



TRIZ Future 2011, Dublin, Ireland

TRIZ-evolution of Programming Systems

Victor Berdonosov^a, Tatiana Sycheva^b

^a *Komsomolsk-na-Amure State Technical University, Faculty of Computer Technologies, Russia*

^b *Komsomolsk-na-Amure State Technical University, Information System Department, Russia*

Abstract

The procedure of systematization programming languages on the TRIZ-evolution base is considered in the report. The procedure of systematization programming languages is illustrated on the example of the programming languages paradigms development. The algorithm of presentation TRIZ-evolution map [1] of the programming languages paradigms is described.

The complete analysis of programming paradigms as specified in the evolution of programming languages allowed to determinate the progressing and inheritance paradigms of programming languages. The major contradictions had been arisen in the modern programming languages is established [2]. With the using of TRIZ tools it is derived solutions of contradictions and defined set of the properties, which will be included to the new programming paradigm.

The Imperative Paradigm, The Object-Oriented Paradigm, the Functional Paradigm, the Logic Programming Paradigm are considered in the report. It is determined the principal contradictions which became the “moving force” of each new paradigm, solution of this contradictions on the base of TRIZ tools [3]. The evolution of programming languages by the criterion “from contradiction to contradiction” is formed. The TRIZ-evolutionary map on the base of TRIZ-tools which *is* used for transition on development programming paradigm is formed.

The present approach usage allows forecast the appearance of new programming paradigms, new programming languages and new methods of programming. It becomes possible to determine the tendency of programming languages development and prediction of the next languages generations.

Keywords: programming systems; TRIZ-evolution; systematization of knowledge; concepts of systematisation programming languages; programming paradigms.

1. Introduction

The expressive power of language affects to the profundity of thoughts and makes it possible to describe thoughts more perfectly with using the control language structures and features of languages. It is

difficult to comprehend the structure, which can't be described verbally or in writing, so for the expression of abstract ideas has become necessary to develop a languages describing these ideas and the control language structures, which could be used to describe them.

The programming languages have been created to express the software ideas, which began to have a variety of individual properties, depending on the assigned task. Thus, different languages have different possibilities of expression programming ideas. Hereupon, there was a problem of using programming languages concerned with the absence of necessary features.

Quite often, the language constructions of the one language can be imitated in other languages, do not support them directly, and therefore many language features can be imitated in different languages. This arose owing to the evolution of programming concepts. Under the programming concepts is understood the contradiction or a set of contradictions, which led to the appearance of a new group of programming languages. Obviously, each new group of languages can't be fundamental new, but will retain features of the predecessor.

Thus, the study of different programming languages is reduced to the study concepts of programming languages, and the selection of programming languages is reduced to the determination the best set of language features for a practical purposes.

In order to demonstrate the evolution of the paradigms of programming languages we must choose the specific task, which we will try to resolve with the using of different programming languages. The task will be to calculate the factorial of number. This task has been selected by the criteria: ease of realization, the availability of various methods of calculation, even within one language.

2. Main definitions

Imaging "programming language", based on the opinions of different authors. It is known [1] [2] [3] [4], that the notion of programming language is a special case of the notion "language", which means the tool of communication between the transmitter and the message receiver to realize the program ideas.

In the field of programming the role of transmitters and receivers can act as computer or people messages, and they will interact with using the commands and operators that are understandable. Understandability of order codes and operators for a human depends on a specific programming language, with which a human realizes a programming idea, meaning that appropriate methods for reacting to reports of the author in the each programming language should be developed.

The "programming idea" in this case should be the direction of activity on basis of which we describe the programming task. The "programming task" is providing interaction between the computer and human.

We have defined the conception "programming" as a process of describing the idea of programming with using programming language, understood by the receiver and transmitter. Therefore, the "programming language" is a special case of the notion "language", which is a means the tool of communication between the transmitter and the receiver to realize the program ideas.

It should be noted that the notions "concept of programming" and "programming paradigms" are different. Notion "programming concept" is the "moving force" for creation of each new paradigm. For the first time Thomas Kuhn introduced the concept of "paradigm" [5]. Kuhn determined paradigms as established system of scientific opinion within which we conducted the study.

3. Programming paradigms

3.1. Native programming paradigm

Native programming is set of computer's machine-instruction code, which is interpreted for a specific microprocessor. This paradigm is become actively developing in 1940s, although the preconditions for the establishment appeared in the 19th century. This is the first and the most primitive of programming paradigms, see Fig 1.

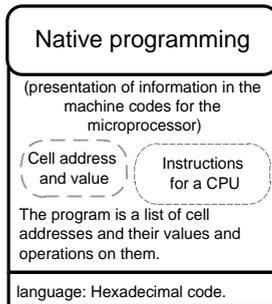


Fig. 1. The frame of “Native coding” paradigm.

We present a program in machine code for the calculation factorial of numbers.

ADR CODE	LINE	COMMENT
	1	; factorial calculation
A000 DB	2	; value N
	3	;
4000 8B 06 00 A0	4	; boot N in the accumulator
4004 3D 01 00	5	; if (@N=1) or (@N=0) then
4007 77 00 41	6	; transition to the calculation
400A B8 01 00	7	; factorial:=1
400D 33 12	8	; register emptying
400F C3	9	; return
	10	;
	11	; calculation
4100 48	12	; else
4101 50	13	; accumulator in stack
4102 E8 00 40	14	; factorial := factorial(@N-1)
4105 F7 E6	15	; * @N;
4107 C3	16	; return

Of course, this paradigm has a lot of contradictions.

Contradiction 1: with the increasing of complexity of tasks the size of machine code (machine operations) increases unacceptably.

Contradiction 2: with the increasing of complexity of tasks the structure of the microprocessor increases unacceptably.

Contradiction 3: with the increasing of complexity of tasks the time of programming increases unacceptably.

To resolve the arising contradictions we use the inventive principle of Copying and the principle of Merging.

3.2. Assembling programming paradigm

Consider the specific resolutions for elimination of the identified contradictions.

Resolution 1: using the principle of Copying, the term of address cells and their values is replaced by the term "operand".

Resolution 2: using the principle of Merging, similar machine instructions and the mnemonic commands are combined.

Owing to the resolution of contradictions, we have a new resource: the ability to convert the machine instructions in the mnemonic code.

Assembling is paradigm, in which there is a compilation of source programming code, written in assembly language, into machine language code. Calls to the processor runs through mnemonic commands that conform to machine instructions of a computer system, see Fig 2. Assembling paradigm is become actively developing in 1940s, following the paradigm of machine coding.

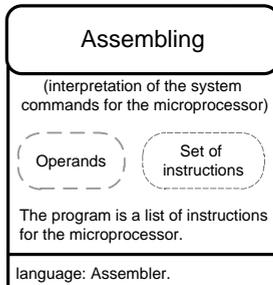


Fig. 2. The frame of "Assembling" paradigm.

Write a program in assembly language for the calculation factorial of numbers.

Factorial PROC

```

                                ; function factorial(@@N:Word):DWord;
                                ; begin
    arg @@N:word
    mov     ax,@@N
    cmp     ax,1
                                ; if (@@N=1) or (@@N=0) then
    ja     @@calc
    mov     ax,1
                                ; factorial:=1
    xor dx,dx
    ret
@@calc:
    dec ax
                                ; else
    push   ax
    call  Factorial
                                ; factorial := factorial(@@N-1)
    mul @@N
                                ; * @@N;
    ret
                                ; end;
endp

```

Based on example of the operator "mov" we can see that using of the mnemonic code, introduced by the resolution 2 of contradictions 1, 2 and 3, not only reduced the programming time, but the size of machine code, without the complicating the structure of the microprocessor.

The operator "mov" acts the function of sending the following operands: *reg, reg; mem, reg, mem, immed*, etc. In turn, for example, operand "reg" should contain one of the 8-, 16- or 32-bit registers from the list: AH, AL, BH, BL, CH, AX, BX, CX, DX, SI, DI, BP, SP, EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP. Operand "mem" should contain a cell address in the memory.

Thus, instead of detailed description of the machine instructions and memory address we get one mnemonic instruction "mov" with subordinated operands.

On the other hand, assembling made a programming too dependent on the certain hardware, so after some time it has been revealed the following contradictions.

Contradiction 4: the transferring the program to another platform unacceptably increases the number of incompatible parts of the software commands and CPU.

Contradiction 5: the increasing of complexity of tasks unacceptably increases the size of programming code.

Contradiction 6: the increasing of complexity of tasks unacceptably increases programming time.

Contradiction 7: the increasing of number of implemented mathematical functions unacceptably breaks the logic of computing.

Contradiction 8: the increasing of complexity implemented mathematical functions unacceptably breaks the logic of computing.

Contradiction 9: the proving of mathematical theorems unacceptably breaks the logic of the calculations.

To resolve the arising contradictions we use the inventive principle of Merging, Taking out and Preliminary action principles.

3.3. Imperative programming paradigm

Consider the specific resolutions for elimination of the identified contradictions 4, 5 и 6.

Resolution 3: using the principle of Merging, we have combined a similar machine code to the instructions (imperatives). For this sets of machine instructions for the microprocessor and sets of elementary operations have been combined and have been associated with instruction word.

Appeared resource: the ability to substitute the machine code on imperative.

Imperative programming is programming paradigm, which describes the process of computing with using the programming instructions that change the status of the program. Imperative program represents the commands that are expressed by the imperative of natural languages, that is, this is a sequence of commands to be performed by computer, see Fig 3. It is become developing in the 1950s at the one timeline with the functional programming languages.

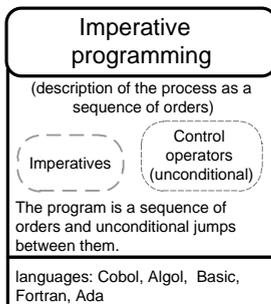


Fig. 3. The frame of "Imperative programming".

Let's write a program in the imperative language Basic for the calculation factorial of numbers.

```
10 n=0
20 j=0
30 k=0
40 cls
```

```

50 input "Input n = ", n$
60 if n<1 goto 90
70 k=1
80 for j=1 to n
90 k=k*j
100 next j
110 print " Factorial n= "; k$
120 END

```

An important event in programming of more complex tasks was the introduction of the imperative to the programming code, which was the same machine instruction but replaced by associative word, for example, the imperatives *input*, *print*, *next* etc.

However, this introduction had a negative side. This example shows the mechanism of the unconditional jump: the operator “*go to*” in the case of true condition $n < 1$ redirect to the certain point in the program. For large-scale problems it became impossible to use such jumps, and eventually revealed the following differences.

Contradiction 10: mechanism of the unconditional jump to a certain point in the program led to increase in unstructured programming constructions, so it unacceptably increases the compile time of program.

Contradiction 11: a large quantity of unconditional jumps unacceptably reduces the interdependence of code fragments.

3.4. Logic programming paradigm

Consider the specific resolutions for elimination of the identified contradictions 7, 8 и 9.

Resolution 4: using the Taking out principle, we have refused the assignment statements and control statements.

Resolution 5: using the principle of Preliminary action, we began to represent the program as a statements of symbolic logic.

New resources: the possibility of uses the automatically calculation of predicates and automatic theorems proving.

Logic programming is programming paradigm, based on automatic theorem proving. Logic programming is based on the theory and mechanism of mathematical logic using mathematical principles, see Fig. 4. This paradigm is appeared more lately then predecessors in the 1950s, when the fundamental paradigm have already been studied yet.

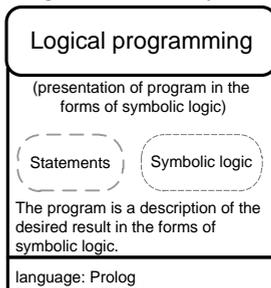


Fig. 4. The frame of “Logic programming” paradigm.

Let's see how the program code of the considered task will be presented in the logical language Turbo Prolog.

```

domains
  N,F=real
predicates
  factorial(N,F)
  result
clauses
  factorial(0,1).
  factorial(N,F):-N>0,N1=N-1,factorial(N1,F1),
    F=F1*N.
  result:-write("Input N"),nl,
  write("N="),readreal(N),factorial(N,F),
  write(N,"!="F).
goal
  result.

```

The program demonstrates how this task is solved by the introduction of the principles of logical output of information based on given facts and input rules. Similar theorem proving, there are divided section of *domains*, *predicates*, *clauses* and *goal*, with helping whom the algorithm solving of task is formed.

That's why this paradigm has been created to describe mathematical tasks, and that's why the logical approach has become the most applicable and effective for solving problems of discrete mathematics and theorem proving. In addition, this approach has always served as an alternative to imperative programming. But with the increasing complexity of programming tasks for a complete solution it had become required the introduction of objects and the bind to the algorithms of mathematical logic.

Contradiction 12: with the increasing complexity of programming task, that require generalized approach, the structure of the program which simulate the object properties of language is complicated unacceptably.

3.5. Functional programming paradigm

Consider the specific resolutions for elimination of the identified contradictions 7, 8 и 9.

Resolution 6: using the principle of Taking out, we abandoned the notion of "variable", assignment statements and, accordingly, on the storage conditions of the program, as well as abandoned method of calculating the sequence of changes conditions of the functions.

Appeared resource: the possibility of using algorithms of mathematical logic by microprocessor.

Functional programming is programming paradigm, in which process of computation is treated as a process of computing values of functions in mathematical understanding, see Fig. 5.

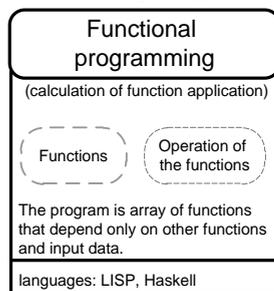


Fig. 5. The frame of "Functional programming".

Program calculation of factorial in a functional language Common Lisp may be as follows.

```
(defun fact (n)
  (if (eql n 0)
      1
      (* n (fact (- n 1)))))
(fact 10)
```

Analyzing the principle of the program code, we can see that in contrast to the imperative paradigm, which describes the steps leading to the goal achievement, the functional paradigm describes the mathematical relation between the data and result.

In this example, the relation between the input experimental data and the result are established with using the description of the special function finding factorial of a numbers. This function describes the typical algorithm for functional programs: if the input number is 0, the result will be 1, otherwise the result will be calculated recursively as $*n$ (fact (- n 1)). Note, that the functional language does not imply a comparison of conditions, because language hasn't the notion of variable (in imperative sense) and the notion of assignment.

In the imperative sense, the variable must keep changing statuses of the data cells, which contradicts the notion of a variable as a storage constant input data. However, nothing prevents to use the imperative understanding variable as function, which would become the repository of the variable values. This is the multiparadigmatic basis of functional languages [6], which was used in the generalized programming.

Contradiction 12: with the increasing complexity of the programming task, which requires multiparadigmatic approach, the structure of the program is complicated unacceptably, because there is an imitation of object concepts (objects, classes, etc.) with constructions on logical languages.

3.6. Structured programming paradigm

Consider the specific resolutions for elimination of the identified contradictions 10 и 11.

Resolution 7: using the principle of Preliminary action, we put the parts of the code previously so code would be executed on the most logical structure using conditional jumps.

There was a new resource: the possibility to represent code in a hierarchical structure using the logical operators.

Structured programming is a programming paradigm based on the idea of hierarchical structure of blocks, see Fig 6. Structured paradigm is appeared in the 1970s, after appearance unresolvable contradictions in the previous paradigm.

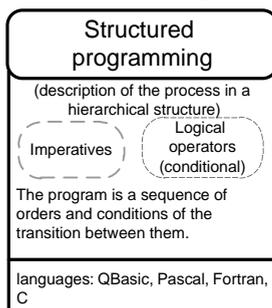


Fig. 6. The frame of "Structured programming".

Program in Pascal language for the calculation of factorial numbers will be presented the structural organization of the code.

```
function fact(n : integer) : longint;
begin
  if n <= 1 then
    fact := 1
  else
    fact := n * fact(n - 1);
end;
```

This example shows how the rejections of the unconditional jumps bring to an increase of structuring the code. There were conditions *if [] then [] else, do [] loop while* and others, providing the execution of certain instruction only if the logical expression is truth, or execution one of set commands, depending on the value of expression.

However, the increasing complexity of programming task bring to the using large number of similar operators, for example, operators *begin* and *end*, previous declarations of variables etc.

Contradiction 14: emergence in the program code sets of the similar operators and constructions, used for similar objects, increases the program compile time unacceptably.

3.7. Object-oriented programming paradigm

Consider the specific resolutions for elimination of the identified contradictions 12, 13 и 14.

Resolution 8: using the principle of local quality, we culminate from the homogeneous to inhomogeneous structure of the code, which combined the homogeneous parts of the code.

Resolution 9: using the principle of Merging, we merged abstract data types into the classes and class instances to the objects.

Resolution 10: using principle of Universality, we made it possible to use the identical functions for different objects, through the method of inheritance of properties one class from another class.

Resolution 11: using the principle of Nested doll, we introduced the notion of a “parent class” and “inheritor class”. The inheritor class became dynamically linked with the parent class, the parent class properties began to move in inheritor class.

There was a new resource: the ability to write code with a higher structure organization.

Object-oriented programming is programming paradigm, in which basic concepts are the concepts of objects and classes, see Fig 7. This paradigm is appeared in the 1980s with appearance such language as C++.

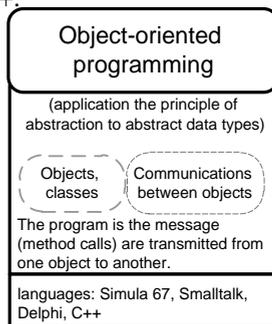


Fig. 7. The frame of “Object-oriented programming”.

The algorithm of the task on the C++ language may be as follows.

```
unsigned long long fact(unsigned int n)
{ if(n <= 1) {return 1;}
}
```

```
else {return n * fact(n - 1);} }
```

This example shows that the rejection of using the homogeneous constructions of structural languages, led to the insertion of visually light constructions. For examples, the operators `{}` replaced similar operators *begin* and *end*, the syntax of the conditional construction dropped to using just *if(condition)* and *else*, finally, it became possible to write condition on one line without breaking the logic of program. As a result we have received a higher code organization.

Contradiction 15: with the increasing complexity of the programming task, requiring multiparadigmatic approach, the structure of the program is complicated unacceptably, because there is an imitation of the logical and functional concepts (λ -calculus, facts and goals) with structural object languages.

4. The formation of new programming paradigm

Let's try to propose a new programming paradigm, which may be as developing of the functional paradigm.

According to the contradiction 13, it is unacceptable to create such language constructions imitates methods another paradigm. At the same time for solving of complex tasks it is necessary: on the one side we must move to the inhomogeneous structure of the code, on the other side the paradigm must describe mathematical relations between the input data and the result data with using the change status of functions, it must involve the principles of multiparadigmatic approach, it must retain the mathematical style of programming, it must support the calculation of higher order functions and λ -calculus (lambda calculus).

Resolution 12: using the principle of Intermediary, we introduced the notion of “libraries”. It is such descriptions of data and algorithms that can be applied to different data types without changing these descriptions.

Resolution 13: using the principle of Local quality, we move from the homogeneous structure of code to the inhomogeneous structure, which combine the homogeneous parts of the code.

Resolution 14: using the principle of Discarding and recovering, we introduce the notion of indention instead of similar symbolic constructions, thus we introduce a hierarchical structure of the code.

Resolution 15: using the principle of Homogeneity, we introduce the notion of macros. Macros can be used as a calling of special methods or as a new language constructs, such as, for example, operators of conditional jumps to do the non-functional concepts to functional.

There is a new resource: the description of non-functional concepts by functional methods.

New programming paradigm is paradigm, where priority is set to the solving of task with multiparadigmatic approach and this approach is realized using the description non-functional notions of the functional methods, see Fig. 8.

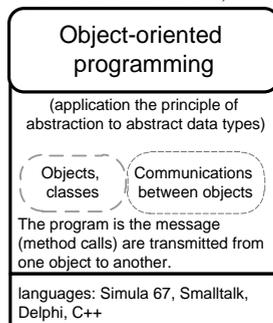


Fig. 8. The frame of “New programming” paradigm.

Example of the program for calculating the factorial by the experimental language of this paradigm may be as follows.

```
macro @if (clause, a1, a2) // example of the macro if
syntax ("if", "(", clause, ")", a1, "else", a2)
<[
  match ($clause : bool)
    | true => $a1
    | _ => $a2
]> // end example
```

```
def factorial(x):
if (x=0) return 1
else reduce(lambda x, y:x*y, xrange(1,x+1))
```

This example demonstrates, how resolutions 12, 13, 14 и 15 is realized for new paradigm.

Firstly, with using the resolutions 13 and 14, we moved to the structured construction of code and introduced the notation of nesting by indents. Second, with using resolution 12 we introduced the notion of libraries, particularly, we used a library of macros. With using resolution 15 we introduced the possibility to write native macros, for example, the macro for operator *if*.

In addition, we retained the mathematical style of programming, the support of computing higher order functions and had the basis on lambda calculus, for example, reduce (lambda x, y: x * y, xrange (1, x +1)).

5. TRIZ-evolutionary map of programming paradigms

The appearance of all (besides the Native coding paradigm) programming paradigms has arose as a result of resolution contradictions with the TRIZ-tools, identified in the prior paradigms. As a result of this, system ideality of paradigms has increased and programming on the languages of paradigms has become more productive and reliable.

Combining programming paradigms and principles of resolution contradictions, generating this paradigms we have TRIZ-evolutionary map, see Fig 9.

Using TRIZ-evolutionary map allows considerably improve learning efficiency through the systematization of knowledge, particularly knowledge about the evolution of programming paradigms. There was implemented the following method [7]. Previously, students learned all of the tools of TRIZ, if for some reason you cannot learn all TRIZ-tools, you may learn only resolution contradictions methods.

After that it is begins the exploration starting from the simplest paradigm of Native coding. We show by example, what contradictions arise when we use such paradigm, shows the areas in which this paradigm the most effective. Then they increase the tasks and offer to students write more complexity programs. Students can see that there are new contradictions. Students are suggested to identify contradiction and attempt to resolve the contradiction by the TRIZ-tools. It means to offer another (more perfect) programming paradigm.

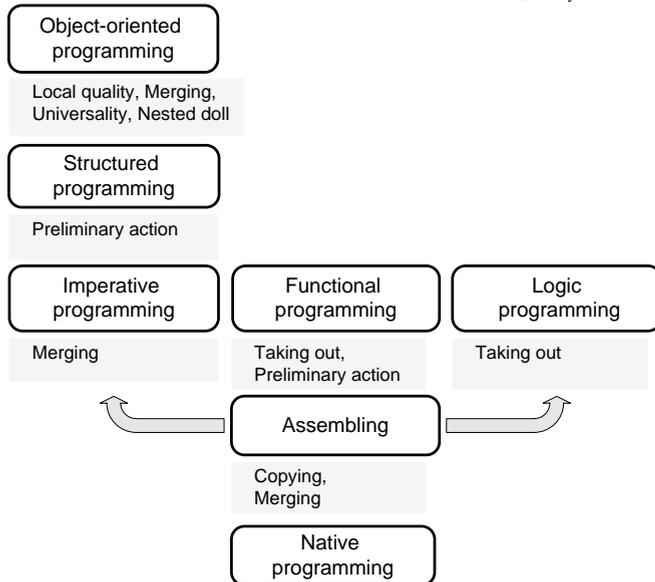


Figure 9: The fragment of TRIZ-evolutionary map of programming paradigms

If it necessary the teacher helps the students. Thus, students "discover" for themselves all of the subsequent programming paradigm. If the course is quite full, they can consider the entire of modern known paradigm (figure 10). Masters can also solve more complicated task. With using TRIZ-evolutionary approach they should offer the new programming paradigm.

6. Summary

TRIZ-analyze of programming paradigms in this report makes possible:

- to substantiate the evolution of the programming paradigms;
- to substantiate the contradictions that trigger the mechanism of evolution;
- to identify the TRIZ-tools resolving the contradictions which became the moving force the evolution programming paradigms;
- to construct the TRIZ-evolutionary map of programming paradigm, which allows substantially intensify the process development of such paradigms by students.

Note that the development of TRIZ-evolutionary map of programming paradigms, through its completion, makes possible to predict the way of development paradigms.

In addition, TRIZ-evolutionary map can be completed in the way of the evolution of programming languages for each paradigm, which allows suggesting the most perfect programming languages.

References

- [1] Sebasta R. "Concepts of programming languages" (edition 9), Addison Wesley Publishing Company, 2009, ISBN13: 9780136073475.
- [2] Gabrielli M., Martini S. "Programming Languages: Principles and Paradigms", Springer London Dordrecht Heidelberg New York, Springer-Verlag London Limited 2010, ISBN13:9781848829138.
- [3] David Lightfoot, Clemens Szyperski "Modular Programming Languages: 7th Joint Modular Languages Conference", JMLC 2006, Oxford, UK, September 13-15, 2006, Proceedi, ISBN13:978-3-540-40927-4
- [4] Daniel P. Friedman, Mitchell Wand "Essentials of Programming Languages", 2008, MIT, ISBN13:978-0-262-06279-4

[5] Thomas S. Kuhn “The Structure of Scientific Revolutions”, 3rd Edition, 2001, University of Chicago Press, ISBN13:9780226458083.

[6] Paul Hudak “Conception, Evolution, and Application of Functional Programming Languages”, ACM Computing Surveys, vol. 21, №3, September 1989

[7] Berdonosov V. “Fractality of knowledge and TRIZ”, Proceedings of the ETRIA TRIZ Future Conference, Kortrijk, 9-11 October 2006, published by CREAX Press, ISBN 90-77071-05-9.

Contacts

Victor Berdonosov
Chair of Design and TRIZ Department
Komsomolsk-na-Amure State Technical University
International Av. 59 - 5
681024, Komsomolsk-na-Amure – Russia
E-mail: ktriz@knastu.ru, berd1946@gmail.com
Phone: +7-4217-535259; Mobile: +7 (962) 2875141

Tatiana Sycheva
Master of Information System Department
Komsomolsk-na-Amure State Technical University
Sidorenko Str. 32 – 15
681029, Komsomolsk-na-Amure – Russia
Mobile: +7 (924) 1168413
E-mail:SychevaTY@gmail.com

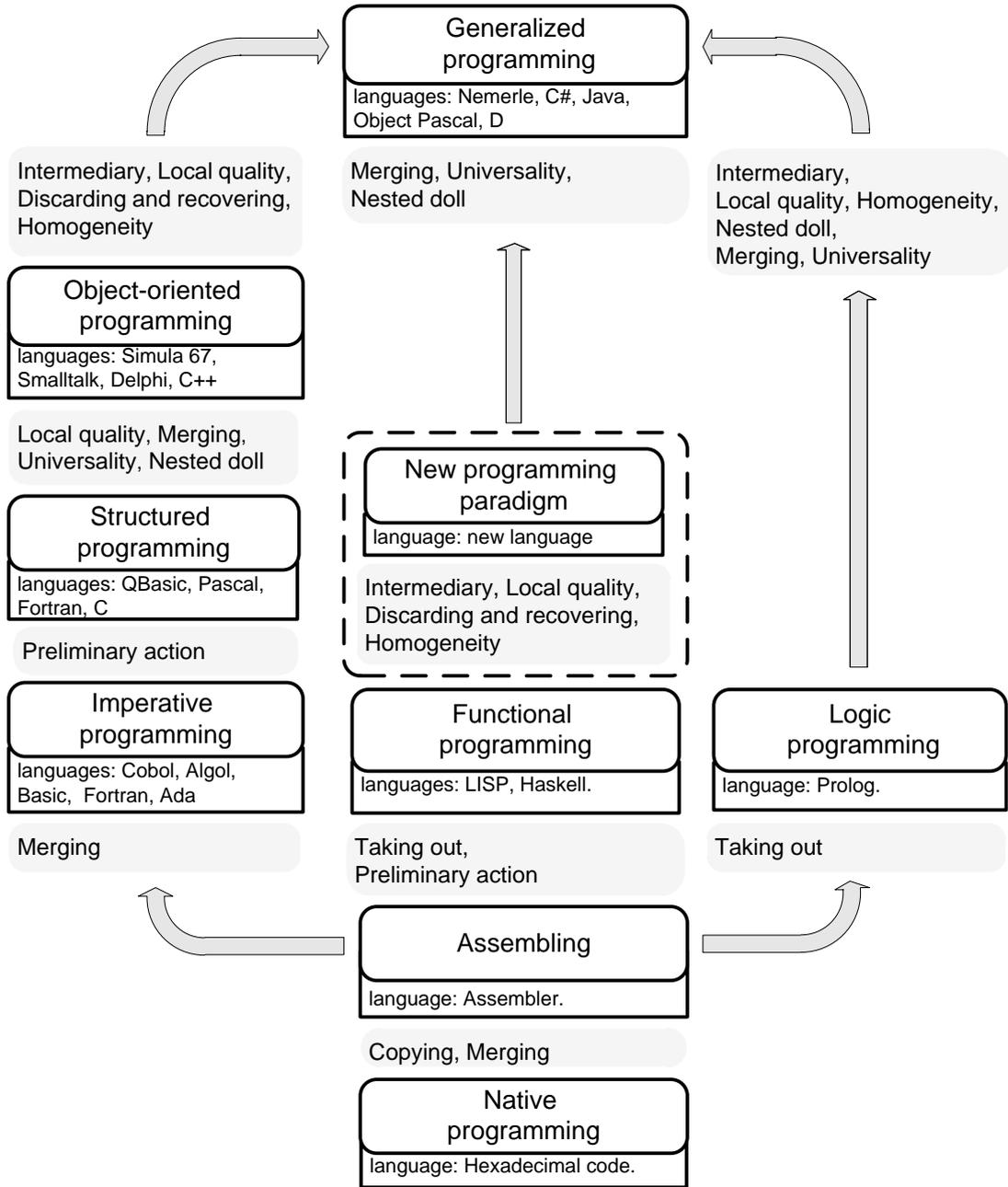


Figure 10: TRIZ-evolutionary map of programming paradigms.