENTIONAL PROGRAMMING

Mankind has always been trying to find graphic images to any of its actions. We know a great number of graphic recording methods in the form of graphs loaded through the nodes: Block diagrams, UML, Workflow, SDL, DRAGON, Google BLOCKLY, etc., see Fig. 18a, 19 (left), and 20 (top). Conventional programming languages are always used in the final stages of those graphic programs. R-scheme are graphs loaded through the arcs, Fig. 18 bc, 19, and 20 (bottom), which



a)

b)
compact R= 4.2

c)
compact R*=14

Fig. 18. Example of recording of algorithm for computing the factorial of N in block diagrams (a) and in RR*-schemes (b, c).

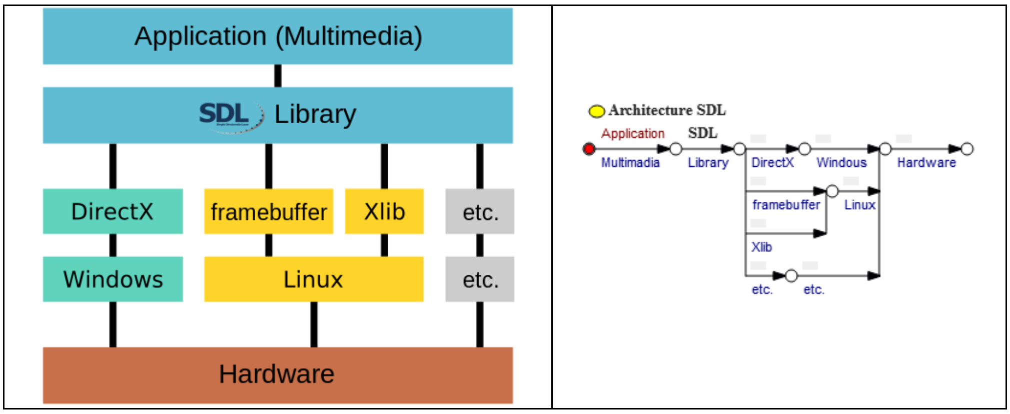
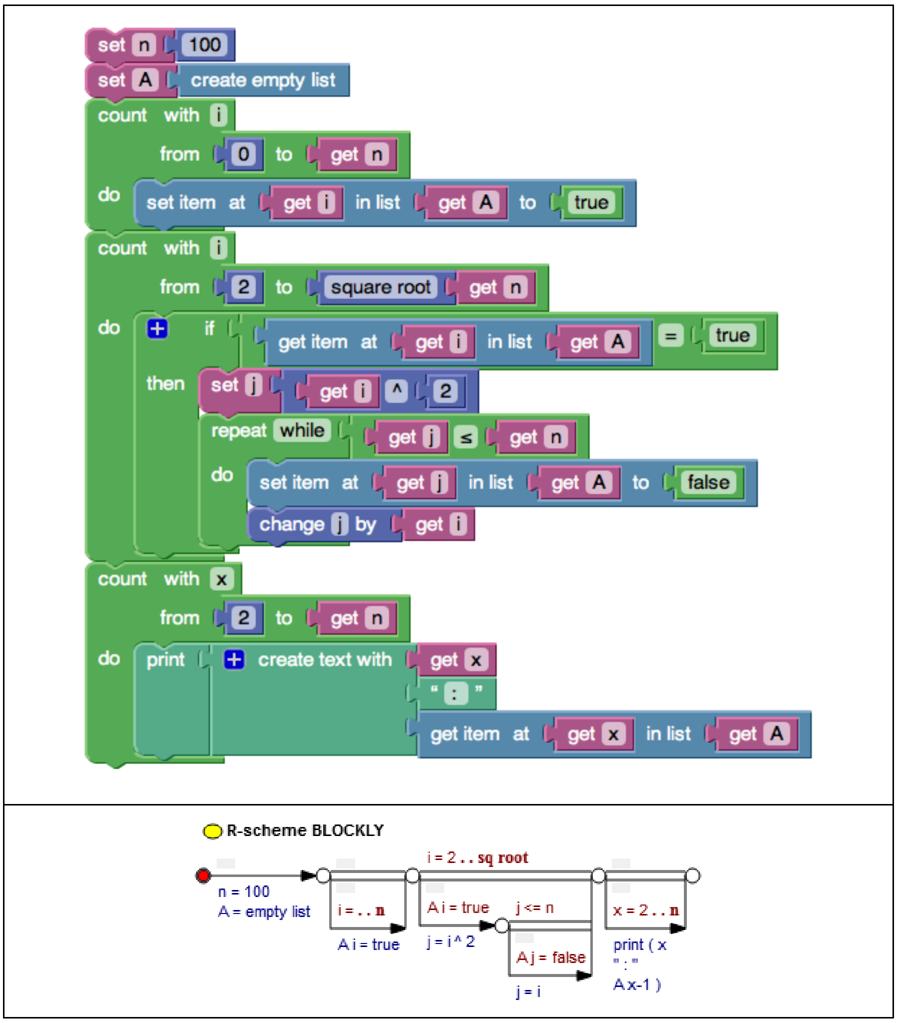


Fig. 19. Workflow R-scheme in SDL architecture.



```
set n 100
set A create empty list
count with i
    from 0 to get n
do  set item at get i in list get A to true
count with i
    from 2 to square root get n
do  if get item at get i in list get A = true
    then set j get i ^ 2
         repeat while get j ≤ get n
         do  set item at get j in list get A to false
             change j by get i
count with x
    from 2 to get n
do  print create text with get x
                           " : "
                           get item at get x in list get A
```

Fig. 20. Example of program recording in the visual programming language Google Blockly [11] and in R-scheme. Compactness R=**3**, R*=**14.**

have considerable qualitative and quantitative advantages. They are *executable* on a computer throughout the lifecycle, and also are simpler, better visualized and more compact. There is no need in the existing programming languages for their computer recording and processing.

Thus, a single graphic environment is proposed in the new concept for all languages, including the human, mathematics and programming ones. Recording of programs and projects in this shell, as well as their translation and interpretation become easier. For the first time the program (R -scheme) becomes an object of mathematics for the human (!) and not only for the computer. RR*-schemes are ideal for reverse translation of programs in any language into a new single graphical shell, as a model. As a result, from a "thing in itself", which is understood only by its author (who is not always available), the program transforms into something clear for development and improvement by other people, and the entire programming process becomes simple and evolutionary with preservation and accumulation of experience in a unified graphics library. Therefore, in the new programming concepts, in the same manner as in mathematics, there is no need in permanent development of programming languages and environments.

4 . ERROR-FREE AND EVIDENTIAL STYLE OF PROGRAMMING WITH THE NEW CONCEPT

The conventional concept of developing an algorithm (program) is based on the construction of computational scheme for solving the original problem with the help of about ten statements, which are not rigorously defined, and are fixed, text-type, machine-oriented and unnatural for a man. Therefore the original problem needs to be converted to match the algorithm (program) recording capabilities in those statements. This is a well-known, complex, multi-step process of designing algorithms and programs. As a result of this process, the original problem is transformed "beyond recognition", new elements are introduced, the relationship between its elements and motivation of the decisions taken is lost, as well as the whole understanding of "how it all works". To simplify this process, numerous languages, methods and environments are invented, which further confuses and complicates the process of programming. Therefore, the current complex and bulky programs are being debugged throughout their entire life.

The new design concept is based on the construction of a logical graphic R-scheme of the original problem in the natural language, terms and wording of the problem statement. For this purpose the method «step by step from logic» is applied. This process identifies and clarifies all the basic inaccuracies in the formulation of the problem. As a result, the logical R-scheme of the original problem becomes equally clear to both the client and the executor and is approved (signed) by them, Fig. 21.

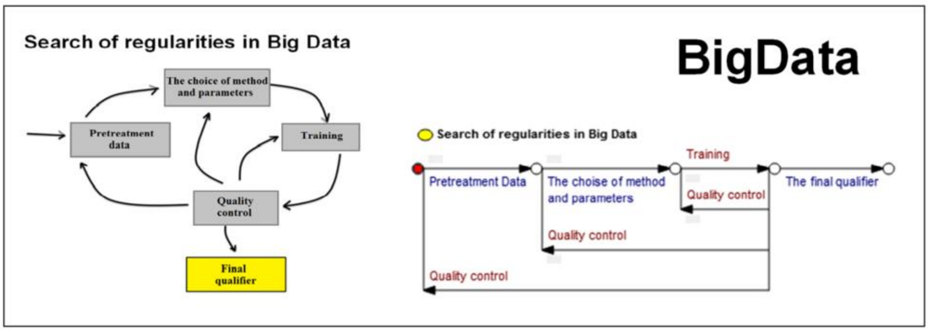


Fig. 21. Example of starting solving a complex problem in the new concept.

After that, the draft algorithm and program development is carried out by formal methods of mathematical derivation in the Graphical Programming Environment. GPE monitors and directs a person in such a way that his formal transformations exclude all the ambiguities in understanding from the original logic scheme and transforms it into the language *natural to a man* (as a rule, into the R-graphical language of simple mathematics) and subsequently, automatically, without a human, into the computer language. As a result the sequence of transformations of the original R-scheme is automatically stored in GPE, and in the future it will be a formal proof of the program correctness and of its development process. The resulting documentation formed automatically in GPE differs from the conventional one, as it saves the motivation of decisions, it is more compact, self-documenting and contains abstract R*-schemes, which are formally *optimized* (transformed) in GPE by various criteria – time, memory, etc.

The language natural to a man is being continuously improved (it evolves with the man) and approaches the effective professional language of the relevant team of program developers. It is important that the language of R-schemes in such description is **uniform (!)** for the computer, for the client, and for all the performers of the project and its users throughout its life cycle. Since this language is graphical and intuitive, it is mainly used mainly in the process of mathematical derivation, it is largely automatic, and it dramatically reduced the conditions for errors (misprints), therefore programmers call it "error-free", and the whole process and style of such programming is called "evidential".

5 . IMPLEMENTATION OF GRAPHICS PROGRAMMING ENVIRONMENT

At present, the graphical programming environment using logical R-schemes (GPE) has been *implemented* (lab version), which includes: the entry system for R-schemes and any texts on the arcs in any language, creation and memorization of the project tree, universal graphics editor; converter of R-schemes into R* and reverse; compiler of RR* in C++, etc. This graphical environment has been developed in C++ as a GPE plugin for *Qt Creator*. It consists of 5 areas, which divide and form the display screen. The main (about 90%) central part of the screen is occupied with 1) The Work Field, which is used for development of all R-schemes of the project. The top three lines are used for arrangement of 2) Menu for establishment of the environment architecture, 3) Toolbar (with 14 graphical icons), and 4) Panel of open (unlimited number) names of Work Fields of the R-schemes, 5) The last area occupies a narrow strip (5%) of the screen for storing the *Tree* of R-schemes of the specific project. The graphical environment is implemented on the basis of Einstein principle: "Everything should be made as simple as possible, but not simpler", therefore there is nothing easier and more efficient at the moment than the said GPE RR*. As a result, the implementation of the RR * GPS was much easier to implement modern environments and translators from traditional programming languages. For example, the now implemented laboratory version of the GPS contains fewer bytes of memory than the description (12 pages, 19 MB) described here, of the new graphical polyglot programming concept, and many (50+) times less than the C ++ interpreter. The main advantage of implementing the new concept is that access to the tools and libraries of programs of traditional programming languages is preserved.

Thus, the implemented graphical environment has summarized the available experience of programming in graphs, and it is considered materially complete (80% according to our estimates) for the construction of commercial version on its basis.

6. OPINIONS OF INDEPENDENT EXPERTS AND FIRST USERS

1) Interesting and strong polyglot concept of programming that is easier vs the existing one; many people can use it. 2) New mathematical tool for efficient analysis, synthesis and learning of programming. 3) It increases the longevity and evolutionary use of programs in relation to their conditions of using. 4) It has higher protection from errors and well-functioning completed graphics programs. 5) After two weeks of drawing R-schemes in the new programming environment I solved the problem, which I would have never solved without them... 6) The new concept is a useful and great tool not only in programming, but EVERYWHERE where it is necessary to find a solution to any logically complex (complicated) problem. Compared with the known Flow-chart, UML, etc., they have prominent advantages. R-schemes are executable on the computer throughout the life cycle of any project, they are better visualized, more compact and not cluttered with unnecessary details (figures). They are suitable for imaging of nonlinear algorithms (e.g., C++ classes) or data structures. Therefore, I absolutely do not understand why this tool does not enjoy a *tremendous* popularity. 7) It is an absolutely smart system. I'm not a programmer, but use it to produce a variety of diagrams and for solving of technological and managerial tasks. Everything is intuitive and easy to use. 8) I am delighted by using R-schemes for teaching the third-year students a special course on "Elements of Structural Programming Technique". Surprisingly even for myself I was convinced in the remarkable capabilities of R-schemes for structured storage of large volumes of data.

7. CONCLUSIONS

Thus, a new mathematical concept (culture) of programming is proposed, which is interesting thanks to its advantages: simplicity, clarity, compactness and possibility of adopting by anyone, not only by the computer programmers. This culture allows introducing the evidence and error-free style into programming, as well as the long-established mathematical methods of analysis and synthesis. Given the extraordinary (1-2 orders of magnitude greater) compactness of program recording, and ease of quick information entering especially through using of touch screens, the new concept has great prospects for effective application in small computer devices (tablets, iPhones) of mass use.

New proposals are particularly effective for the elementary training in programming due to their simplicity and visualization. Therefore, they can be included into the system of compulsory education of any specialists. **EVERYONE** should be able to program, and programming should become a part of the universal literacy and society culture. The effect of applying the new proposals for professional programming is the greater the more logically difficult and confusing is the problem to be solved. The new programming culture has tremendous perspectives in the future for building a human-computer society and its *evolution together* with a man, where a computer implanted into a human being will control, together with the brain, the physiology of a specific person, compensating its deficiencies from birth and providing treatment through life without using the conventional drugs.

It is common knowledge that the ***links*** between neurons and regions play the major role in the human brain. ***Arcs*** between the nodes play a similar role in the R-schemes. The number of arcs, their direction and 3D-orientation between different nodes and R-schemes are infinite. The node can remember the number of references to it and have a sensitivity threshold for its work, like a brain cell. It is promising of automatic generation of programs by data recording in the same concept. Nothing like that can be found in the contemporary programming. On the whole the new concept is simpler compared to all the known ones and it eliminates the problem of complexity in modern programming. It is quite well justified for new and effective applications and opens an interesting era of perspective development of the computer science, including the implementation of new computer hardware architecture and ensuring cyber security.

References

[1] Glushkov, V.M.; Velbitskiy, I.V. Programming technology and problems of its automation, Ukraine USIM 1976, №6, pp.75-93.

[2] Velbitskiy, I.V. Programming technology, publishing Technics, Ukraine, 1984; 279p.

[3] Sergeev, V.G. – Chief designer of control systems of rocket and space complexes, publishing HARTRON, Ukraine, 2014; 448p.

[4] Dijkstra, E. Letters to the editor: go to statement considered harmful, *Communications of the ACM*, 1968; pp. 147–148.

[5] Standard ISO/IEC 8631. Information technology, Programme constructs and convention for their Representation, 1989.

[6] McHenry, W.K. Technology: A soviet visual programming», *Journal of Visual Languages and Computing*, 1990; v.1, № 2.

[7] Velbitskiy, I.V. Graphical Programming and Program Correctness Proof, IEEE: 10.1109/ CSIT-13.6710368, 2013; pp.85-89.

[8] Velbitskiy, I.V. New Graphic Poliglot Concept of Programming, IEEE 40th Annual COMPSAC 2016, USA, 4p.

[9] Valvachev, A.N.. Syrkov, K.A. Programming in Delphi, http://www.rsdn.ru/article/Delphi/, 2005

[10] Stephen Prata, C++ Primer Plus, Fifth Edition, SAMS Indiana, 46240 USA, 1184p., ISBN 5-8459-1 1 27-3 Moscow. 2007.

[11] *Cade Metz* Google Blockly Lets You Hack With No Keyboard, https://en.wikipedia.org/wiki/Blockly — 2012