SCIENCE OF THE COMPUTER PROGRAMS AND SYSTEMS IN XX-XXI CENTURIES: PAST, PRESENT, FUTURE

E.M. Lavrischeva, prof., doctor of phis.-math.science
ISP RAS, MIPT, lavryscheva@gmail.com

ABSTRACT

The ways of development of the programs and systems science in connection with the advent of computers in the (1945-1991, 1992-2018) in domestic and foreign directions are defined:

1) The schematic-theoretical, mathematical, algebraic, algorithmic, conception, reliable, etc. and theory of modular, synthesis, assembly, composition, explication programming in the USSR (1953-1991);
2) The formal mathematical apparatus VDM, Z, RSL, B, Clear etc. and proof theory of correctness of such programs in the Europe, USA, (1972-2000);
3) The object-oriented programming, system modeling UML, MDA, MDL, SOA, SCA etc., programming paradigms (automatic, functional, object, Agile, EX etc.) in the USA, Europe (1987-2017);
4) Software Engineering Methods and Theory (SE-1968, SEMAT-2009) for development of the theory of programs and systems for the academic and educational systems (USA);
5) Methods of production of variants (Product Line/Product Family) systems on feature model (MF), multi programming GDM, model configuration, certificate of software quality etc. (2004-2017);
6) Modern theory and methods modeling systems from software resources (objects, components, services, etc.) by OCM - verification models and resources configuration in the building output code products of system in Russia (2012-2018) and in the future;
7) The perspective ways of development theory systems - Web-services and smart Internet, graph theory in programming and nanotechnologies for apparatuses' and smart-computers for the biology, genetic, chemistry, medicine, etc.

Keywords: theory, schema, method, program, systems, reuses, object, model, technology, engineering, modeling, verification, testing, reliability, configuration, product.

INTRODUCTION

More than twenty years ago A. P. Ershov wrote that theoretical programming is a branch of mathematical sciences; the object of study which is an abstract program expressed a logical structure and information, to be performed on the computer. Theory programming is based on mathematical disciplines (logic, algebra, combinatory, graph theory, mathematic), and mathematical method reflects the thinking of an expert in conducting analysis of the subject area, understanding of formulation, and description of the programs to retrieve the machine a mathematical result. The theory of programming focused on professionals with mathematical knowledge and ability to apply them to the logic, algorithms and automats [1-6, 37-39].

At the initial stage of creating a computer in the USSR was formed the theory of programs and systems, based *on schema and graphs* (A.A. Lyapunov, A. P. Ershov, V. M. Glushkov, and Yu. I. Yanov, S.S. Lavrov, E.L. Yushchenko, E.H. Tyugu, etc.) [1]. Lyapunov's operator scheme (1953) from the theoretical point of view is <U, V, A, R, T>, where

$U = \{S_1, S_2, ..., S_m\}$ - set of S_J operators; $V = \{x_1, x_2, ..., x_n\}$ – set of variables X_I; $A \subseteq V$ x U -relation "to be an argument" (x_i, A, S_j); $R \subseteq V$ x U-relation "to have results" (S_j, R x_i); $T \subseteq V$ x U is the relation between the predecessor operator and the successor operator (S_j, T, S_k), i.e. the set of possible transitions.

Following the operating scheme, the algorithmic description of programs in programming languages (Algol, Fortran, PL.1, etc.). This description was transformed to a machine-view program using translators or interpreters. Such software modules were used to build systems.

A module is considered a software element that converts the plurality of source data X in a variety Y of the output results. System modules is a pair $S = (T, \chi)$, where T – model of the system; χ is the characteristic function, defined on the set of vertices X of a graph of modules [1-7].

The process of developing programs and systems from the modules gradually become regulated on base graph, the mathematical specification programs and the models or using the life cycles of (waterfall, spiral, integration, etc.) under designing.

Abroad were formed the *new formal approaches* to specification (*operational, axiomatic, denotational)* semantic *of programs* by D. Burner, D. Gries, I. R. Abrial (VDM, RSL, Z, B, etc.) and proof such programs of by methods R. Floyd, K. Hoare, E. Dijkstra, and others [8-12]. In *operational approach* semantics is described in terms of an abstract machine. For the implementation of the description creates the interpreter. John. McCarthy made such an interpreter from Lisp language for Lisp programs. In the process, the set of values of variables changes as a state. The result is the state of the program at which the program completes. *Axiomatic approach* is based on the system of axioms for basic language constructions and rules of derivation from these axioms of the result (R. Hoar, B. R. Floyd and P. Naur). *Denotational approach* is based on the model of the algorithm, which is described in terms of the theory of sets, relations and maps (VDM, Z, B, etc.), as well as the description of the results of the program execution by the machine, as a link between the initial and final state.

After a module is a new element of programming was *the object in OOP G. Booch* [13] and related math concepts such as class, inheritance, polymorphism, encapsulation etc. Any subject area in the OOP-a set of objects related to each other a set of relations and behavior: < object orientation> = <objects> + <inheritance>. A class is a set of objects that share variables, structure, and behavior. An object is an instance of a class. One of the properties of objects is encapsulation, which is implemented using interfaces that consist of methods and attributes that define the external variables of the class instance. For the subject area, an object model (OM) is created, on which the system is implemented. An example implementation such models OM - CORBA, Rational Rose, UML so on. Developed CASE tools object modeling systems (Rational Rose, UML, MDA, MDD, PIM, PSM, SOA etc.) [13-15]. The system was developed on the basis of characteristic patterns and ready-made software resources (objects, components, services, etc.).

The standard Swebok (www.swebok.com) identified 10 areas of expertise to describe the systems, from the requirements process to the delivery of the finished quality product. Then it were standards ISO/IEC 12207 Life Cycle 1996 (2007) and ISO/IEC 11404 – GDT 2007, ISO/IEC 9000 (1-4) Quality SW for building the different systems. Through Software Engineering Methods and Theory (*SEMAT-2009*) classification of SE disciplines [7] and proposed a promising theory and methods for determining scientific foundations for the programming to increase the level of knowledge and competence of specialists and postgraduates preparing for the production of software, information and application system (http://www.semat.org).

It was *defined first variable of the model* FM (Feature Model) and products defined in Product Line/Product Family, GDM, Grid, etc. by K.Pochl [16-26]. These models are based on of model characteristics FM and configuration model (MC) for building the basic artifacts and ready software resources of variability systems. The process of developing programs and systems are produced by on the ISO/IEC 12207 Life Cycle 2007, ISO/IEC 11404 – GDT 2007, ISO/IEC 9000, and Quality etc. For the Life Cycle standard 12207 was developed an ontological model and was presented by Lavrischeva E.M. "Ontological approach to the formal specification of the Standard Life Cycle" at the conference "Science and Information-2015" (www.conference.thesai.org).

The way of *development modern theory* and methods modeling systems from software resources (objects, components, services, etc.) was prepared OCM [18-23] - verification models and resources configuration in the building output code products and system and generation new system from web-services in Russia (2012-2018) [24-26];

In perspective on future it was the Web-service, Smart Internet and nanotechnologies for apparatus and systems for the biology, genetic, chemistry, medicine, etc. It was based on the concept of minimization of software elements and their grip on the type of nano in a small tool to apply the devices in medicine, geofizservice, Avio, and others. It will be developing intelligent machines, proposed by V. M. Glushkov, which helps to fill the mini computers elements of the human mind on the type of quantum [27-31]. Conception nano for programs was made in report of Lavrischeva E.M., Petrov I.B. on International conference «Future Technologies Conference (FTC) 2017 [32].

In this paper, a brief description of the scientific of programs and systems in USSR and abroad, the new approaches to modeling of systems on Product Line with new models (UML, MDA, SOA, SCA and etc.) and method OCM from logical-mathematical apparatus modeling SW systems and the way of development programs bases on web-service, smart Internet and the nanotechololology.

1. DEVELOPMENT THEORY PROGRAMS AND SYSTEMS IN THE USSR

The first theories programs and technologies programming

Theory of programs by A. P. Ershov forms a new branch of mathematical science, the object of study which is mathematical abstractions of programs, regulations, expressed in a special language with predetermined information and a logical structure for execution on a computer. The basis of the theory was a scheme that was first introduced by A. A. Lyapunov and which was developed by Yu. I. Yanov and S.S.Lavrov, E.H.Tuygu and others [1-7, 33, 34, 38-40].

The scheme is a finite directed graph describing the communication scheme of the individual functions programs using signatures of operations and mathematical symbols. Janov scheme is a model of operation S_i (x_i) and relation V, A (arguments) and V, R -resultats. The schema defined a complete system of transformation that appears in the protocol sequence of operations and values of their variables.

A.P. Ershov developed the concept of schemes and formulated the idea of information computable functions in terms of determinants, invariant to different ways of setting the computation of the programs.

Automatic, perceiving this determinant, considered as a finite state machine, allowing formal equivalence, which coincides with the functional interpretation of the algorithm of the program. Formal notation programs is given in the lexicon, and contains a description of the semantics as a set of non-trivial facts about calculated its functions. The theory of schemes of programs, the computability of the algorithms were developed by A.P.Ershov, V.E.Kotov, S.S.Lavrov and others [33-40].

Lavrov S.S. is described theory of schemes of programs, based on model of Janov operations S_i , relation V from A and R. Denotational semantics of composite operators, pointers, calls of procedures and functions is determined and monotonicity of the Sem_n sequence of the $\mathrm{Sem} \subseteq f(\mathrm{sem}_n)$ function is proved in methodical book [34] with the different examples practical program and more 10 years teaches the students of Leningrad University. About this theory professor R.I.Podlovchenko was read in MGU and Erevan University from 1957 until 2016.

Ershov A.P. in the report on the title of academician of the USSR (1986) and the all-Union conference "Technology of programming" (1987) [2, 3] defined the elements of the theory of programming technology (*TP*), including methods of *synthesis*, *Assembly and concretic programming*. Synthesizing programming is based on the method of evidence-based reasoning about program correctness. Assembly programming builds programs from existing (checked for validity) is prepared of fragments of programs (reuses) and their assembly into a complex structure. Concretizing the programming to ensure the system building on the universal model of some subject area.

Basic tenets of *the theory* TP of A. P. Ershov stated: "We must distinguish between the TP as a technological theory and as a particular way of organizing, creating, distribution and maintenance of software product (SP) and how the procedure individual for professional activity, developing program product (PP). Technology and methodology is always science, while the method is included in them part. Professional technology, production programming is of fundamental importance to the transferability and replication of PP.

The technology starts when it covers the life cycle of PP. TP is a set of methodological guidelines, institutional, and instrumental-technical means, information and software (SW) regulating the activities of people involved in the process of creation, distribution and tracking of PP. Final TP should cover the entire life cycle of PP;

– to promote methodologies that increase the level of validity, reliability and evidence-based practice programming on modern technical means in the form of workstations, united in a local network;

– to ensure control over production processes;

– to ensure the stability of PP in relation to the change of the technical means;

– to ensure the development of PP in connection with change of conditions of functioning of target systems;

– to use this product in other environments".

Thus, A.P. Ershov made a guide for the development of TP in Soviet conditions. In [2, 3] has identified three directions of development of the TP and its prospects.

"The *first* direction (organizational programming) 1975-1985. The programming language is not formalized. The transition from prototype to a software version not formalized in programming languages

–ALGOL, FORTRAN, COBOL, PL/1, Assembler etc [4-7, 33, 34]. The knowledge base is missing, product development – versioned.

The *second* direction (Assembly programming) 1985-1995 [6, 20-22]. The language of the specifications regulated. The transition from prototype to the commercial version of the regulated. Language development – this is a formal high-level language with modularization and integration ...".

The *third* direction - evidence based programming in 1995-2005 [22, 33-38]. Language development is formalized and contains a system of formal transformations required for proof of programs... Language programming combined with development language... Project database computers. Product development – evolutionary – adaptive...".

In the *final* part of the paper by A. P. Ershov said: "It would be useful to develop a standard for second-generation technologies, which, without affecting the specific methodological or linguistic content, standardized to: General stages of development of PP; standards; performance and reliability; the structure and documentation of computing environment; inter-module interface support *Assembly programming*...».

All of TP fundamental bases of Ershov developed in the USSR in the author's works 2016 [1, 2, 20-22].

Subsequently, the *method of Assembly* of the modules and interfaces (1982) [4-6] was universal for all system-wide environments (IBM, MS, Intel, etc.) and was defined as the standard configuration build - ISO/IEC JTC 1/SC 7 Configuration etc.

It should be noted that was developed by the programming system “PRIZE” (E. H. Tyugu, M. I. Kahro. “Instrumental programming system on OS ES”.– Finance and statistics, 1981) for program synthesis based on semantic domain model and descriptions of the programs in PL/1, FORTRAN, Assembler [33], etc. The statement of the problem operator was given in the form:

$$\text{on z compute } x_1, x_2,\ldots, x_k \text{ from } y_1, y_2,..., y_m,$$

где $x_1, x_2,\ldots, x_k$, $y_1, y_2,\ldots, y_m$ - имена понятий, описанные в семантической модели z, а x_i и y_j - входные и выходные переменные задачи этой модели [33]. In this systems was given the *synthesis method* which based on substituting the semantics of implementation of the synthesized program.

Composition programs (Red’ko V. N.) is the operation of combining functions and data type: "data–feature–name and function-composition-description" the set of named data, descriptions and denotations (meanings). The composition operation is a subclass of the standard compositions and composite functions. They provide function composition on the level of LP. (Compositions of programs and programming // Programming. – 1978.

Further development of this method is *nominative programming* with nominative data. The class of nominative data provides building named data, nominative data in a multivalue or multi named data. Under this programming developed new tools for defining system data functions and compositions of the nominative type, the names of the arguments which belong to the set of names Z. the Composition is determined by the Z-nominative sets of registered functions (Nikitchenko N. Compositional–nominative approach to the clarification of the concept of the program, Probl. of programming. –1999. –No.1; Nikitchenko N.C., Chkilnayk S.S. “Mathematic logic and algorithm theory”, VSH of KNU, 2008 [40].

Assembling modules [6] based on the interface that specifies the link modules to each other (1975-1982). Interface – intermodule, interlanguage and technological (Interface-SEV, 1987). The *intermodule* interface is a module proxy (stub, skeleton) between two communicating modules to exchange data. *Interlanguage* interface defines the methods convert the transferred data types of PL OS ES with help functions transformation type data of modules in library of 64 for all data type PL [6, 19, 20 and 22].
Then various theories of program description in the USSR are considered.

Theory of discrete, information and program systems

Another kind of theory of programming is the theory of algebraic and algorithmic programming (Glushkov), is based on algebraic mathematical apparatus for specifying operations on program elements. Academician Victor M. Glushkov developed a camera operator schemes programs, in terms of the theory of equivalence of discrete transducers computers. The basis of this theory is χ finite state machine of a Mile with an input alphabet *X* and output alphabet *Y*, with the given initial and final States. V. M.

Glushkov considered a Moore automaton Gm (infinite) set of States *G*, the inputs *X*, outputs *Y*, the initial state e, the output function m(g) and a transition function $q(g, y) = gy$ [27-31].

Automatic χ, working in conjunction with gm, is called the discrete Converter. If the machine χ as input takes the output of *Gm*, the output χ is defined. The output of the automaton χ corresponds to the state of Gm at the time of stop of χ. Discrete converters of the semigroup is equivalent relative to G if for every mapping m from *G* to *Y* both do not stop with the work of the *Gm* or both are stopped and have the same output. The problem of equivalence of discrete transducers is solvable for the semigroup with left reduction and an insoluble unit in which the solvable problem of the identity of words.

It is open to all solvable and unsolvable cases of equivalence of discrete transducers with respect to the commutative semigroup. This theory was used in the construction of series computers MIR 1-3 [30] and bio computer to day "50-year Institute Mathematic Computer and Systems".-K.: 2010.- 528p.

The theory of discrete systems

When solving difficult mathematical problems of discrete systems, a new general mathematical language, called Concerting Language (LC) [41, 42], which based on the theory synthesis of automata and the algebraic programming [27, 28]. LC was allowed to give a formal description: the summation of infinite series, to perform multiple operations with infinite sets, Gilbert operator, and so on.

LC is a multi-sorted logical-mathematical expression language *X*, you can specify by using the objects and types.

Type is a means of creating expression and structuring of sets of values of denotations. The expression consists of terms and formulas.

Baths are objects of the subject area, and formulas are statements about objects and relations between them. The formulas are described using four categories of concepts: functors, predicates, connectors and subsectoral.

A functor is a constructor that transforms terms to terms (arithmetic and algebraic operations on numeric sets). *Predicates* that transform the baths into the formula.

Connectors include logical connections and quantifiers to transform one formula into another.

Subsector (descriptor) is a constructor for the formulas of terms and expressions that contain formulas on numerical sets and real functions (tuples, relations, family, work sets, etc.).

In *LC* almost asked the logical-algebraic specification of recognition tasks in dynamic environments hydroacoustics, radar, and other discrete systems. In particular, *LC* were used in the creation of technical facilities of new technology.

Discrete system (DS) contains a finite set of inputs, outputs and states. Its functioning is determined by the set of partial mappings, which are part of the signatures and form a partial algebra state S of the system. If the bill is to replace Boolean functions, we get the characteristic function of relations. The semantics of logical and algebraic specifications were developed based on term rewriting and proof theory theorems. The language used in the development of a new machine "Ukraine" in the Institute Cybernetic of Academy of Sciences of the USSR in 80-ies of XX century.

Algebra algorithms

V. M. Glushkov (1957-1964) identified algebra and mathematical analysis as a means of modeling the parameters and properties of solutions of differential and integral equations and systems [27-30]. The basis of algebra was:

- physical model that includes the properties and characteristics of individual elements;
- mathematical model, which specified the dimension, settings, and operations data;
- formal description of mathematical models and numerical models;
- description of mathematical tasks using the Targeted language [31] and the language of the ANALYTIC [30], which included mathematical operations (+, x, $\cup$, $\cap$, /, $\Diamond$, $\oplus$), decimal integers, rational numbers and operation is identical transformations.

Theorem: if the expression of Q_1 and Q_2 belong to certain sub algebra *Q* that specifies the canonical form of *F* and *F* (Q_1) and *F* (Q_2) coincide, then Q_1 and Q_2 are equivalent. Language ANALYTIC of modeling of mathematical methods on the computers MIR 1-3 [35]. Description of engineering and mathematical task is done using mathematical operations and logical operators, and structures of mathematical analysis

and operations identical transformations canonical forms using the standard functions (trigonometric, logarithmic, exponential, etc.).

This language provides tools for analyses debugging, tracing, and carry out the numerical problems of computational mathematics. It includes elements of the targeted language E. L. Yushchenko (1957) [31], namely the operation of the theory of sets and relations. Variables are represented with letters; they corresponded to the cells of the machine. The content of some of the addresses were marked by pointer, the address of the second rank (this concept has entered into foreign languages). More than just the language used to describe programs translators for machines, Umshn, Ural, Dnepr, etc. The development of this language is: universal algebra (sub algebras, logic, polybasic algebra, etc.). In book [29] it was describes SAA (Systems of Algorithmic Algebras), which contains: a mathematical description of this algebra; algorithms for converting the program schemes by G. E. Tseytlin, formal languages and grammars; the theory of synthesis of automata by V. Gluchkova: methods of analysis and CM-formalisms by E. Yushchenko for the analysis of programming languages.

Automata-based programming

Machines are specified counts of the transitions to distinguish between vertices that introduced the concept of coding States. The peculiarity of automata-based programming is that transition graphs are used in specification, design, implementation, debugging, documentation and maintenance of programs [31]. Programming is done "through States "rather than" through events and variables", which allows you to better understand and specify the task and its components. The transition from graph representation to text representation is carried out formally and isomorphical using the switch construction (in C language) or its analogues (in other languages). Therefore, the automata-based programming style often referred to as "Switch-technology".

In [60] the model of automaton based on States and transitions is proposed. She mapped trace model LTS (Labelled Transition Systems) and a close automaton model Miles FSM (Finite State Model). In these models, synchronous transition means a stable synchronous state, which is obtained as a result of external influence. If the transition from one state to another occurs asynchronously, the input test queue is tested. Programming state machine is held in the notation of CSP - Communication Sequential Processes (B. Hoar). The automata structure of the system is given by a graph with marked vertices from 0 to n. We prove the isomorphism of two labeled graphs, give the definition of the memory required to represent the graph and trace the vertices of the graph passing into some synchronous or asynchronous States. In [61] an approach to solving problems on a graph by a group of automata is proposed.

The theory of information systems

In the last book "Basis of paperless computer science" (1982) V. M. Glushkov defined for IS, ASM and ASM TP principles [35, 36]:

1. A systematic approach to the analysis of management systems, structuring and highlighting their goals and criteria.
2. Decomposition of system and functional characteristics and properties of subsystems for different purposes (staffing, record keeping, monitoring, control, etc.).
3. Modeling of system elements, typifying the functions and objectives of the system and the creation of economic-mathematical model of search of design decisions and building system choices.
4. Adding new tasks for the improvement of the organization, improvement and introduction of new management functions. (Monitoring of the information systems, business graphics, document exchange, etc.).
5. Define the external tasks (analysis, accounting, control) and internal (collection, registration, storage, search, etc.) with the aim of making management decisions. Based on the formulated principles of the system of electronic document management and control by the government based on models of the documents, their sizes and characteristics.

Model of information systems (IS) documents [44] include:

1). *Characteristics* of the volume, which includes the regular part of a sequence of repeated groups of data fields and irregular part without repeat data structures. The volume of documents was determined by the formula:

$V=l_h+n_s k_s l_s^{\max}$ - medium;

$V_{\max}=l_h+ n_s^{\max} k_s l_s^{\max}$ is the maximum, where l_h is the size of the irregular part of the document;

n_s – the number of rows of this type of documents;

k_s – coefficient of filling;
l_s^{max} is the maximum regular size of the document.

2). *Features run-time comprising*:

- the total values of processing time of different types of documents in accordance with their route;
- the execution time of certain operations on documents in different nodes of the system;
- transferring documents between different nodes of IS.

This theory was tested in [44] and applied in the education system of Ukraine.

Graphic R-style descriptions of the programs

R-style is a graphic description of the processes of program construction (Velbitskiy I. V.). The transfer graphic of R-schemes in the linear form for presentation in a computer to implement the relevant programs in different PL for the translators of the complex RTK [43]. R-diagram depicted with the symbols >, -, | and "o" letters. But as in the 70 years of the last century was not graphic displays, the description of the scheme was carried out by means of signs of the alphabet, and it was difficult viewing and analysis. Technological systems RTK was created on the machines BESM-6 and SM computers. The R-diagrams were used in the various military organizations of the USSR for word processing and creating an interface with existing software systems. In the framework of a Unified system of software documentation (ESPD, GOST 9126 quality) is represented by the standard GOST R-technology. However, after the collapse of the Soviet Union, these funds are rarely used, as it is not implemented in the class new computers.

Modular theory of programming

The term module appeared in connection with the transition to the EU computer (1976), which were implemented programming systems with languages ALGOL, FORNAN, PL/1, COBOL, ASSEMBLER. Each language was independent of each other in form and content. Program in any of these languages got the name of the module. For the organization of communication of multilingual modules, an interface was formulated, which described the General data and data transfer operators from one module to another [4]. The method of modular programming was developed, which provides the formation of a modular graph structure, by which the Assembly of individual program elements from modules and interfaces was carried out.

The module is the basic software element with properties [4-6, 20, 22]:

– the logical completeness of function;
– the independence of one module from the other;
– the replacement of individual module without disturbing the structure of the program;
– call other modules and return data to the caller module;
– the representation of modular structure by a graph.

The module converts the multiple input data *X in* a variety of output *Y* and is given as a mapping $M: X \rightarrow Y$. Communication between modules:

– Relationship management ($SR = K_1 + K_2$);
– Connection according.

Modular graph $G = (X, Y)$, where X is a set of vertices.
G is a subset of the direct product X x X x Z on the set of arcs.
The graph G is represented as adjacency matrix, symmetric about the main diagonal. This matrix is used to prove the attainability of one of the vertices xi to another x_j.
A modular structure is a pair $S = (T, \chi)$, where

T – model of the modular structure;
χ - the characteristic function defined on the set of vertices X of a graph of modules graph G.

The value of the function χ is defined as:

$\chi(x) = 1$ if the module with vertex $x \in X$ included in *SR*;
$\chi(x) = 0$ if module with top $x \in X$ is not included in the *PS* and it's not referenced from other modules.

The graph of the modular structure is represented by the adjacency matrix m_{ij} and the call vector $V(i)$, by which the ratio of the reachability from the i-top of the module m_{ij} to the j-module is established with the proof of the correctness of the data transfer by the call vector V.

Definition 1. Two models of modular structures $T_1 = (G_l, Y_1, F_1)$ and $T_2 = (G_2, Y_2, F_2)$ are identical, if graphs $G_1 = G_2$, modules $Y_1 = Y_2$, functions $F_1 = F_2$. Model T_1 is isomorphic to T_2, if $G_1 = G_2$ between the modules Y_1 and Y_2, there exists an isomorphism φ, and for any $x \in X$ $F_2(x) = \varphi(f_1(x))$.

Definition 2. Two modular structures $S_1 = (T_1, \chi_1)$ and $S_2 = (T_2, \chi_2)$ are identical if $T_1 = T_2$, $\chi_1 = \chi_2$ and modular structures S_1 and S_2 are isomorphic if T_1 is isomorphic to T_2 and $\chi_1 = \chi_2$.

The module is described in PL and has a description section of the passport, which specifies external and internal parameters. To pass parameters to another module, use the Call (...). The parameters may be converted to the form of the calling module and back in case of differences of their types. The library of 64 primitive functions convert of dissimilar data types (TD) PL is developed [46-51] for IBM-360.

Theory of Assembly of the modules

Method based on the interface that specifies the modules to each other and exchanging data (1975-1982). Interface – intermodule, interlanguage and technological (Interface-SEV, 1987). The *intermodule* interface is a module proxy (stub, skeleton) between two communicating modules to exchange data. *Interlanguage* interface defines the methods convert the transferred data types of PL using algebraic systems and 64 functions of the library interface [19-22]. The formal conversion of Data Types (TD) objects of the Assembly is performed using algebraic systems for each data type *t*:

T_α^t: $G_\alpha^t = <X_\alpha^t, \Omega_\alpha^t>$, where

t - the data type *b, c, i, r, a, z, u, e*;

X_α^t - a set of values for variables of that type;

Ω_α^t - a set of operations on these TD.

For simple and complex TD modern PL built classes of algebraic systems:

$\Sigma_1 = \{G_\alpha^b, G_\alpha^c, G_\alpha^i, G_\alpha^r\}$,

$\Sigma_2 = \{G_\alpha^a, G_\alpha^z, G_\alpha^u, G_\alpha^e\}$.

Systems Σ_1 and Σ_2 the transformation $t \to q$ for the pair of languages L_t and L_q have the properties:

1) G_α^t and G_β^q – isomorphic to q and t defined on the same set;

2) X_α^t and X_β^q are isomorphic if Ω_α^t and Ω_β^q are different. If $\Omega = \Omega_\alpha^t \cap \Omega_\beta^q$ is not empty, then there is a isomorphism between $G_\alpha^{t'} = < X_\alpha^t, \Omega >$ и $G_\beta^{q'} = < X_\beta^q, \Omega >$.

3) Between the sets X_α^t and X_β^q may not be isomorphic matching, then build such a mapping between X_α^t and X_β^q that it is isomorphic.

Theorem 1. Let φ – displays the algebraic system G_α^c to G_β^c. In order to φ be an isomorphism, it is necessary and sufficient to φ isomorphic reflected X_α^c and X_β^c, preserving linear order.

Each class of systems transformation $t \to q$ for the pair of languages *lq* and *lt* of PL may be some properties of mappings:

- systems G_α^t and G_β^q are isomorphic if their *q, t* are defined on the same set TD;
- between the values of X_α^t and X_β^q of data types *t, q* there is an isomorphism if the set of operations Ω_α^t and Ω_β^q are different;
- if the set $\Omega = \Omega_\alpha^t \cap \Omega_\beta^q$ is not empty, then we have the isomorphism of the two systems $G_\alpha^{t'} = < G_\alpha^{t'} = < X_\alpha^t, \Omega >$ и $G_\beta^{q'} = < X_\beta^q, \Omega >$.

If data types are different, for example, *t* - string, and type *q* is real, then there is no isomorphic correspondence between sets X_α^t и X_β^q.

The maps preserve the linear order of the elements based on the linear order of the elements of the algebraic systems of these classes.

Processing data types of PL

The data type TD is used in the description of PL programs (Algol, Prolog, PL/1, Fortran, Ada, Pascal, etc.). Axiomatic TD destroy developed by E.Dijkstra, N. Wirth, W. Tursky, P. Naur, A.Zamulin, N. Agafonov and others in the 1970. So on, which operate programs in PL include [22, 25]:

- *FDT* – Fundamental Data Type implemented using primitive functions (64) transform TD one LP to another and displayed in IBM-360 (1982);
- *GDT*- General Data Types (ISO/IEC 11404 GDT) for modern object-type LP (Basic, Java, Python, etc.);
- Big Data for Cloud Computing.

FDT and GDT data types - simple and complex.

Simple TD – integer (*i*), valid-real (*r*), boolean (*b*) and characterr (*c*) have a common form:
type $T = X\,(x_1, x_2, \ldots x_n),),$
where T is the type name, $X\,(x_1, x_2, \ldots x_n)$ - names of values from the set of values of type *T* of *X*.

Operations ($<, \leq, >, \geq, =, \neq$) or ($\leq$) determine the linear order of the elements of the set X. Operations on binary types include (true, false) and unary operations pred and succ, which define the previous and subsequent elements of the set X.

Complex TD-arrays, records, sets of Union, etc. The array m shows the set of indices in the set of values M: I → Y and has the form^

type T^{α} = array $T\,(I)$ of $T(\overline{Y}\,)$ задается в виде:

$$G^{\alpha} = < X^{\alpha}, \Omega^{\alpha} >,$$

$$X^{\alpha} = \{ x \,|(\forall\, x_1 \in X^{\alpha}) \,\&\, (\forall\, x_2 \in X^{\alpha}) \Rightarrow I(x_1) =$$

$$I(x_2)) \,\&\, (Y(x_1) \cup Y(x_2) \subset \overline{Y}\,(X^{\alpha}))\}, \quad \Omega^{\alpha} = \{ \leq \}.$$

In GDT TD uses simple and complex *TD*, based on the concept of cardinality and can be finite; accurate, infinite and approximate. A *Datatype* generator is an operation on one or more TD to create a new TD as a set of criteria for the characteristics of the TD using ;procedures and a set of operations in the final value space to define a new TD. *Aggregate datatype* is a generated TD, each value of which includes the names of the TD elements from the aggregate TD value space.

Operations of *TD generation* are carried out by means of operations with:

1) zero arity to generate the values of this TD;

2) unary operation (arity 1), which turns the TD value into a new value of the same TD or boolean value;

3) arity 2 that transform the pair of values of the TD in TD of the same value or a boolean.

4) *n*-arity, which converts an ordered *n* - group and TD values to a parametric type, or aggregate.

In the value space there is an order relation (order), which is given by the sign (≤) and satisfies the rules:

1) for each pair of values (*a, b*) from the value space the following condition is met $a \leq b$ or $b \leq a$ or both;

2) for any two values (*a, b*), if $a \leq b$ and $b \leq a$, then $a = b$;

3) for any three values (*a, b, c*), if $a \leq b$ and $b \leq c$, then $a \leq c$.

The GDT <–> FDT conversion system are based on these operations, implemented in [59] and displayed in .Net. It presents value type and reference type. Types-values are static types in STS (Common Types Systems), their values can take up memory from 8 to 128 bytes. Reference types use pointers to the objects they type, as well as mechanisms for storing in class library FCl (Framework Class Library) and releasing memory. Reference types include: object type, interface type, pointer type, etc.
In 1990 IK NANU is commissioned by the State USSR the processor for Fortran, Pl/1 and Assembler for ES OS that handles these languages based on the language syntax setting in tabular form. The build module calls other components of the translator to the programs of machine view (H. M. Mishchenko. "About Assembly programming of language processors". - Intellectualization of information and computing systems.– K: IK HANU, 1990).

The theory of the quality assessment of programs

The main feature of the military-industrial and an airborne system was reliability and quality. These issues are paid much attention to V.V. Lipaev [45, 63]. He's one of the first in the USSR has developed methods of ensuring the reliability and quality of such systems (Lipaev V. V. The Reliability of ACS the Software, Energoizdat.- 1981, Software Quality. Finance and statistics, 1983, 2008, etc).

Under his leadership created the GOST 2844-87 "Quality software". It defines the quality management system, including a combination of organizational methods of process control complex program (CP) on the stages of the lifecycle and the methodology for the calculation of technical indicators of quality in the course of life cycle for later use in the evaluation of reliability and quality in accordance with the requirements. Quality is the totality of technical, technological and operational characteristics of the CP, which are part of reference model quality, represented by six basic characteristics, whose values can be determined quantitatively or qualitatively.

On the stages of the life cycle, CP is the control of the individual quality indicators with special service quality and address emerging threats to the quality of the discovered defects in the CP. The

objectives of the service quality includes the planning and tracking process to deliver quality programs and systems at the stages of life cycle, qualification testing, test the trial version of the CP and assessment of the basic quality indicators based on the collected technical data of the design process of the CP. After the appearance of American quality standards ISO/IEC 9000 (1-4) and ISO/IEC 9126 (1-4) Life Cycle by V.V. Lipaev creates a "Handbook on software engineering" (2006) and the textbook "Software engineering a complex custom software" (2014). They described the methodology of designing high-quality *PS* with the accounting standards for Life Cycle ISO/IEC 12207-2007 and theory of quality assessment. These benefits can be used in the training course "Software engineering" (2012).

Evaluation of the reliability and quality of PS

The key characteristics of quality attributes is reliability and completeness as properties of the PS to eliminate failures with hidden defects with this criterion and a quality model, which relates the measures and metrics of the internal, external and operational type. From the standpoint of completeness of the product is the main indicator of quality are defects and failures [22, 40, 51, 63].

The model of defects based on multiple quality factors, analysis of causal relationships between them, combining qualitative and assessments of their impact on the density of defects. To calculate the *reliability function* from J. Musa on a formula:

$$R(t \mid T) = \exp(-(m(T+t) - m(T)))$$

where t - the operating time of PS without a failure when testing in a period of time T;

$m(T)$ is a function of reliability growth, as the average number of defects.

The function of reliability growth $m(t)$ is defined by the formula:

$$m(t) = N_0(1 - \exp(-\frac{\lambda_0}{N_0} \cdot t))$$

where N_0 – the number of latent defects in *PP* at the beginning of system testing

λ_0 – the failure rate of *PP* at the beginning of testing.

To assess the *quality* systems used the standard ISO/IEC 9000(1-4) quality model:

$M_{gua} = \{Q, A, M, W\}$, where

$Q = \{q_1, q_2, \ldots, q_i\}$ $i = 1,\ldots, 6$ – various quality characteristics (Quality – *Q*);

$A = \{a_1, a_2, \ldots, a_j\}$ $j = 1,\ldots, J$ – the set of attributes (Attributes – *A*), each of which captures a separate property of the q_i quality characteristics;

$M = \{m_1, m_2, \ldots, m_k\}$ $k = 1,\ldots, K$ – the set of metrics (Metrics - *M*) each element of the attribute a_j for the measurement of this attribute.

$W = \{w_1, w_2, \ldots, w_n\}$ $n = 1,\ldots, N$ are weight coefficients (Weights - *W*) for metrics of *M*.

The quality standard identifies 6- quality characteristics: q_1: functionality; q_2: reliability; q_3: using; q_4: efficiency; q_5: maintainable; q_6: portability. The quality of *PS* are assessed by formula

$$q_1 = \sum_{j=1}^{6} a_{1j} m_{1j} w_{1j}$$

These quality *qi of PS* are created the certificate on program products.

Paradigms programming for development systems

Paradigm (from Greek. παράδειγμα, "example, model, pattern") – a set of fundamental scientific attitudes, the concepts and terms adopted and shared by the scientific community. Provider continuity of development of science and scientific creativity. Thomas Kuhn called paradigms established systems of scientific views, in which research and development [46-51].

In SE emerged programming paradigm. It is a set of ideas, concepts, theories and methods that determine the style of formal presentation of computer programs. This term R. M. Floyd defined in his work "the Paradigms of Programming" (Communications of the ACM. 1969. V. 22 (8). P. 455-460), E. Dijkstra in the book "Discipline of programming" (M.: World, 1976) And D. Gris in the book "Science of programming". They defined it as a method of conceptualization and formal definition of programs and

systems. Some parts of the theory are implemented in the framework of applied (functional, logical, automatic, etc.), theoretical (VDM, OOP, Z, B, OCM, FODA, etc.), system (parallel, distributed, etc.) and commercial (Agile, SCRUM, EX) programming [10-26, 47-51].

Thus, the new theory of object-component modeling systems (OCM, 2012-2016) was created, which was recognized in Ukraine, Russian and abroad [20, 48-52]. The main author of OCM Lavrishcheva E. M. for the article "Ekaterina Lavrisheva, Andrey Stenyashin, Andrii Kolesnyk. Object-Component Development of Application and Systems. Theory and Practice//Journal of Software Engineering and Applications, 2014, http://www.scirp.org/journal/jsea" made Franklin Membership ID#513442 (UK) of London Journals Press 4 November 2016. The theory of paradigms – component, service and aspect were discrubed in book Software Engineering of computer systems. Paradigms, technology, Case-tools.- K.: Nauk. Dumda, 2014.- 284p. and in "Theory object-components modeling systems" (http://www.ispras.ru/lavrischeva/ru)

2. ABROUD MODERN THEORY AND METHODS PROGRAMMING

Methods mathematical specification of programs

The Vienna method – VDM. To formal methods specification include: VDM of D. Burner method, Z-method (Abril, I. R., B. Meyer), RSL and others [8-18]. These methods are using in many real projects.
VDM – the language of formal specification of programs and data by using mathematical symbols and the following data types: X are natural numbers with zero, N are natural numbers without zero, *Int* – integer, *Bool* – Boolean, *Qout* – character string, *Token* – characters and special notations of operations.
A function in VDM specifies the determination of properties of data structures and operations on them, applicative or imperative. In *the first* case, the function is specified using a combination of other functions and operations (through expression) that corresponds to the synonym functional. In *the second* case, value is determined by the description of the algorithm that corresponds to the synonym for algorithmic. For example, the function specification for computing the minimal values of the two variables has the form: $\min N_1\ N_2 \to N_3$.
Value description this function has the form:

min (x, y) = if x < y then x else y.

The objects of the VDM are data elements that operate functions that can form many, trees, sequences, display, and shape a new larger object. A set can be finite and denoted by *X*-the set.
Use operations $\in$, $\subseteq$, $\cup$, $\cap$ etc. to verify the correctness of the task of these operations. Distribution the Union of the subsets has the form: *union* ((1, 2), (0, 2), (3, 1)) = (0, 1, 2, 3).
Lists (sequences) is a chain of elements of the same type from the set *X*. Operation *len* specifies the length of the list, and *inds* is the number of the list item. Can also be used concatenation and this concatenation is distributive. Wood is the design is *mk*, which allows to combine complex objects of different nature (sequences, sets and mappings).

For example, let m_k – time (*h*, *m*) = t tin determines the value of *h* = 10, *m* = 30. The display is a design map that allows you to create an abstract table with two columns: keys and values. All table objects are of the same type of data – set.

When the specification of programs by means VDM are set pre - and post conditions, axioms and approvals are required for carrying out proofs of correctness of programs. The method of VDM is focused on step-by-step detail program specification. Initially built a rough specification of the model programs in the language of VDM, which is gradually refined, until the final text in the modern PL.

Algebraic specification languages Z

The *language Z* schemas specifies the description of the generalized model VM program in the form of control nodes model; the labels in the nodes of the graph model; the external characteristics and external parameters of the modules. This model is represented by a set of Z-schemes with a set of declarations and constraints that contribute to the formation of the state [10-18].
According to this model is:

– Selection of modules from a library of templates, renaming and concretization in the nodes of the graph;
– each module has an interface for a port tagged in Z-scheme;
– verification of communications modules through the description of the interfaces;

– create a new template and interface for the task of new developments in the scheme.

The interface contributes to the formation of the sequence of events when observing the behavior of executables. When the specifications of the module M is the description of the different situations that shape specific events, their analysis and definition of new events and conditions events.

Generalized model VM consists of two modules: the control unit CONT, and memory allocation STOR and has the form: VM = (CONT || STOR) \ {reguest, response} = (coin --> choc --> VM). In this module CONT sets the following specification: CONT = (coin --> reguest --> response --> choc --> cont).

The general purpose of this specification is that the consumer can insert the data in coin to send the request memory module. This module gives you the answer to requests according to the following specifications: STOR = (reguest --> response --> stor).

Model VM determines parallel processing of modules, CONT, STOR, as well as communication with the processing environment model.

CLEAR specification of functions and relations determine the behavior and equivalence relation of properties of objects and operations on them. Specifications are the presence of descriptions of functions that support data abstraction by special means.

Algebraic programming (AP)

The theory of algebraic programming (AP) is developed and described in [28, 29]. It is based on a system of rewriting rules for the transformation and processing of algorithms for calculations. The rules of symbolic processing are included in the action Language and form a system of symbolic calculations. In the description of the algorithms, the operations of multi-basic data algebra, including groups, rings, fields, etc., are used. The rewriting Technique is used in proving theorems and symbolic processing of algebraic calculations. The AP System was used in the "ANALYTIC" system and forms the basis of Computer Algebra.

Proof Methods of correctness of programs

Formal mathematical proof of the programs is based on the specifications of the algorithms, axioms, assertions and conditions, called preconditions and Post conditions, which determine getting the correct result of some specific program [10-12, 4]. Preconditions are constraints on the set of input parameters and Post conditions on the output parameters. The pre - and post condition are specified by the predicate, which will result in a Boolean value (true/false). The precondition is true when the input parameters are in the range of permissible values of the function. The post condition specifies the formal definition of a correctness criterion of obtaining results.

It is true when the set of values satisfies the requirements that specify functionality. The proof is carried out using a set of assertions to verify the correctness of programs at the given points. If the statement corresponds to the end statement in the program, i.e. is the final approval and with the help of a postcondition is the final conclusion on full or partial program correctness. The most famous methods of proof are a method of B.Floyd, P.Naur [10-12], etc.

A method a recursive induction of Floyd applies to programs that are developed through decomposition of the task into several subtasks and each of them are formulated approval subject to the conditions of input and output points of the program, which is located between the input and output statements. The essence of the evidence – the truth of the fulfillment of the terms and statements in a given program.

The Hoare method of structural induction is based on axiomatic description of the semantics of the source programs in the form of axioms. For each label program is a rule of inference, which displays the values of variables.

Example of proof of the elements of the array $T[1{:}N]$ in ascending order in $T'[1{:}N]$ [4].

The input condition is set to the initial statement:

A_{start}: (T [1: N) is array & (T' [1:N] array of integers).

Output approval A_{end} is the conjunction of (2, 3) conditions:

(1) (T is array) & (T' is an array of integers),

(2) ($\forall i$, if $i \le N$, then $\exists j$ ($T'(i) \le T'(j)$),

(3) ($\forall i$, if $i < N$, then ($T'(i) \le T'(i+1)$).

If the assertion (1) is true, then is true and (2). That is if (1) assertion – A_1 is converted to A_2, then the theorem is: $A_1 \rightarrow A_2$.

If A_3 is the next point of conversion, then the theorem is: $A_2 \rightarrow A_3$.

And so on until the end of $A_{start} \rightarrow A_{end}$.

The final theorem is that the condition is true in the last point if it conforms to the truth of the output assertion: $A_{start} \rightarrow A_{end}$.

3. THEORY OF OBJECT-ORIENTED DESIGN SYSTEMS BY G. BOOCH

G. Booch introduced a new style of programming called OOP [13-15]. It defines the underlying mathematical concepts - object, class, polymorphism, inheritance, isomorphism, etc. Objects can be grouped into classes and subclasses. A class is a set of objects that have common variables, structure and behavior. The object is an instance of the class. The behavior of the instance is determined by the operations of creation, destruction, and serialization. A set of external variables and methods the class defines the interface between object of classes for interaction. For each external variable, there are methods you select (get-method) and assigning a new value (set-method). From a General point of view, the interface object instance consists of a set of methods. Each object can have many interfaces that define its functional properties. An object can have a special interface whose methods work with instances (Home interface in the EJB model to Java). That is, the class object has a special interface or one or more interfaces that are implemented in the component instances. On an abstract level, the interface can be considered as a partial class.

Proposition: Any instance of a particular class has all the methods defined in this class. The class instance may be provided in accordance with one of the superclasses in accordance with an interface that is implemented in this class.

Theory G.Booch allows you to design subject area, on the basis of the claim that the entire material world consists of objects. Any subject area is a collection of objects linked by some set of relations and behavior for some time: < object orientation> = <objects> + <inherit>.

Every concept of the subject area, together with its properties and a specific behavior is a separate object, and the whole area is a set of objects with relationships that are established on the basis of the relations between these objects. As the object of the act as abstract images and concrete physical objects or groups of objects with the specified common characteristics and functions.

Development of the OOP Application is made the UML [15]. In this process of building the system is at the stage of domain analysis and design. In the analysis process creates an object model (OM) in the form of diagrams precedents.

It clarifies the external functional behavior of the system. The frame is created of the designed system. To determine the behavior of classes of objects use State charts and activity. The placement of objects in OM that is fixed by the component diagrams in the nodes of the network computers that are deployed for execution.

Thus, the theory G.Booch is the basis of the modeling systems in UML, MDA, MDD, SOA, etc. this theory entered in PL (C++, C, Basic, JAVA, etc.) [47].

The Unified modeling language (UML)

UML is a language for designing systems using visual diagrams [15]: of use cases; classes; behavior; activities; interaction; sequence and cooperation; and implementation (component diagrams and deployment). The placement of objects in the development environment is set of component diagrams, and the location of the software modules in nodes (computers) of the network — a deployment diagram. Given the precedents (use case) or diagramms are used to design the following object models system:

1) Structural (static) model that specifies the structure of concepts of the system, including classes, interfaces, relationships, attributes;

2) Model behavior (dynamic), which specifies the behavior of objects, using interaction diagrams and state changes of components and systems;

3) Operation model specifies the control operation to the calculation of *PS*.

UML are a diagram language for modeling and documenting systems. The graph model of the system is given by a set of Use Case diagrams: use cases; classes; behavior; interaction diagrams; statechart diagrams; activities diagrams, implementation, etc. The model of the system is the architecture of the

system from ready – made objects of functions that are transformed to program elements in languages Pascal, Basic, JAVA and et al.

UML models from diagrams describes the structure of system. The elements of this model are implemented in the evolution environment.

The theory of modeling variability systems from ready-made resources (CRU)

The simulation is based on mathematical operations, building models of systems (domains) and models characteristics of MF (Model Feature) which displays the functional properties of the system elements and the ability to edit, delete, and replace with new. The elements of MF can be marked points of variability for the formation of the versions of systems in family systems and evidence of the architecture of the system by means of mathematical apparatus of the matrixes adjacency and reach ability [20, 47-51].

Theory modeling changeable systems and their families is based on the configuration of the Assembly of the finished PP firm SEI (Product Lines/Product Families) and the theory of control systems, subject to the requirements of the customer. Based on this theory, the analysis models of existing operating systems (Linux, Intel, etc.), real-time systems and models Web-based systems (SOA, SCA) by extracting the basic elements and generation options OS and Web-systems are developed methods of verification, transformation and Variability Mining ready Legacy Systems [50]. It was described the interfaces of the individual elements of the system in languages IDL, SDL, etc. and transformation the display of non-equivalent complex data to simpler data. These mechanisms are the foundation of the theory of transformation of unstructured TD to the modern fundamental and storing them in Big Data and solves problems in Cloud Computing [25].

Transformation TD of a set *PL*

The GDT<=>FDT generation scheme is based on a library of functions in XML [25]:

- conversion of TD PL_1, ..., PL_n;
- submission TD the FDT to the kind of special functions;
- convert the GDT to the form the FDT;
- equivalent GDT<=>FDT mappings.

The transformation theory is based on a set of functions:

1) displays all GDT (primitive, aggregate, and generated) to FDT TD (simple and complex) that allow components to interact with each other as in Grid;

2) specifications of external TD by means of GDT and their saving in storages and repositories;

3) specification of data formats of interface program intermediaries using GDT<=>FDT functions;

4) generate conversion programs and so on given platforms for their location. The library of functions of transformation TD GDT<= > FDT provides Assembly of heterogeneous programs which are described in modern *PL*.

4. SOFTWARE ENGINEERING METHODS AND THEORY

The basis of Software Engineering

Software engineering (Software Engineering – SE, 1968) is a systematic approach to the development, operation, maintenance, and termination of use of software tools (www.swebok.com). The core of the SWEBOK (Software Engineering Body of Knowledge) developed by international committees of the ASM and IEEE and is designed to teach software engineering Curricula 2001 (2004, 2007, and 2014). The composition of the SE include the methods, means and tools that provide quality and performance in the work of programmers during the software development and PS [51]. The appearance of the term SE in the USSR was preceded by the term Software Engineering which is meant methods, means and tools that facilitate creation of the PS. These definitions are so close, that in fact, SE can be interpreted as the further development of the Technology Programming (TP) in providing software engineering work methods (planning, accounting, control).

This development essentially means a transition from a single creation of programs by individuals to industrial production. In the SE lifecycle standard developed by the ISO/IEC 12207-1996, 2007, regulating the development processes of PP with desired properties (functions and quality). In the field of

SE, there are other types of work: reproduction of software and documentation; configuration and generation of programs; entering and monitoring data, etc. These types of works are standardized.
There are also specialized operations (debugging, verification, testing, etc.) in the framework of the ISO/IEC 12207. They can be performed in accordance with the technology development which is the core of the SWEBOK 10 knowledge areas that define lap knowledge in the field of realization of various purposes. For quality management PP created the ISO/IEC 2844-89 "Guality Assessment of software" (1987). It provides a large amount of routine work on the evaluation of individual properties and characteristics of PS with 50 criteria and factors 6 measurement standard qualitative and quantitative indicators program systems.

A scientific disciplines of software engineering (SE)

To these disciplines SE (2008) [7, 22, 46-51] are included:
Research – a combination of theoretical and formal foundations paradigms of programming (on classes, prototypes, agent-based, component, service, extremal and so on) and means of development of the different new software and application systems.

Engineering – a set of tools and methods of design based on the standard of life cycle the application systems, their verification and testing, correct the found errors and defects. Then, testing the system on test sets to find external errors and generate a quality certificate for the created product for evaluation function of systems.

Management – methods planning works for the manufacture of a team of separate elements of systems, risk analysis of failure of plans, verification and generation of product variant.

Economy – the set of methods of the expert, qualitative evaluation of the results of creating with the necessary calculations the total time, volume, complexity and cost of manufacture of the finished product. Estimation of labor costs and the cost of work in the development of the application and system by the method of COCOMO1, 2.

Industry is an industrial technology conveyor Assembly production system from finished software resources (CRU, Reuses) from the libraries and repositories on the user request.

Learning SE international programs Education -2000, 2010, 2025 (http://www.teachingbox, http://www.microsoft.com etc.) and on the national website http://7dragons.ru/ru.
These subjects are included in the International program of training - Curricula SE and Computer Science (2007, 2013).

SEMAT – Software Engineering Methods and Theory

In SEMAT (I. Jacobson, B.Meyer and R. Solley) the goal is to develop in SE new methods and theory to the development qualify as a rigorous mathematical discipline (idea, close to the ideas of A. P. Ershov 1986 [1, 2]). The aim is to continue on the creation of the theory and methods SE and bridge the gap between academic theory and professional community of developers of specific types. Working in SEMAT structured in four areas: Practice, Education, Theory and Community. *Practice* developing applied and practical work. *Education* addresses issues associated with training developers, students and professionals. The *theory* deals with developing a General theory for software and systems development. *Community* professionals who build websites and develop them to user needs. With time, Practice, Education and Theory will be integrated. Theory should guide research and create systems and application.

Basic methodology programming of SE

The program Curricula for teaching Computer Science (2014) includes courses CS1011 "Programming fundamentals"), CS102I "The Object-Oriented Paradigm", and event-driven (event-based programming), automation, Agile etc. This program focuses on the development of scientific basics of programming.
To introduce several programming paradigms developed formal apparatus in theoretical and applied design of individual software resources for building of the software system.
The programming paradigm is a set of ideas and concepts, theories, methods that determine the style of writing computer programs, software and information systems. Analysis of existing paradigms given in the Gorodnya L. P. and their classification is given [20, 22]:

- applied programming (functional, logical, automatic, etc.));
- theoretical programming (object, component, aspect, service, etc.));
- system-oriented programming (parallel, distributed, etc.) and others.

In [20] describes the theoretical programming of the paradigms (object, component, service, aspect and othes). Made in these paradigms, elements of software resources are described in PL, and their interfaces in standard WSDL. Propose the formal apparatus of OCM as programming paradigm. It formalizes the resource Assembly into complex programs and systems using the method of Assembly programming. This method provides a mechanism of interaction between resources of these paradigms in the new system or family [19-24, 54]. Application paradigms are: functional, agent, etc.

Functional programming

Functional programming is used to solve problems related to pattern recognition, implementation of expert systems, automated proof of theorems, symbolic calculations, etc. [21, 54]. Its basis is the theory of lambda calculus (A. Church) and combinatorial logic (M. Schönfinkel and H. Curry). A functional program consists of a set of function definitions that represent calls to other functions and proposals that control the call sequence. The description of the functions dependence on each other actually determines the way of the task solution. Recursion is the fundamental basis of program semantics.

The branching of calculations is based on the mechanism of processing the arguments of the conditional operator, and cyclic calculations are implemented by means of recursion. LP belongs to the functional:

- LISP (1958) and many of its descendants, the most modern of which are Scheme and Common Lisp;
- ML (1979) and its most famous dialects-Standard ML and Objective CAML;
- Miranda (1985) developed Haskell and others.

These languages are used in scientific (academic) applications, with the exception of the Erlang language (for developing applications in telecommunications), languages J and K (financial analysis), and domain-specific languages, such as XSLT.

Agent programming (AP)

The basis of the AP is an intelligent agent-an entity capable of formulating goals, learning, planning its actions and making decisions in dynamically changing conditions [49]. A simple example of the agent's task is to find the necessary information on the Internet, analyze and filter out unnecessary information. Agents are investigated within artificial intelligence and computer science. Under agent means a computer program, which are inherent properties: autonomy, reactivity, pro-activeness etc. the Agent can be considered responsive (reactive), interface, task, informational, and mobile (distributed) agent.

To agent-based architectures include: PRS, JAM, COSY, INTERRAP. Agents can form a multi-agent system to achieve common goals. In it, each agent has an idea of the model of the world, about himself (mental model) and about the agents with whom he interacts (social model).

Languages are the basis of AP:

- description of models (mental, social);
- specifications of information, temporary, motivational and functional actions of the agent;
- interpretation of specifications of the agent; conversion of programs to the agents.

The agent specification is refined, interpreted, and compiled into a computational representation of the agent program that can be run in a functional environment. The means of interaction of agents include coordination, communication, cooperation or coalition. The means of communication of agents include the transport Protocol TCP/IP, FIPA and the standard of the message language – ACL (Agent Communication Language). In practice, the agent can be implemented as a Java component, COM object, Lisp program, or description in TCL. APRIL and MAIL languages are used to create multi-agent systems. All modern tools for building multi-agent systems are divided into two classes – libraries (for example, JATLite to Java language) and environments (AgentBuilder).

Paradigms SE for commercial systems

In the field of software engineering specialists K. Beck, M. Fowler, K. Schwaber, J. Sutherland formed a non-profit organization Agile Alliance, promoting Agile Software Development, eXtreme Programming methodology, SCRUM, Dynamic systems Development Method (DSDM), Adaptive Software Development (ASD), Feature-Driven Development (FDD), Crystal, Pragmatic Programming [23, 24 and 54].

The Agile methodology is focused on the close collaboration of a team of developers and users. It is based on a waterfall model lifecycle incremental and rapid response to changing demands on PP. The team works according to the schedule and financing of the project.

eXtreme Programming (XP) implements the principle of "collective code ownership". It any member of the group can change not only your code but also code another programmer. Each module is supplied with the Autonomous test (unit test) for regression testing of modules. Tests written by the programmers and they have the right to write tests for any module. Thus, most of the errors are corrected at the stage of encoding, or when you view the code, or by dynamic testing.

SCRUM is agile methodology project management firm Advanced Development Methods, Inc., used in organizations (Fuji-Xerox, Canon, Honda, NEC, Epson, Brother, 3M, Xerox and Hewlett - Packard etc.) are based on an iterative lifecycle model with well-defined development process, including requirements analysis, design, programming, testing (http://agile.csc.ncsu.edu).

DSDM (Dynamic Systems Development Method) for rapid development of RAD (Rapid Application Development) prototyping and ASD (Adaptive Software Development) for extreme projects and is based on the theory of complex adaptive systems. *FDD* (Feature Driven Development) methodology focused on functionality for a major Bank (www.nebulon.com) and is model-driven (model-driven) process.

Theoretical paradigms are: functional, agent, automaton, etc.

5. THEORIES PARADIGM PROGRAMMING. NEW OBJECT-COMPONENT MODELING SYSTEMS

Theory of object-component programming

To introduce several programming paradigms developed formal apparatus in theoretical and applied design of individual software resources for building (configuration) in the software system. In [26] describes the theoretical foundations of the paradigm of object, component, service, aspect and generating programming. Made in these paradigms, elements of software resources are described in PL, and their interfaces in standard WSDL. Propose the formal apparatus of the object-component method (OCM) as programming paradigm formalizes the resource Assembly into complex programs and systems using the method of Assembly programming. This method provides a mechanism of interaction between resources of these paradigms in the new system or family [20, 47, and 48].

A new paradigm of object programming in OCM

This paradigm is based on the logic-algebraic apparatus of modeling the graphical object model of the domain at four levels (generalizing, structural, characteristic and behavioral). The first level defines a set of domain objects On the second level are established, relationships between objects in the graph vertices with arcs (fig.1). The third defines the interfaces of objects as elements of the graph. On the fourth - states of objects on a set of predicates depending on their execution time [20-29, 47- 51].

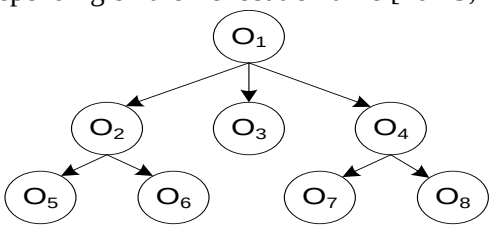

Fig.1. Structural graph G

The graph G has the properties:

- the vertices of the graph specify one mapping of objects to sets of objects;
- each vertex has at least one link with other nodes of G;
- there is one vertex O_1 of graph G, which shows the subject area as a whole.

Each level allows to drill down and refine the system objects in a graph and their relations on the set of domain objects, with a gradual clarification of their denotations and concepts of the theory of Frege and displays them in model MF, using mathematical and logical operations ($\cup$, $\cap$, /, $\Diamond$, $\oplus$, –, &, V...). It is defined by their external characteristics MF, in which vertices are objects and the arcs specify their interface to transfer data between them.

The constructed graph G is adjusted is restructured by adding new objects and interfaces by operation (union, delete, replace, etc.). Then it turns out the extended graph. Thus, a graph G = (O, *I*, *R*), where *O* - a set of objects (functions); *I* - many interfaces and relations *R* (relations defined by the arrows in the graph) between the objects (Fig.2).

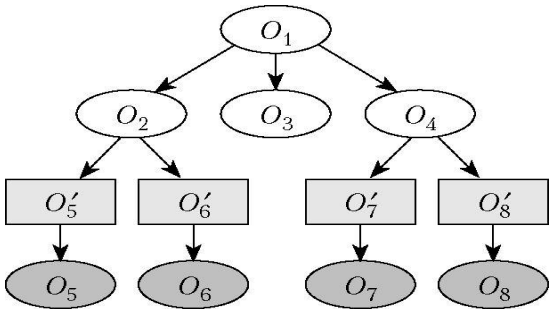

Fig.2. A graph G on the set of objects and interfaces

The objects of the extended graph G are transferred to the software components that are immersed in the component environment. The elements of these models are collected in repositories, component-based environment and can be configured in different variants of the PP SP. At the vertices of a graph G are functional objects O_1, O_2, O_3, O_4, O_5, O_6, O_7, O_8 and interface objects — O'_5, O'_6, O'_7, O'_8. All objects are contained in repositories, and arcs correspond to relationships between all kinds of objects. The graph elements O_1—O_8 are described in PL, and front-end objects O'_5— O'_8 in the language IDL.
The parameters of the external characteristics of the interface objects are passed between objects through interfaces, and tagged them *in* (input), *out* (output), *inout* (input and output) in the language IDL.

For the graph *G* in Fig. 2 the set of programs P_0 — P_5 defined in a mathematical ∪ join operation (Assembly) link:

1) $P_0 = (P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5)$
2) $P_1 = O_2 \cup O_5$, *link* P_1 =*In* O'_5 $(O_2 \cup O_5)$;
3) $P_2 = O_2 \cup O_6$, *link* P_2 =*In* O'_6 $(O_2 \cup O_8)$;
4) P_3;
5) $P_4 = O_4 \cup O_7$, *link* P_4 =*In* O'_7 $(O_4 \cup O_7)$;
6) $P_5 = O_4 \cup O_8$, *link* P_4 =*In* O'_8 $(O_4 \cup O_8)$.

The result of when two objects of the count (example, $\boldsymbol{O_{25}}$ и O_{47}) is an interface object O'_5 , O'_7 in which a multiplicity of interfaces coincides with the set of interfaces the object receiver and many source interfaces with multiple input interfaces of the object transmitter.

Axiom. Advanced graph G with interface objects structurally ordered (top), controlled completeness, redundancy and the lack of duplicate elements.

Worked out formal mechanism of transition from objects OM to components and interfaces of CM and component programming were defined by formal models (component, interface, component environment), by the operations of external and internal component algebra with facilities of transformation of unrelevantic types data.

The paradigm of component programming

The concept of this paradigm was proposed in article [4, 19, 29] and realized by V.N. Gerashchenko in project of the informatisation NANY and was described in dissertation “Theoretical and Applied Methods of Components Programming” (K.: IK NANU, 2007). He was died in 2009. This concept was continued in other works [26-29, 47- 49]. On this paradigms design the component models (*CM*) of the system which include: model of component - *Comp*; interface model; model of the component environment *CE* and *CM*.

The CM obtained the formal elements [20, 23]:

CM = (*RC*, *In*, *Imp*, *Film*), where

RC - the basic elements from the set components (*C*) and the OM;

In - the interface of components with variation points;

ImC - implementation of the base component in the environment;

Fim (·) - functions for converting interface and data parameters in the interface signature.

OM and *CM* are verified on the correctness of the inclusion of components *Comp*.

The model of component:

MComp = (*CName*, *CIn*, *CFact*, *CImp*, *CServ*), where

CName is the unique name of the component;

CIn = {CIni} - the set of interfaces of the component;
CFact - managing instance instances;
CImp = {*CImj*} is the set of implementations of the component;
CSer = {*CServ*} - a lot of system services.

The set *CIn* = *CInI* ∪ *CInO* - input *CInI* and output *CInO* interfaces.

The Interface model: *CIn* = (*InName, InFun, CIn, InSpec*), where
InName is the name of the interface;
InFun - interface functionality;
CIn - interface for managing instances of the component;
InSpec - specification of the interface (descriptions of types, constants, methods, etc.).

The interface specifies the operation of managing instances:
Cln = {*Locate, Create, Remove*}, where
Locate - search and determine the instance of the component;
Create - create an instance of the component;
Remove - removes the instance of the component.

Model of the component environment:
CE = (*NameSpace, InRep, ImRep, CSer, CSerIm*),
where NameSpace - set of environment component names;
InRep = {$InRep_i$} - the repository of interfaces;
ImRep = {$ImRep_j$} - repository of implementations;
CSer = {$CSer_k$} - a set of system services;
CSerIm = {$CSerIm_i$} - a set of implementation of the services.

A component environment is a set of application servers where components, containers, and instances that implement component functions are deployed.

Thus, based on the CM model, the components of the *PS* can be distributed among the network nodes and interact with each other through the interface and messages.

The object and component algebra:
θ = {φ1, φ2, φ3}, where
φ1 = {*CSet, CESet*, Ω1} is an exterior algebra;
φ2 = {*CSet, CESet*, Ω2} is an inner algebra;
φ3 = {*Set, CESet*, Ω3} is the evolution algebra;
φ4 = {*Set, CESet*, Ω3} is the assembling algebra.

The exterior algebra φ1 = {*CSet, CESet*, Ω1}, where
CSet is the set of components;
CESet is the set of component media;
Ω1- set of operations;
CSet2 = *Cset* ⊕ *CESet1* - installation (deployment);
CESet3 = *CESet1* ∪ CESet2 - association of media;
CESet2 = *CESet1* - *CSet* - removes the component from the environment.

The inner algebra φ_2 = {*CSet, CESet*, Ω2}, where
Ω2 = {*addIm, addIn, replIn, replIm*},
addim - operations of adding an implementation *C*;
addIn - adding an interface;
replIm - replacement operation for the implementation;
replIm - interface replacements.

The evolution algebra Ω3 = {*CSet, CESet*, Ω3},where
Ω3 = {*Orefac, OReing, ORever*},
Orefac = {*AddOImp, AddNIm, ReplIm, AddIn*} - refactoring operations;
Oreing = {*rewrite, restruc, adop, conver*} - reengineering operations;
ORever = {*redesign, restruc*} - reverse engineering operations.

The assembly algebra φ4 = {*CSet, CESet*, Ω4},
where Ω4 = {incon, redev, linkconf, makeaw, add, insert, redo} - operations for assembling and interacting components with conversion of data types formats transmitted between the reuses (CRU),

using primitive generation functions of GDT types ISO / IEC 11404-2007 to the fundamental types of data of PL (and vice versa).

At an exchange data *o*f every pair of components of are correcting are after next functions represented:

FN_{ij}: $N_u \rightarrow N_j$, FT_{ij}: $T_i \rightarrow T_j$, FV_{ij}: $V_i \rightarrow V_j$,

where FN_{ij} - inflicts accordance between ім'ями of variables on the great number of formal and actual parameters; FT_{ij} is description of equivalent reflections of types given; FV_{ij} will realize necessary transformations of values of noequivalent types of data.

The task of construction of transformations of FN_{ij} decides by bringing organization over of the names variable of Reflection between the types of data of FT_{ij} given as an abstract system of algebra T = (X, Ω), where X is a great number of values for this type, and Ω is a great number of operations above these variables. the reflection of FV_{ij} is used in case of noequivalent of types of T_i and T_j.

Under transformation $T_i = (X_i, \Omega_i)$ to type $T_j = (X_j, \Omega_j)$ understand such transformation at that semantic maintenance of operations from Ω_i equivalent to maintenance of operation from Ω_j. In case one-sided alterations as T_i in the type of T_j the equivalence of transformation does not exist. More about transformation data types look in [25].

Service programming (SOP)

SOP (service-oriented architecture) - development of a component approach based on the use of services with standardized interfaces. They can be distributed across different network nodes .Their interface provides encapsulation of the implementation details of each component. SOP provides a flexible way to combine and reuse components to build complex distributed software systems and enterprise software applications [41, 51, and 62]. The systems are implemented as a set of web-services integrated with standard Internet SOAP languages, WSDL W3C (www.w3.org) et al.

Web services are a new promising architecture that provides distribution at the Internet level [49]. Thanks to web-services, functions of any program in the network can be accessed via the Internet, and the results of accessing them – using PHP, ASP, JSP-scripts, JavaBeans, etc. An example of a web-service is the Passport system on Hotmail, which allows to implement user authentication. Web services are based on standards, open exchange protocols, data transfer in this order of action:

- definition of the format of requests to the web service and its responses;
- any computer on the network makes a request to the web-service;
- the web service processes the request, performs the action, and then sends the response.

The difference between web services and other technologies (e.g. named pipes, RMI) is that they are based on open standards and are supported on all Unix and Windows platforms, etc. A SOAP envelope contains a request to perform an action or a response. The envelope and its contents are encoded in XML and sent via HTPP to the web service. The problem with using web services is to find them. IBM, Microsoft, and Ariba have come up with an initiative project for Universal Description, Discovery and Integration (UDDI) to provide a common catalog of web services to enable all companies to "publish" their web service. This directory works as a phonebook for all web services.

Variability systems and model MF in OCM

The objects of the graph G form a model of the system under system configuration. Elemental which can be changed in the graph are labeled by points of the variance (variability) [22-26].

The *point of variance* in one place in the model system PS, which selects the variant of the system. A point of variance are handled by the Configurator and allows you to transform the prepared system by replacing some of the components used reuse (CRU) by other more functional or correct.

Variability – a property of a product (system) to expand, change, adaptation, or configuration for use in a particular context and ensure its subsequent evolution (ISO/IEC FDIS 24765 - 2009 (E).

Models with variable elements MF (Model Feature) used in product configuration of the finished *CRU* or taken from the libraries of reuses.

Model of variability PS*:* MF_{var} = (*SV, AV*), where

SV – submodel of variability of the artifacts of the structure of the PS;

AV - submodel of variability of the developed PP.

The MF_{var} model provides a level of variability of the artifacts and products of the PS reduces costs and reduces the development time of the product PS.

SV = ((*Gt, TRt*), *Con, Dep*), where *Gt* = (*Ft, LFt*) – count of artifacts of type t (requirements, components, tests, etc.);

Con and *Dep* – the predicates on the Cartesian product of the sets of artifacts that define the constraints and dependencies between the functions and indicators of the quality of the PS.

The AV submodel determines the structure of the PS from the CRU, which have the passport and stored in the repository.

Submodel *SV* displays functional and vibrational characteristics of CRU and product, as well as aspects of the relations between them. The *SV* model specified in the Line/Product and Assembly in PS.

Configuration model of system

The models systems use the artifacts (reuses, object, services, and components) that transformation software view using this model M_{konf} [22, 52, 57]:

$M_{konf} = (OM, M_{SD}, M_{ps}, MF_{var}, M_{IN})$,

where M_{in} – a model of interaction of individual elements of the created system; M_{sd} – model of domain; MF_{var} – model of variability; M_{IN} - a set of input parameters.

Based PS on the model M_{konf} are doing:

- selection of artifacts and resources of the PS in the base configuration of a given system;
- allocation of common and variant characteristics of the *PS* in the model FM and model of the *PS*;
- planning for multiple resource use for PS in the points of variability and their fixation for their removal replacement;
- build resources in the *PS* and their adaptation to new conditions of environment;
- management options for PS with the replacement of individual functions in the *PS*;
- manage the interaction of artifacts in a heterogeneous environment.

Realization model *OCM* realized on the website http://7dragons.ru/ru. Site database forms a repository of ready resources, models *PS*, variability and interaction of system-wide tools - Visual Studio, Eclipse, CORBA, IBM WebSphere [47-56]:

1). Visual Studio.Net↔Eclipse defines the environment of the interaction of individual elements in the C# language and interface. The model establishes the relationship of the elements with a given environment via the config file.

2). CORBA↔JAVA↔MS.Net provides communication between these environments with specified in these languages, the elements to access by other developers.

3). IBM Sphere↔Eclipse provides communication between programs in PL in these environments.

Verification and testing of various systems

Variability of MF can be checked for correctness with the help of solvers and tools for the automatic proof of various types of models (SAT, BDD, etc.) described in the formal language Alloy, B or Z [22, 47], and using Model Checking applicable to MF with a finite number of States described by formal specifications. While in the method Kripke model MF is defined formally as:

$$M = (S, S_0, R, L),$$

where *S* - the set of states, S_0 — the set of initial States, *R* - the ratio of transitions $L : S \to 2^{AP}$, x_0 — function marking. This model is described using the language of temporal logic with assertions the truth of which is verified by the verification.

Approach to the constraints (Constraint Satisfaction Problem) applied to the model variability, if it set the conditions for executing the restriction. Another way of verification are ontology based on broadcast models of *MF* to model the ontology described in the OWL DL (Ontology Web Language Description Logic). After the broadcast, the description in this language uses an automated tool RACER [47. 57-59].

Testing of the finished product created by the configuration is carried out using a set of tests for individual items of PP. Method John. McGregor [53] "from requirements" (requirements-based testing) is carried out using tests that verify functional and interface objects. As a testing tool used framework Visual Studio 2010 tools validation testing of different types of objects. It includes the Test Manager, which manages the planning of the testing process and run test scenarios. An appeal to the framework of testing produces the website http://7dragons.ru. When errors are detected during testing shall be corrected in MF and PS. Then there will be a new configuration and getting PP with the addition of new objects or removal of old.

Evaluation of quality characteristics PP

In the field of quality G. I. Koval [54-59] developed new formal methods of measurement and evaluation of characteristics quality PP in the class of tasks of SOD, namely:

- A model of quality with an orientation on the estimation of reliability of PP;
- A model of distribution of fail safety from components after the functions of utility of PP:

$$Q_{nc} = \sum_{j=1}^{l} v_j^* \cdot q_j = \sum_{s=1}^{m} w_s^* \cdot r_s$$

depending on the ponder able coefficients of w_s and reliability $q_j = \prod_{n \in E_j} r_n$ of separate components;

- conceptual model of making decision from a quality management, including Bayes methods and methods of systematic control of reliability on the early stages of LC, measuring of quantitative requirements to reliability and prognostication of defects.

The results of researches from the problem of providing of quality of PP are systematized by the collective of authors in the monograph of "Foundation of engineering quality of the PS" [54, 63].

CASE – tools of paradigm programming

As a means of realization of life cycle processes ISO/IEC 2007 elected the language Device and the DSL Tool VS.Net etc. In them ontological description transformed to the XML, which is the implementation language of the marked features of the Life Cycle domains and define the communication and data exchange between them. The paradigms of programming of OCM (objective, component, service, generation) are realized too. With the participation of the students was developed a program of processing FDT and GDT, a variant of the ontology Life Cycle using the tools, DSL Tools VS.Net and Protege [22, 59] etc. They are available on the website http://7dragons.ru/ru.

This website provides the manufacturing of ready PS CRU, build and test objects in environments (VS.Net, Corba, Java, Eclipse). It is a link to a website of programs of KNU http://programsfactory.univ.kiev.ua. It accumulates the scientific artifacts of the students of KNU. The website also includes courses in Java, C#, VS.Net and "Software engineering" for training students of KNU and MIPT. The site was turned over 150,000 of the different specialists.

6. MODERN THEORY OF PROGRAMS FACTORIES

A new conception factories

Conception of factories of the programs was first set forth by an academician В.М.Глушков in 1975р. Her aim was to accelerate a transition from the art of programming to the industrial methods of production of PP for the decision of various pertaining to economy tasks within the limits of the new computer-assisted system OGAS. Foundation of factory are being a technological line (TL) or product lines (PL), which are a well-organized set under processes (methods, instruments and prepared resources of the programs) and organization of their co-operation intended for environments. These lines are created by the analysts of domains, skilled programmers, that to beginning of production, on the stage of Life Cycle, a set the technological route (algorithm) for implementation of collection of the programs from resources. This route includes subprocesses and operations of choice of CRU (reuses) and testing, configuration, evaluation of quality and others like that [50, 52, 58, 59].

The sound analysis of dynamics of development of factories of the programs is given in the world [57-59], in particular, after such important attributes of their environments, as a platform, programming, copula languages, facilities, formal types of data, types of transformation and presentation of mediators between the polyglot programs and others like that.

Factory basis are as follows:

1) ready software resources (artifacts, programs, systems, reuses, assets, CRU), etc;
2) Interfaces - specifies of ready resources in the IDL language, API, SIDL, WSDL;
3) TL and PL (Product Lines) Production of PP products;
4) The assembly line;
5) Methods of work programs on the lines of the factory;
6) The system-wide environment PP manufacturing factory.

To these requirements of infrastructure of the European project of Grid with an operating environment (cube) and configuration assembling (ETICS) was realized for the decision of scientific tasks of the global measuring etc.
The general structure of the software system for its production at the factory was set by a graph structure, in the tops of which separate modules were located in different programming languages. In some cases, this structure has been described in high-level meta languages (VDL, CASL(Common Algebraic Specification Language), Lotos, Z, B and so on). Programming methods (modular, Assembly, synthesis, generation, etc.) were based on the specification modules in PL and demanded verification of software from ready elements.

7. THE FUTUTE TECHNOLOGIES INTERNET AND NANONECHNOLOGY

The Future Internet technologies [24-26, 32, 59, 62]:

1. The information objects (IO) that specifies the digital projection of real or abstract objects that use Semantic Web Ontology interoperability interfaces. IO through Web services began more than 10 years ago. Interaction semantics IO is based on RDF and OWL language of ISO 15926 Internet 3.0. The next step of the development of the Internet is Web 4.0, which allows network participants to communicate, using intelligent agents.
2. A new stage in the development of enterprise solutions-cloud (PaaS, SaaS) who spliced with Internet space and used to create Adaptive applications. Cloud services interact through the Web page by using agents.
3. Internet stuff (Internet of Things, Smart IoT) indicates the Smart support competing APPS using distributed micro services such as Hyper cat (mobile communications); industrial Internet (Industrial), covering the new automation concepts-smart energy, transportation, appliances, industry», and another.

Concept of nanotechnologies

The idea of an Assembly of atoms in macro atoms special programs assemblies proposed by R. Feynman (1959) in the form of a manipulator of an atom, which are gravity, and the action of intermolecular Vander- Valesov force [32]. Can be an arbitrary number of such mechanisms (machines) submitted by the manipulator of the elements, reduced to four and more times the copies of the "hands" of the operator, which can tighten small bolts and nuts, drill a very small hole, to perform the work in scale 1:4, 1:8, 1:16.
Nanotechnology is the technology of production of new materials and devices with predetermined atomic architecture (E. Dressler).
An atom is 10^{-10} =1 nanometer (nm), and the bacteria is 10^{-9} nm. Particles from 1 to 100 nanometers are called nanoparticles. Some nanoparticles have the property of sticking together with each other, which leads to the formation of new agglomerates (in medicine, ceramics, metallurgy, etc.).
One of the greatest challenges facing nanotechnology is how to make the molecules group in a certain way and to organize themselves so to finally get a new material, substance or device. For example, proteins that can synthesize from few proteins of DNK in complex structures with specific new properties.

Computer nanotechnology

Today computer nanotechnology is actually already working with the smallest elements, "atoms" similar to the thickness of the thread (transistors, chips, crystals, etc.). For example, a video card from 3.5 million particles on single crystal, multi-touch maps for retinal embedded in the eyeglasses, etc [51].

Computational geometry is a part of computer graphics and algebra. Used in the practice of computing and control machines, numerical control etc. is also used in robotics (motion planning and pattern recognition tasks), geographic information systems (geometric search, route planning), design chips, etc.
In the future, ready-made software elements will be developed in the direction of nanotechnology by "reducing" to look even smaller particles with predetermined functionality. Automation of communication, synthesis of such particles will give a new small element, which will be used like a chip in a small device for use in medicine, genetics, physics, etc.

8. CONCLUSION

The author examines the formal bases of the theory and technology of programming in the initial period of computer development in the USSR and in subsequent years. An overview of the basic theory of Soviet and abroad specialists the essence of the theory of programs, methods development and proof of programs as well as software engineering modules in PL and engineering techniques create a *PP* in Software Engineering and SEMAT are discussed. Mathematical design of systems from ready-made resources (objects, components, services, etc.) in the OCM and the models of variability, interaction and configuration of systems from these resources are defined. The formal device of transformation of elements of OM to a component model, given model variability and interaction, and implemented the configuration Assembly of the elements are given. Methods of production of factories (Product Line/Product Family) programs and Appfab and certificate them of quality are discussed. Within the Internet (Things, Smart IoT) technologies are developing in the direction of creating smart computers, cities, robots and devices for use in medicine, genetics, physics, etc.

9. REFERENCE

[1] E. M.Lavrischeva. Development of the theory of programs and systems in the USSR. History and modern Theory of programs and systems. - A collection of SORUCOM-2017. - Development of VT in Russia countries of the former USSR. History and prospects. - 3 to 5 October 2017.- p. 162-176.(in rus).
[2] A. P. Ershov, “Scientific basis of evidence-based Programming”– The report of USSR, 1985.–p.1–14.
[3] A. P. Ershov, “The methodology and technology of Programming”, Conf. Programming technology", 1986, №3.-pp.12-18.
[4] E.M. Lavrischeva, “ Programming Methods. Theory, Engineering, Practice”, Nauk. Dumka, 2006. – 451p.
[5] Parnas David Lorge. Really Rethinking "Formal Methods". Computer (IEEE Computer Society), V. 43, No 1, January 2010.
[6] E. M.Lavrischeva, V. N. Grishchenko, “The connection of multi-language modules in the OS of the ES”, M.: Finance and statistics, 1982.- 136c. (in rus).
[7] E.M. Lavrischeva, “Classification of Software Engineering disciplines”, Cybernetics and Systems Analysis, Vol. 44, No. 6, 2008.
[8] D. Biorner, C. B.Jones, “The Vienna Development Methods (VDM)”: The Meta– Language.– Vol. 61 of Lecture Notes in Computer Science .– Springer Verlag, Heiderberg, Germany, 1978.–215p.
[9] D. Gries, “The Science of programming”, M.: MIR. –1984.
[10] R. Burstall, I.Goguen.”The semantic of Clear, a -specification language”, Lect.Notes. Comp.Sci.-1980, V. 86.-40 p.
[11] R. W. Floyd, “Assigning meanings to programs", Proc. Symp. Appl. Math., 19; in: J.T.Schwartz (ed.), Mathematical Aspects of Computer Science, p. 19-32, American Mathematical Society, Providence, R. I., 1967.
[12] Hoar K. “Structural organization of data, structured programming”.–M.: MIR, 1975.– pp.92 –197.
[13] G. Booch “Object-oriented analysis”, M.: Binom, 1998.-560 p.
[14] I. Jacobson, “ Object-Oriented Software Engineering. -A use Case Driven Approach”, Revised Printing.– New York: Addison-Wesley Publ. Co, 1994.– 529 p.
[15] J. Rumbaugh, A. Jacobson, G. Booch, “UML: special reference book”, SPb. Peter, 2002.– 656c.
[16] P. Clements, L.Northrop, “Software Product Lines: Practices and Patterns”. SEI Series Software Engineering, Addison-Wesley, 2001. ISBN-13: 978- 0201703320.
[17] K. Pohl, G.Böckle, F. J. Van der Linden, “Software Product Line Engineering: Foundations, Principles and Techniques”. Springer-Verlag, 2005.
[18] F. Bachmann, P. Clements, “Variability in Software Product Lines”, CMU.SEI Technical Report CMU. SEI- 2005.-TR-012, 2005.
[19] Lavrischeva E.M, Grishchenko V,N. “Assembly programming” , K.: Nauk.dumka.- 1991.- 213p.
[20] E. M. Lavrischeva, “Theory of object-component modeling changeable software systems”, www.ispras.ru/preprints/docs/prep_29_2016. 2016, ISBN 978-5-91474-026-9.
[21] L.P. Gorognia Programming paradigms: analyses and comparison.- SO RAN, 2017.-239p.
V. N. Grishchenko, L.I. Kucachenko, “ Automatization Information system to support international activities NANY, State intellectual property Committee, certificate, №32304 23.12.2009.
[22] E. M. Lavrischeva, V. N. Grishchenko, “Assembly programming, Fundament of industry programs products”, 2009. -Nauk.Dumra.-371p.
[23] E. M. Lavrischeva, “Component theory and collection technology for development of industry application from ready resources”, The 4-APSSE Conference, 20-21may 2015, pp. 101-119.
[24] E. M. Lavrischeva, A.K. Petrenko, “Modeling of families of software systems”.- The proceedings of ISP RAS, Volume 28, issue 6. – pp. 49-65.
[25] Lavrischeva E.M., Ruznov A.G. Application the theory general data Types standards of ISO/IEC

12207 GDT to Big Data", Actual problems in science and ways their development, 27 december 2016, http://euroasia-science.ru
[26] E.M. Lavrischeva, V.S. Mutilin, A.G. Ryzhov." Designing variability models for software, operating systems and their families", 2017, Proceeding of ISP RAS, 2017, volume 28, issue 5, 2017, pp. c.162-176. Doi: 10.15514/ISPRAS-2017(5).
[27] V. M. Glushkov, "Automata theory and formal transformations of microprograms", Cybernetics. – 1965, № 5.– pp. 1-10.
[28] Y. V. Kapitonova, A. A. Letichevsky , "Methods and means of algebraic programming", Cybernetics. – 1993, № 3. – pp.7-12.
[29] V. M. Glushkov, G. E. Zeitlin, E. L. Yushchenko "Algebra. Languages. Programming", Kiev.- Nauk. Dumka, 1974.- 287p.
[30] V. M. Glushkov, V. G. Bondarchuk, T. A. Grinchenko, "ANALYST", etc.-74, 79, 89, 93, 2000. – Cybernenica and system analysis. -1995, No. 5. –pp. 127–157.
[31] E. L. Yushchenko, "Address language".- K.: Cybernetic in transport", 1962..-52p.
[32] E.M.Lavrischeva, I.B.Petrov "Ways of Development of Computer Technologies to Perspective Nano Future. - Proceedings of Technologies Conference (FTC-2017), 29-30 November 2017| Vancouver, Canada, p.339-348.
[33] Tuygy E.Kh. "Conceptual Programming".-Moscow.- Science, 1984.-256p.
[34] S.S. Lavrov. " Lection on Programming Theory".- Test book.-Sankt-Peretburg, Min edu.PF, 1999.-107p.
[35] V. M. Glushkov, "Bases of paperless computer Science", M.: Nauka, 1982, - 281 p.
[36] V. M. Glushkov, "Cybernetics, VT, Informatics" - (ASM).–Elected. works in 3 volumes, K.: Nauk. Dumka, 1990.-262p.- 267 p.- 281 p.
[37] A. P. Ershov, "Introduction to theoretical programming", M., 1977. -321p.
[38] V. E. Kotov, "Introduction to the theory of schemes of Programs", Novosibirsk, 1978.
[39] V.A.Evstigneiv. "Application of graph theory in programming, M.: Science. Main editorial Board for physical and mathematical literature, 1985.- 352 p.
[40] E.M. Lavrischeva, N.S.Nikitchenko, L.H.Omelchuk. "Technology development of information System", K.: VSKN.-2017.-386 p.(in ukr).
[41] V. N. Koval, "Conceptual languages. Prover of development".- K.: Nauk..Dumka, 2001.-182 p.
[42] V. N. Koval, Z. L. Rabinovich, "Logical-algebraic approach to verification of discrete systems". - IV Method. Conference "Technology SW". -1995. (in rus).
[43] I.V. Velbitskiy, "Technology programming", K.:Technik, 1984.-270 p. (in rus).
[44] N.M. Zadorozhnaya and E. M. Lavrisheva, "The Management of document turnover of IS education", Kiev, 2007, Ped. Dumka.-225 p. (in ukr).
[45] V.V.Lipaev, B. A. Posin, A. A. Shtrik, "The Technlogy of Assembly programming", M., 1992.- 324 p.
[46] Grishchenko V.N. "Theoretical and Applicational Aspects of component programming", Autoref. of doctor Disser.- IK NANU, 2007.- 34 p.
[47] Lavrischeva E.M. "Software Engineering and technology development of complex systems".- Urait.- 2016.- 486p (in rus).
[48] E. M.Lavrischeva, "Software engineering. The theory of programming" and "Technology Programming".- 2016.- MIPT.-48p., and 51 p. (in rus).
[49] Ekaterina Lavrischeva, Andrey Stenyashin, Andrii Kolesnyk. "Object-Component Development of Application and Systems. Theory and Practice". Journal of Software Engineering and Applications, 2014, http://www.scirp.org/journal/jsea.
[50] Lavrischeva Ekaterina. "Ontological Approach to the Formal Specification of the Standard Life Cycle", "Science and Information Conference-2015", Jule 28-30, London, UK,www.conference.thesai.org.- p.965-972.
[51] E. M.Lavrischeva, "Software engineering computer systems. Paradigms, technology, CASE-tools Programming".-K.: Nauk. Dumka, 2014.- 287p. (in rus).
[52] A. I. Ostrovsky, "An approach to the interoperability of software environments JAVA and MS.Net", The problems of programming, 2011.-№2.- pp.37-44 (in rus)..
[53] J. D. McGregor, P. Sodhani et al. "Testing Variability in a Software Product Line", Software Product Line Testing Workshop (SPLiT). Boston, MA, Avaya labs. –2004. – pp. 45– 50.
[54] Andon F.I., Koval G.I. and others. "Foundation of Engineering quality of the Software systems".- K.: Akademperiodica, 2007.- 680 p. (in rus).
[55] Koval G.I. Model and Methods engineering guality program systems on early Life Cycle.- Autoref. Diser.-2006.- 22p. (in ukr).
[56] Koval G.I. , Lavrischeva E.M., Korotun T.M.Approach to modeling of software Quality of family systems.-Probl. Programming.-2009.-N4.- p.49-58 (in ukr).
[57] Lavrischeva K.M. "Theoty and Practice of Software Factories".- Cybernetic and Systems Analyses, 2011.- Vol.47.-N6.-p.961-972.

[58] Lavrischeva K. A. Aronov, A. Dzubenko." Programs Factory – A conception of Knowledge Representation of Scientific Artifacts From Standpoint of Software Engineering" // Comp. and Inf. Sci., Canadian Center of Sci. and Edu. – 2016, 2013. – P. 21–28.

[59] E.M. Lavrischeva." Scientific basis of Software Engineering.- Theoretical and Applied Aspects of Program Systems Development» (TAAPSD'2017), 4-8 desem. 2017. - c. 201-214.

[60] I. B. Burdonov, A. S. Kosachev, V. V. Kulyamin. The theory of the matching blocking and destruction. – M.: Fizmatlit, 2008.-412 p.

[61] I. B. Burdonov, A. S. Kosachev. "A General approach to solving problems on graphs by a group of Automata". - Proceeding ISP PAS, ISSN 2079 – 8156 (print), volume 29, ussue 2.- p.27-77

[62] E. M. Lavrischeva, L. E. Karpov, A. N. Tomilin. "Approaches to the representation of scientific knowledge in the Internet Science". Sat. XIX All-Russian scientific conference "Scientific service on the Internet", Novorossiysk, September 18-23, 2017.-pp. 310-326.

[63] Lipaev V. V. Software engineering of complex custom software products. Textbook.- M.: Ed. Max-press. 2014.-399p.