SOFTWARE ENGINEERING: NEW DISCIPLINES AND E-LEARNING THEME FOR DEVELOPMENT OF APPLIED SYSTEMS

Prof. Dr. Ekaterina Lavrischeva
Moscow Institute of Physics and Technology (MIPT)
Institute for System Programming RAN

ABSTRACT

This article presents the new scientific disciplines of software engineering (SE), which oriented on constructing different applications, applied systems (AS) and product families system (SPF) from ready reusable components (RC) in conditions of factories. These disciplines: are such as: theories of reuses, them engineering (technology), economy, management and productions of programs products from RC. The new formal basis of programming by components, models of variability and interoperability systems of components for executions in the modern environment and practices building AS and SPF of RC for these disciplines are realized into the instrumental and technological complex (ITC) as the programs factory based on systems tools and instruments of MS.Net. This factory ITS may be used for electronic learning of different aspects of proposed disciplines for the building AS and SPF, which describes in our textbooks, which may be use for building simple applications and AS.

Keywords: disciplines, technologies, variability, industry, applied systems, educations, e-learning.

INTRODUCTION

The course «Software engineering» is ratified the cabinet of minister of Ukraine in December 2006. The program of a course was created on the basis of Curricula-2001 and 2004, SWEBOK-2001 and developments of scientific department «Software engineering» (SE) (1980г.) at Software Iinstitute of the NANU. Software Engineering was elaborated more than in twenty NANU fundamental projects and the several applied project. For this time after direction SE is protected near the 12 candidate's and a few doctoral dissertations. The author delivers lectures (with 1970) in KNU Taras Shevchenko on technologies programming (programming languages, compilers, systems of automation of programming, application systems, OOP, UML, Reuses of components and etc). For preparation of students by author three textbooks SE (2001) are created and in websites 2007 – www.intuit.ru, 2011 - www.programsfactory.univ.kiev.ua and www.sestudy.edu-ua.com are developed. Textbooks are published in Department of Education of Ukraine and Russia.

There are the main direction researches was programming of technology, SE for industry of AS and SPF from RC. The training SE for course KNU and of the Kiev Branch Moscow physic - technical institute (MFTI, 2001) are connected with current trends of development of the system software in operational environments (VSTS, Eclipse, Cobra, Java) and methods of industrial manufacturing of software products (technological and Product Lines, methods of conveyor assembling on these lines by reuses of paradigms programming (module, object, component, service, aspect and etc.), variability, interaction, quality, etc.). As results, it was

created the new classification SE of disciplines producing AS, SPF from RC on websites factories, which was developed by students was developed.

In this article the author point of view on separate theoretical and technological aspects scientific disciplines of theories, engineering (technology), economy, management and productions of programs products from RC are described [1, 2, 3]. The approaches to teaching students of these disciplines for training building program of SE from RC in KNU are stated.

SOFTWARE ENGINEERIN: NEW DISCIPLINES FOR INDUSTRY OF PRODUCTS

The main principles of SE are productivity, industry and quality. SE accorded to the corresponding body of knowledge—SWEBOK (http://www.swebok.org), which was developed in 1999, 2004 by the international committee, formed by ACM and IEEE. SE is connected with such general disciplines

Computer Science, electronic engineering, mathematic, telecommunication, cognitive sciences and so on. SE has relations with these sciences and integrates the principles and ground of fundamental sciences such as: the theory of algorithms, the mathematical logic, a set theory; the management theory; a proof theory (Dijkstra, Hoare, and Wirth and so on), the theory of classification mathematical, informational object and programs, etc.

Ten areas of SWEBOK knowledge have the two directions:

1. Development: requirement engineering, designing architecture, developing programs, testing and maintaining software.
2. Management: managing project, configuration, quality, methods and means of SE.

These areas knowledge are corresponded on the lifecycle processes of ISO/IEC 12207 - 1996, 2007 standard.

Definition 1*(SE by SWEBOK)*: Software engineering is a system of methods, techniques and disciplines of planning, development, maintenance and exploitation of software that can be mass-produced. This definition covers all the aspects of the creation of software beginning with the formulation of requirements, development of a product and its maintenance and ending with its removal from exploitation.

Body of SWEBOK oriented on software objects, provides means for the software development of multipurpose applied, systems of Computing Science and informational systems (fig.1) [3]. It does not introduce target objects (AS, SPF, domains, etc.) and technologies for their developing.

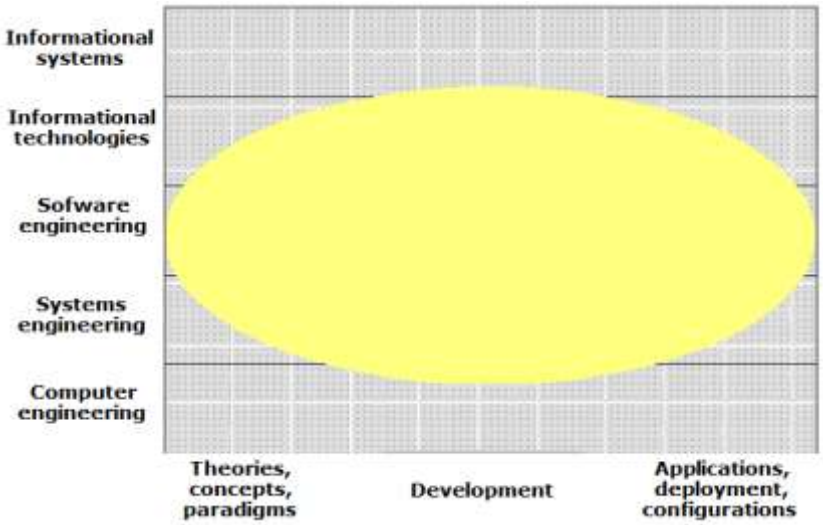


Fig. 1. Place of Software Engineering in Informatics

According to the approach towards teaching computer sciences in the United States, software engineering plays a prominent role in the informational disciplines (fig. 1). The area of SE comprises the entire hardware level, as well as technological and organizational levels of informational sciences. As a discipline, SE covers systematic software development of application, product system families, domains, and other scalable software projects. The main goal of software engineering is developing systematic models, reusable readymade components, and reliable methods for producing high-quality software products.

This goal encompasses theories, concepts, and paradigms, including both development structures for reliable computational software systems and management aspects of the designing process to meet customers' needs.

Definition 2: SE is a system of methods and means of theory programming, engineering, management of planning and team of manufacturing target software systems (AS, SPF, domains, and software project), economic methods of measurement and estimation of the computability of their various characteristics as to their conformity with customer's requirement. Production oriented toward the use of base objects (reusable component, reuses, assets, artifacts, services, and est.) and automated operation similar to those used in conveyor production [4-8].

It was proposed by author a new classification of SE disciplines, which is oriented on different aspects industrial production of AS and SPF [1- 6], such as:

1. *Scientific discipline* consists of the classic sciences (theory of algorithms, set theory, logic theory, proofs, and so on), lifecycle models, fundamentals data types, theory of integration and interface, theory of programming and the corresponding language tools for creating abstract models and architectures of the specified objects, etc.
2. *Engineering discipline* is a set of technical means and methods for software development by using standard lifecycle models; software analysis methods; requirement, application and domain engineering with the help of product lines; software support, modification and adaptation to other platforms and environments.
3. *Management discipline* contains the generic management theory, adapted to team-based software development, including job schedules and their supervising, risk management, software versioning and support.

4. *Economy discipline* is a collection of the expert, qualitative and quantitative evaluation techniques of the interim artifacts and the final result of product lines, and the economic methods of calculating duration, size, efforts, and cost of software development.

5. *Production discipline* consists of product lines, utilizing software resources (reusable components, services, aspects, agents, and so on), taken from libraries and software repositories; it also contains assembling, configuring and assessing quality of software.

The disciplines of SE recommended for students training on courses for disciplines: management, economies and production. This disciplines are learning on baccalaurean, magisterial and aspirant courses and developing product lines for building AS and SPF from RC and estimating quality of such products [9, 10].

SCIENTIFICALLY DISCIPLINES

Theory of programming is a set of methods, languages, means of description (specification) and design of target object and methods of their proof, verification, and resting. Methods of programming in SE include:

Theoretical (algebraic, algorithmic, logical etc.) and applied (object, component, agents aspects, etc.) intended for designing different types of target objects; Methods of program verification with the help of formal (assertion, inherence, and proof) procedures of OOP, UML, VDM and so on; Methods of estimation of the results design from analysis environment to programming and estimation characteristic quality (reliability, accuracy, functionality, etc.) of finished products.

Besides these theories, we proposed as new scientific results of the long-term research and development as follows:

1. Concept formal definition of RC, AS and SPF from reuses which developed on paradigms programming (module, object, component, service, aspect and etc.) [1-5];
2. Methods of paradigms programming for the formal design AS from different reuses and evolution them by reengineering reverse engineering and refactoring [6, 7];
3. A new models variability and methods for management process AS into SPF [11];
4. Interoperability programs, systems for modern environment [12, 13];
5. E-learning to disciplines of Software Engineering [12].
6. Evolution conceptions of generative programming by K. Czarneski and Y. Aisenecker and our results into final fundamental projects of Software Institute of NANU (2007-2011) [13].
7. Programs Factories – basic foundation of industry programs products [14-17].

Reusable Components (RC), AS and SPF

The *model RC* for component-based development has the following specification [3]:
RC = {T. I, F, Imp, and S},

where T – type, *I* – interface, *F* – functionality, *Im* – implementation, *S* – interoperability service.

Basic operations of components are: Specifying components and their interfaces (pre– and post–conditions, which must be satisfied by caller them) in such languages as IDL, API, WSDL etc.

Maintenance of components and reuses into the component repository for their future integration in AS. Collection, assembling RC into Applications, Domains, AS, SPF. All aspects development RC and using them in AS and APF – the goals reusability disciplines in SE.

Applied systems (AS) is a collection of software means (or function), including general tools (DBMS, protection systems, systems services, etc) constructed subsystems or components of AS with marshaling parameters from one AS to second.

Family of the program systems (SPF), understood by us, as an aggregate of the program systems and AS, incorporated (constrained) between it the common incurrence of concepts of domain, their general characteristics on set RC. The method of its manufacturing is realized by example, component paradigm [9].

Component theory of building AS from RC and interfaces

The theory of component programming: It has been developed for modeling Subject area, domains, AS with RC from set components C . Its main postulates are as follows [8, 17-19].

Axiom 1. Subject area, which is modeled with a set of components C, is a component itself.

Axiom 2. Subject area that is modeled may be a separate component C in the another subject area.

Based on the theory of sets, if $C = (C_0, C_1, ..., C_n)$ - the set of components on and C_0 corresponds to the subject area, then following equation holds for elements C:

$$\forall\ i\ [(i > 0) \rightarrow (C_i \in C_0)]. \quad (1)$$

Each component is represented as an element of a set. In this case, the expression (1) is transformed into

$$\forall\ i\ \exists j\ (i > 0) \wedge (j \geq 0) \wedge (i \neq j) \wedge (C_i \in C_j) \quad (2)$$

For each component of a set C (except C_0), algebraic operations from the set theory may be applied, such as "part-whole" relation, aggregation, and so on.

Axiom 3. If $C = (C_1, C_2, ..., C_n)$ a set on components, and $P' = (P_1, P_2, ..., P_r)$ is a set of unary predicates associated with the properties (features) components of elements of C, then the component C_i corresponds to a set of statements and predicates of P', and for a certain set of pairs (i, k), $P_k (C_i)$ = true.

Building component of AS, families of the elements from the elements of these sets is performed by collecting RC according to interface data, which corresponds to the defined properties and characteristics of the components of the set C, which accumulate in the repositories (interfaces and implementations) in the environment of component and the repository.

Definition 3: Component is understood by us as a PL-independent self-relevant of software product that provides execution for a certain set of application functions and services of applied domain, which can be accessed with remote calls.

Component model is a consequence of the generic typical solutions or functions for AS, which are presented by the components and its interface with the properties of in the following form:

$C = (CNa, CIn, CFa, CIm, CSe)$, (3)

where

CNa - the name of the component;

$CIn = \{CIn^{in} \cup CIn^{Out}\}$ is a set of input and output interfaces;

CFa is a set of methods and instances of components;

CIm is interface of implementation component;

CSe is system service for components of *C*.

Input interfaces CIn^{in} correspond to the components implementation, while output interfaces CIn^{out} match another component from the set *C*.

Input interface $CIn^i \in CIn$ has a model $CIn^i = (InNa^i, InFu^i, InSp^i)$, (4).

Interface implementation $CIm^j \in CIm$ has the following model:

$CIm^j = (ImNa^j, ImFu^j, ImSp^j)$, (5)

where suffix *Na* denotes the name of an interface or an implementation, *Fu* denotes a list of function, *Sp* is its specification.

The necessary condition for the component $Cj \in C$ to exist is its integrity:

$\forall \; CIn^i \in CIn \; \exists \; CIm^j \in CIm \in Pr\,(CIn^i) \subseteq CIm^i$, (6)

where $Pr\,(CIn^i)$ devotes functionality that provides implementation of the interface methods from *CIn.*

Axiom 4. To combine two dissimilar components C_1 and C_2, the following conditions must be satisfied: if $CIn^I_1 \in CIn_i^{out}$, then there must exist such that $CIn^i_2 \in CIn_2^{in}$, then satisfies there must be

$Sign\,(CIn^i_1) = \text{Sign}\,(CIn^i_2) \;\&\; Pr\,(CIn^i_1) \subseteq CIn^I_2$, (7)

where Sign (...) is the signature of the corresponding interface and *Pr* (provide) is functionality to implement interface methods CIn^i.

The model of the *component environment* has the following form:

$CE = (CNa, InRep, ImRep, CSe, CSeIm)$, (8)

where $CNa = \{CNa^m\}$ is a set name components that make up the environment,

$InRep = \{InRep^i\}$ is the repository interfaces components of the environment,

$ImRep = \{ImRep^j\}$ is the repository implementations,

$CSe = \{CSe^r\}$ is the system services interface,

$CSeIm = \{CSeImr\}$ is a set of implementations of system services.

Each element of *InRep* described by the pair (CIn^i, CNa^m),

where CIn^i component interface (4), *CNam* – the name of the implementing component. By analogy, each element in *ImRep* is described by (CIm^j, CNa^m), where CIm^j is the implementation interface (5).

These theoretical axioms, as well as the theory for transforming data types in heterogeneous components are implemented in the component environment website of ITC [20].

Transforming data types of RC

Component model emerges from generalized solutions to the nature of objects. The methods of objects are converted to the component architecture, properties, and characteristics. There are two approaches to solving the problem of integration of RC:

1. Use of the interaction model through the intermediary module (stub, skeleton) using cross-language and intermediate interface [3, 9];
2. Description of the reusable component interface in the IDL language with **in**, **out**, **inout** parameters to specify the data values.

In the theoretical aspect, multilingual component integration is based on formal mappings (presented as a superposition of base mappings).

Multilingual RC assembling model: Let $CSet = \{C_i\}$ a set of components written in multiple programming languages. During their interaction, components C_i are exchanging data. Each pair of components C_i and C_j may be equivalent provided they have the same semantic structure and type, or non-equivalent otherwise. In the latter case their conversion is required using functions represented by mappings:

$$FN_{ij}: N_i \rightarrow N_j, \quad FT_{ij}: T_i \rightarrow T_j, \quad FV_{ij}: V_i \rightarrow V_j, \tag{9}$$

where FN_{ij} establish correspondence between the names of variables,
FT_{ij} was describing the equivalent mapping of data types,
FV_{ij} implementing the necessary conversion of data values.

Problem of variables replacement FN_{ij} is solved by ordering variable names (for example, in the configuration description for the components in question). Mapping between data types FT_{ij} is based on the transformation of data types, each of which is represented by an abstract algebra and algebraic system $T = (X, \Omega)$, where X is a set of values that can make a change of this nature, and Ω is a set of operations over these variables. Reflection FV_{ij} is applied in case types T_i and T_j are not equivalent (e.g., the conversion of an integer value to real) [12].

Conversion from the type $T_i = (X_i, \Omega_i)$ to the type $T_j = (X_j, \Omega_j)$ is meant as a conversion, in which the semantic content of operations from Ω_i is equivalent to content of operations from Ω_j. These conversions are available for common data types in most programming languages as described by the ISO/IEC General Data Types (GDT) standard 11404–2007; it provides mechanisms for generating fundamental data types from general data types [2].

The problem of component interconnection occurs when assembling them into compound structures. To solve this problem, the set of reflections is built for different types of method calls, in order to establish a correspondence between the set of actual parameters $V = \{v_1, v_2, \ldots, v_k\}$ of the object and set of formal parameters $F = \{f_1, f_2, \ldots, f_l\}$ for components.

The problem of constructing data types using algebraic systems for basic data types used in various program languages and isomorphic mapping between algebraic systems are considered in great detail in [2,3].

Implementation of interfaces: Interface is based on transformation of irrelevant type of objects in different PLs, passed through mechanisms of formal and actual parameters. The data, related to general data types and fundamental data types can correspond to the request parameters of other objects. They may be incompatible between themselves because of the differing platforms, number of parameters sent, and varying compiler implementations of data types; therefore, they require the proper transformation [9].

Different programming languages do not have a common implementation of fundamental data types and general data types, which are described in ISO/IEC 11404 standard.

Fundamental types are:
Simple data types (real, integer, char, etc.);
Structural data types (array, record, vector, etc.);
Complex data types (set, table, sequence, etc.).
They are used by all programming languages and are implemented by translators.
General data types are:
Primitive data types (character, integer, real, complex number, etc.);
Aggregate data types (enumeration, pointer, set, bag, sequence, etc.);
Generated data types (which emerge as a product of data type generator from one or several data types);

Reusable components (reuse, artifact, object, component, service, etc.) are described in a programming language using one of the standard WSDL, Grid or IDL interfaces. They are saved in the RC repository and interface repository.

For data type transformation from one PL into other, three libraries are developed:
GDT library;
Library for reflection functions GDT↔FDT;
Library for functions for data transformation $PL_i \leftrightarrow PL_j$;

Intermediate libraries for functions and routines for transforming standard data types within a certain environment (*CTS*, *CRL*, *FCL* .Net) [9].

Algebra modification of Components and RCs

Methods of transformation RC are divided into two types: methods that change the functionality and behavior of the components and methods that are associated with non-functional changes. The first type includes changes in the interface (changes in interface signatures, adding a new interface) and the implementation (changing algorithms and logic, replacing and adding realizations) and others. The second type includes changes related to non-functional characteristics of the application (reliability, efficiency and mobility), languages and execution platforms [9].

The system is assembled from RCs by modifying or adding their interfaces and/or implementations [2-5].

The general component algebra includes external, internal and evolutional algebras:

$$\sum = \{\varphi_1, \varphi_2, \varphi_3\} = \{CSet, CESet, \Omega_1\} \cap \{CSet, CESet, \Omega_2\} \cap \{CSet, CESet, \Omega_3\}, \quad (10)$$

where φ_1 is the external component algebra, φ_2 is the internal component algebra, φ_3 is the evolution component algebra.

External component algebra $\varphi_1 = \{CSet, CESet, \Omega_1\}$, where *CSet* is a set of components *C*, *CESet* is the environment *E* with set of components *C* and interfaces *Int.*

$\Omega_1 = \{CE_1, CE_2, CE_3, CE_4\}$ represents algebra operations:
CE_1 – operations of component processing;
CE_2 – initialization operations;
$CE_3 = CE_1 \cap CE_2$ – assembling operations;

$CE_4 = CE_1 \setminus C$ – operation for extracting component C from its environment;
$C_2 - CE_2 = C_2 \oplus (CE_1 \setminus C_1)$ – substitution operation.

Internal component algebra: $\varphi_2 = \{CSet, CESet, \Omega_2\}$,
where $CSet = \{OldComp, NewComp\}$ is a set of old components *OldComp* and a set of the new components *NewComp;*.

the set $OldComp = (OldCName, OldInt, CFact, OldImp, CServ)$ includes interfaces, implementations in the server environment;

the set $NewComp = (NewCName, NewInt, CFact, NewImp, CServ)$ includes interfaces, implementations for these components;

$\Omega_2 = \{addImp, addInt, replInt, replImp\}$,
where *addImp* denotes the operation of adding an implementation; *addInt* is the operation for adding an interface; *replImp* is the operation of the substitution of a component implementation, *replInt* is the substitution of a component interface.

Component Evolution Algebra. Reuse is the basis of the components evolution. Component evolution algebra is $\varphi_3 = \{CSet, CESet, \Omega_3\}$, where $\Omega_3 = \{O_{refac}, O_{Reing}, O_{Rever}\}$ is a set of component evolution operations; O_{refac} is the refactoring operation, O_{Reing} is the reengineering operation, O_{Rever} is the reverse engineering operation for a certain component.

Component refactoring model is as follows:

$M_{Refac} = \{O_{Refac}, \{CSet = \{NewComp^n\}\}$,

where $O_{Refac} = \{AddImp, AddNImp, ReplImp, AddInt\}$ is the refactoring operation, the pair $(CSet, O_{Refac})$ is an element of the evolution component algebra.

Components of evolution algebras are built on base of the semantics terms and requirements on features refactoring implementation $O^{Refac} = (CSet, Ref)$,
where $CSet = \{C_n\}$ is a set of components, and $Refac = \{AddImp, AddInt, RelImp, AddInt\}$ is a set of refactoring operations. *AddImp* adds a new implementation of the existing interface; $NewComp = AddImp(OldC, NewCIm, NewCInO)$ is the operation of adding a new component, $NewCInt = OldCIn \cup NewCInO^s$ is the operation of adding output interfaces for a new implementation, $NewCImp = OldCIm \cup \{NewCImp\}$ is the operation of adding a new implementation.

Theorem: Algebra component refactoring $\sum^{refac} = (CSet, Refac)$ is complete and consistent.
Model for component reengineering is as follows:

$M_{Reing} = \{O_{Reing}, \{CSet = \{NewComp^n\}\}$,

where $O_{Reing} = \{rewrite, restruc, adop, conver\}$ are reengineering operations, the pair $(CSet, O_{Reing})$ is an element of evolution component algebra.

Reengineering algebra components $\sum^{Reing} = (CSet, Reing)$ are used in case of component integrity violation or changes in their functionality.
Axiom 5. For $\exists OldInt \in OldInI$ there exists an adding operation of input interfaces and corresponding functionality.
This operation is associative and commutative and observes integrity of the component.
Operation *AddImp* denotes adding a new implementation of the input interface, which is not included in the set of component interfaces:
$NewC = AdNIm(OldC, NewImp, NewCInt)$.
These operations are associative and commutative. The integrity of the component is retained.

Operation *ReplIm* is the replacement of an existing implementation by a new one without changing the input interface: *NewC =ReplIm(OldC, NewCImp, NewCInt, OldCImp, OldCInt).*

Lemma 1. Operation of replacing the existing implementation with a new one given terms and semantics, mentioned above, preserves the integrity of the component.

Operation *AddInt* is adding a new input interface for an existing implementation.

Lemma 2. Operation of adding a new input interface for an existing implementation given terms and semantics, mentioned above, preserves the integrity of the component.

Reverse engineering model: $M_{\text{Rever}} = \{O_{\text{Rever}}, \{CSet = \{NewComp^{n}\}\}$,

where $O_{\text{Rever}} = \{restruc, conver\}$, the pair $(CSet, O_{\text{Rever}})$ is an element of evolution component algebra $\sum^{\text{Rever}} = (CSet, Rever)$.

In reverse engineering, reengineering operations can be used: $Reeng \subset Reverse$. The feature of the set *Reverse* is that its operations are not completely defined, but rather partially. It is because the reverse engineering means complete alteration of the program system using components.

Overall, the component algebra $\sum^{Reface}$, $\sum^{Reign}$, $\sum^{River}$, as well as models M_{Refac} , M_{Reing} and M_{Rever} constitute the formal apparatus of evolution algebra from models, operations and methods for the evolution of the components.

Concept of AS and SPF Variability

Definition 4: *Variability* – is the ability of software systems or artifacts in software product family to extended, changed, customized or configured for use in a specific context with the proper quality characteristics to mitigate, its current limitations [9, 11].

Variability serves to increase reusability and sensitivity of application to the changes in business processes and environmental conditions, by expanding the context of its applicability. In general, it can be implemented in SPF and in specific application, AS. It's provide variability of a Software Production Line that supports the existence of the SPF as a set of AS, and creates a set of basic RC which are contained in the repository. The variability is aimed at ensuring the variants of SPF and it's AS [11].

At the same time it increases the complexity of integrated development environment (IDE) and, consequently, the duration and cost of AS production cycle. Compliance with the balance between expected benefits and costs of implementation leads to variability assessment and management in IDE.

Variability is based on features, variation points, variants, constraints and dependencies. A feature in this sense is a characteristic of an AS for some users. A feature can be for the example a requirement, a technical function or function group or a non-functional (quality) characteristic. Variation points identify locations in AS or artefacts at which a choice can be made between values, or zero or more variants. Variant is an instance of different configuration items that are similar-but-different; they typically represent alternative functional capabilities. Only one variant from a set of similar-but-different variants is typically present in any given configuration.

A theoretical basics technology development variable AS on the options and their SPF vas building, improves the modeling environment and the algebra of the theory of component

programming [13]. Components of the model environment: an object and component model SPF and model artifacts.

Definition 5: *Object model* has the following form:

$$OM = \langle G^t_1; G^t_2, G^t_3, G^t_4 \rangle,$$

where G^t_1 – count objects to generalizing about the level of its design (t=1);

G^t_2 – feature model characteristic level (t = 2);

G^t_3 – structural component model (t = 3);

G^t_4 – interface interaction model components at the behavioral level (t = 4).

Object functions G^t_1 and their characteristics match the methods and data (level 2 and 3) necessary for their implementation in the AS and coordination.

Definition 6: *Component Model* SPF – OM development, methods of objects which are implemented by RC for one and only one of object and the interface between them. It has the following form:

$$CM = \langle RC, In, ImC, Fim \rangle,$$

where *RC* – basic components of the set *C*;

In – interface components of C;

ImC – features that make the·implementation of basic components;

Fim ( ) set of data characteristics, including variant in the signature interface.

Variability is based on the management processes for SPF variants:

imposing requirements on the SPF by solving scheduling tasks;

adapting to the new operating conditions and changing certain functions and programs;

widening the context of usage for SPF, when processes or components of the object domain are changing;

control of configuration structure SPF for creating files for deployment.

Basics of variability processes are realized on configuration ITC.

Interoperability Programs and Systems Model

Definition 7: Interoperability is the ability of components or systems to interact with each other and exchange common information.

Formally, interoperability model is understood as the representation of relationships parameters between different elements of software or informational systems. This model reflects the relationship system and the designing process for software product development. The relationships may be described with mathematical means, such as abstract algebras, standard OSI, interoperability theory and so on [9].

Migration of connective components and systems into new environments is based on using interfaces and network communication protocols. Practical foundation of system interoperability is based on the OSI standard model and includes:

– Model and mechanisms for describing system interfaces,

– Mechanisms and operations for transmitting (marshalling) data via networks from local and global storages in heterogeneous environments.

System interoperability model M_{inter} covers interconnection between systems, developed in a certain heterogeneous environment, to another one, extending their limits. It has the following generic structure:

$$M_{inter} = \{M_{pro}, M_{sys}, M_{env}\}, \quad (11)$$

where $M_{pro} = \{Com, Int, Pr\}$ is a program model, *Com* – a component, *Int* – an interface, *Pr* – a program;

$M_{sys} = \{AS, Int, Prot\}$ is a software system model for the system AS, *Int* – an interface, *Prot* – the data transmission protocol;

$M_{env} = \{Envir, Int, Prot\}$ is an environment model, in which *Int*, *Prot* contain the set of external interfaces and remote calls that transmit data between programs via networks.

The basic parameters for the interconnection model *Minter* are program, interface and message or protocol. Fig. 2 represents data flow between the modern programming environments (VS.net, CORBA, IBM, and Eclipse Java).
The applied models for system interconnection via Eclipse IDE implement three relationships: VS.NET ↔ Eclipse, CORBA ↔ Java, and Web Sphere ↔ Eclipse.

With our opinions', it is necessary to teach the students not only for the rise of theoretical level of their qualification in SE, but also the new conceptions, ideas and facilities of programming, which had substantial influence on the industrial production AS. *Scientific discipline* is a basic course of teaching in Universities, related to the informatics and computer science. This course, per se, theoretical, he must be supported by some classic courses, some courses of the systematic programming (object-oriented, component, service and etc.) and additional courses from the Curricula-2004 program.

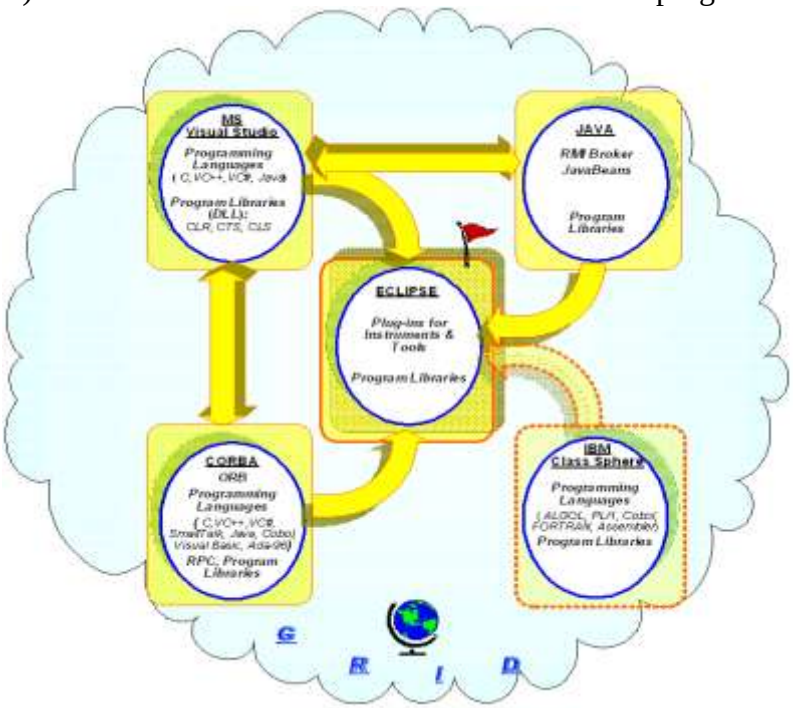


Fig. 2. Interconnection between Modern Environments

ENGINEERING DISCIPLINES

An engineering discipline determines an aggregation of engineering receptions, tools and standards oriented towards making target AS objects using the scientific discipline [1-9]. The basic means of this discipline are as the follows:

– standards (ISO/IEC lifecycle 12207, assessments 15504, 9126 (1-4) quality as a set of regulated rules of constructing artifacts on the life cycle processes, their measurement and assessment quality characteristics;

– Engineering reuses, applications and domains;

– Infrastructure – environment conditions, methodical and organizational providing of base process of SE and support of activity of the SP developers;

– General system facilities and instrumental environments of processes of the SP making.

Engineering Disciplines are consisted the following base tasks:

– developing product lines from repository components and services;

– developing AS from multi-language reuses by means of applications and domains engineering [9];

– Methods and means of supplying correctness (V&V); assessing AS (quality, cost, capacity) and configuring AS from RC [7];

– Designing ontology-based models for lifecycle (OSI/IEC 12207 standard) in DSL and domain of computational geometry and so on;

– measuring the characteristics of quality, cost, capacity RC and AS, SPF;

– New tools of web-services for developed AS.

Life cycle of Standard ISO/IEC 12207

This standard defines the general structure and content of the life cycle, starting with requirements for recycling (cancellation) AS. Structurally, it consists of a description of many processes (43), the relationship between them, and set out actions and tasks to be performed in these processes. All processes in the standard ISO/IEC 12207 divided into three categories:

Basic processes;
Support processes;
Organizational processes.

For each category are listed in the standard works, but given the way they perform and the presentation of results. The *basic processes* include the development, operation and maintenance. The process of development it is design requirements and systems, coding, integration, testing, system testing and installation of the product.

To support processes include: documentation, version control, verification and validation, reviews, audits, evaluation of the product and so on. Process control versions (or variants) are responsible for the content of configuration management systems, which must be checked for correct implementation of the project objectives and compliance with customer requirements. By organizational process include: project management (management development AS), quality, risk, etc. These processes are performed special services that carry out planning activities in the project control processes, defining metrics to measure the product, checking quality and others.

Definition standard life cycle by language DSL: To represent conceptual or ontological model life cycle used language Domain Specific Language Tools and Eclipse – DSL. It creates a visual model domain life cycle (LC) with descriptions of classes and relations between them (fig.4). Each class provided the methods necessary for the visual representation of the domain and its generation in language XML [9].

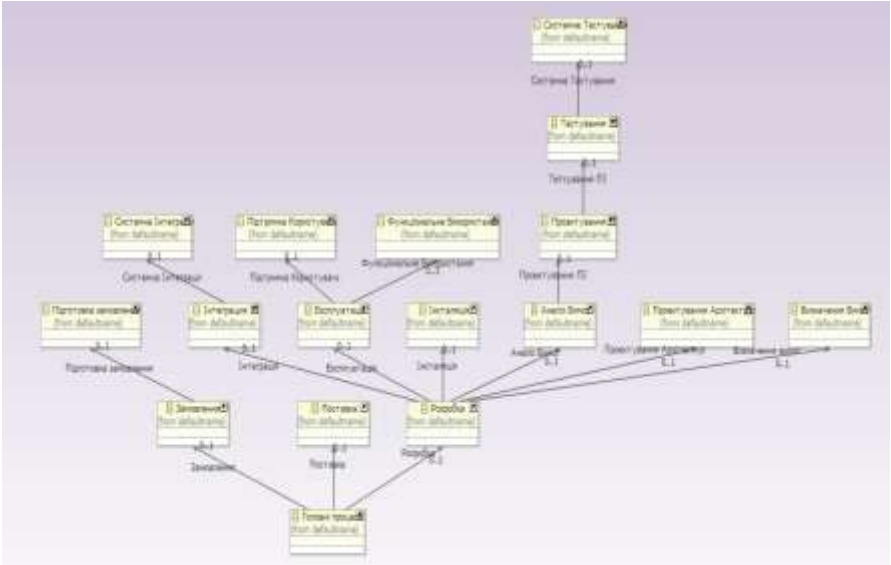


Fig. 4. Model of basic process LC

Designing Technological lines (TL) and Product Lines for fabrics

Technological lines is created at the stage of technological preparation work precedes the production of aircraft and includes the work of formation of the structure or schema line processes and actions that determine the processing elements of the future aircraft means corresponding technological module (TM) or systems programming from PL [14].

The main requirement imposed on the product line development is thorough selection of the lifecycle processes, standard tools of the operational environment, and development of a certain set of normative documents. Bearing in mind the specifics of the chosen applied domain, one can choose appropriate reuses, reusable components, generative and implementation tools, as well as the route for the product line processes.

Product line and product family are defined in the ISO/IEC FDIS 24765:2009(E) – Systems and SE Vocabulary as 'a group of products or services having the common manageable set of properties which meets the requirements of a certain market segment' [5, 9]. Models for representing processes of software development, according to Software Engineering Institute (SEI), are the engineering and the process model.

The engineering model is created at SEI and corresponds to the three-step production: developing reusable components; merging them into software systems; managing components. The development process encompasses defining SPF domain, designing the manufacturing process for a set of components, allowing for its context of use, restrictions and production strategy. The merge process includes designing implementations of each software system on the base of the manufactured resources and components. The management process is oriented towards processes' coordination.

The process model sorts out a set of processes that run on the two levels: domain engineering level, which is also referred to as the development "for reuse", and software systems' engineering level, referred to as the development "with reuse". Assembly lines use readymade components, which improve the time of development and are able to support the whole manufacturing cycle according to the specific requirements and needs.

Thus, TL and Product Lines for experimental manufacturing target object elements are developed the instrumental and technological complex (ITC) [20] by us, as a web site (http://sestudy.edu-ua.net), which oriented on developing variable and interoperable components, AS, and SPF f or evolution into modern environment.

Engineering reuse, applications and domains

Technology of the engineering production SP is based on the repeated use of reusing components prepared facilities, resources and instruments of their construction. Such technologies presently are: Reuse Engineering, Application Engineering, Domain Engineering and Family Engineering without lines [2, 9].

Reuse Engineering was formed as systematic and purposeful activity on the search and selection of prepared repeated use of RC, which take place in the modern repositories or libraries. A base of making is: product framework, set of new realized components and prepared functional reusing. As reusing can be used prepared applied (used in business, commerce, economy and etc.) or general system facilities (translators, editors, ОС, СУБД, integration systems, generations and etc.).

Engineering Applications and domains are also based on reusing components and different program elements. Main task these kinds engineering – this construction of the application systems or SPF, which will realize the tasks of application or domain taking into account the general and changeable features of making their elements (family members). The domain technology close walked up to the modern principles of conveyer production of products from the prepared «details» – reusing on the domain model in DSL (Domain Specific Language) and specifications of every member of family. A basic essence of this technology is a management by the AS making based on plan–graphs of works, control of results of works and evaluation of degree of applicability of prepared resources in the process of realization of specific tasks of domain.

The base components of given engineering discipline must be continuously perfected and be adapted to the new type of target objects and conditions of production environment (that in spirit perfection conceptions, stopped up in the models CMM, SPICE, Trillium and so on).

It is seen, that this discipline can be presented by some textbook, in which description of base components of engineering SE, general principles and technologies of engineering the AS making, modern languages of specification of components, domains, members of families and processes of their production is led.

Means Engineering Disciplines

The site in question was developed as a collection of tools for SE and at the same time was displayed during lectures on SE at Kiev National University [15, 16]. This drove authors to orient the complex towards teaching students and graduate students the basics of SE, including various tools and means of its support, along the following aspects: developing programs and reusable components, services new reusable components in repository, generation components and domains subject-oriented languages DSL; assembling software systems and their families from software resources and RC, as well as developing, generating, interoperability, and ontological modeling techniques for object domains (lifecycle, computing geometry and configuration components in new exchanging structure

AS, SPF). Besides that, electronic monographic on SE and technologies of software development in programming languages Java, C#, C++, Basic [20], and e-learning of course – software engineering and other new disciplines [9].

Taking the above into account, we have chosen a strategy of teaching various aspects of industry-compliant SE. In order to gradually and consistently implement this strategy within the ITC, we utilized the Internet-based methods and modern programming systems that support different aspects of software development, namely:

– Protégé system to model objects domain ontologism.

– Eclipse as a tool to embed different programming and system components into the ITC by using its plug-ins.

– Microsoft Visual Studio as a multifunctional tool to organize team development of the new systems, including developing software via Internet using PL, OOP, UML, and cloud computing frameworks (Azure, Amazon, etc.).

– CORBA system that has a universal broker providing interoperability between programs, written in different PL by using stub/skeleton mechanism.

– New subject-oriented DSL tools for designing domains, AS, SPF and for implementing DSL-based descriptions (Eclipse-DSL, Microsoft DSL Tools, and others).

The start page of the web site features a list of implemented sections and subsections concerning SE. The sections in question are: Main Page, Technologies, Interoperability, Tools, Presentations, and Learning (www.sestudy.edu-ua.net).

Each section contains subsections with keywords that specify the names of product lines (10 altogether). All sections and subsections include standardized pages, such as an overall theoretical description, an example that illustrates the concerned topic (developed with one of the workbench programming environments, in most cases), a thorough description of the example, and so on.

Employees of the SE department and students of KNU and MFTI in course of writing their thematic and graduate papers implemented the web site and several product lines of software development on the basic of RC and components. Particularly, they have developed an experimental program factory [11], software means of variability and interoperability support between programs and systems, a domain description in DSL, Protégé, and so on.

ECONOMIC DISCIPLINE

The economy of SE is an independent discipline of SE and is connected with economic aspects of the AS industry. It is based on economic calculations of different aspects of activity of executors of a project with allowance made for knowledge of all economic factors and current expenditures in the project [6, 7].

This discipline has its theory and practice of solving problems on the examination of a project, cost estimation, assessment quality and terms and economic indices specified in the requirements on an AS during entering into a contract for its development. This discipline provides the estimation of requirements, design solutions, architectures, and development risks connected with available material and human resources, quality indices of an AS, and also financial calculations with each executor at all stages of contracts being performed.

This discipline is most developed from the viewpoint of methods of economic calculations in SE, namely, it includes size prediction methodologies for AS (Function Points Analyses, Feature Points, Mark–II Function Points, 3D Function Points, etc.), estimation of labor expenditures for the development of AS with the help of the family of models COCOMO, several other mathematical models of estimation of labor expenditures for the development of AS (Angel, Slim, Seer-SEM, etc.), and also models connecting economic indices of AS with quality characteristics [6].

In forming this discipline, fundamental economic methods should be used that are connected with principles of distribution and examination of jobs in complex systems, methods of calculation of costs of separate parts of a system depending on the size of its component parts and the cost of the system as a whole, existing standards providing the estimation and certification of finished products, etc.

A systematized and scientifically grounded course of the economic discipline of SE will eliminate the gap that exists in SE and is caused by the absence of the corresponding manuals and textbooks for teaching specialists who will be engaged in solving economic problems in the industrial cycle of AS production.

Module forecasting costs on AS

Module “Softest” was designed to predict estimates of labor and the cost of aircraft by model COCOMO II. This model is a three-level [3, 9]: Application Composition Model provides a preliminary assessment of labor in the early stages of development. Early Design Model provides a preliminary assessment of the labor on Pre aircraft design and cost allocation processes AS. Post Architecture Model refines the estimation of labor Early Design Model when accompanied by airplane. The first two models of COCOMO II use the same equation estimates of nominal labor costs and the difference between the cost and level of detail of estimated elements of AS. Nominal labor costs (average) calculated at the average productive working time, equal to 152 hours a month, according to the formula

$$T_{пот}=AV^{\tau B},$$

where A = 2.45 constant, obtained for example more than 80 real projects

V- the estimated size of the aircraft or aircraft component in thousands of lines of source code KSLOC,

B - Exponent of V, which is defined by the formula:

$$B = 0.91 + 0.01 \sum_{J=1}^{5} Fj,$$

where *Fj* – coefficients attributes scale.

For takeoff field assessments in person-years takeoff field should be divided by 12 and the man-days - multiply by 19.

The nominal duration of the development of AC is calculated by the formula:

$$Д_{ном} = 3.67 \cdot T_{ном}^{(0.38+0.2\cdot(B-1.01))}$$

The principles of this module and its implementation are presented on our web site ITC.

A model quality of RC and AS

It is quality AS characterizes his fitness to the use on setting. For different types AS is set nomenclature of indexes and their base values that must be a set to be attained at development of separate application entities after their models [2, 3, 7].
A model of quality of M_{qv} is the formalized presentation of descriptions (whether indexes) of quality, their properties and methods of estimation on the processes of life cycle. In obedience to the standards of ISO/IEC 9126-2, ISO/IEC 9000 (1-4) and kernels of knowledge of SWEBOK the model of quality got a standard kind for all kinds and types of software components and AS and has such kind:

$$M_{qv} = \{Q, A, M, W\},$$

where $Q = \{q_1, q_2, \ldots, q_i\}$ $i = 1,.., 6$ is a set of quality characteristic (Quality - Q);

$A = \{a_1, a_2, \ldots, a_j\}$ $j = 1,.,$ J is a set of attributes (Attributes- A), each of that fixes separate property of q_i –characteristic of quality;

$M = \{m_1, m_2.,m_k\}$ $k=1,..,k$ is a set of birth-certificates (Metrics - M) for every a_j attribute, used after this measuring;

$W = \{w_1, w_2, \ldots, w_n\}$ $n = 1, \ldots, n$ is a set of weighed coefficients (Weights - W) for leveling of birth-certificates of this set.

In the standard of ISO/IEC 9126-2 six base characteristic of software quality are certain:

q_1: functionality, q_2: reliability, q_3: efficiency, q_4: usability, q_5: maintainability, q_6: portability.

Each of indexes is characteristics and by attributes, that determine her different properties that must be known to the user at a desire to use corresponding RC or AS.

A model of M_{qv} is a four level.

At *first level* there are indexes of quality (Q) that represent properties, that must own created AS. Indexes of the *second level* are attributes of indexes (correctness, faultlessness and other), necessary for AS, to attain the set characteristic of quality of the first level.

Every attribute appears the set of single properties that specify indexes of characteristic.
A complex estimation of quality Q of product is the sum of all present indexes after formulas for a separate component (Q_{com}) and system (Q_{sys}):

$$Q_{com} = \sum_{J=1}^{h} q_{j,} \; Q_{sys} = \sum_{i=1}^{N} Q^{i}_{com,}$$

Where index of qj, as well as other, has such general view:

$$q_6 = \sum_{j=1}^{5} a_{6j} \, m_{6j} \, w_{6j} .$$

There are birth-certificates at third level. A birth-certificate is totality of evaluation elements that allow defining the degree of achievement of group of attributes for each to the index separately.

In other words, a birth certificate that characterizes the index of quality can have one or a few evaluation elements. At fourth level there is evaluation elements that are elementary characteristic qj from separate property created AS.
The set of evaluation elements forms group and characteristic distributed on the corresponding processes of technology of development AS.

MANAGEMENT DISCIPLINE

The basis of this discipline consists of the classical management theory, project Production management and the standard PMBOK (Project Management Body of Knowledge) (IEEE Std.1490-2003) [1].

Management theory and also organizational management theory were developed by academician V. M. Glushkov in the 1970. This theory is field-proven in constructing technological processes in the metallurgical, ship-building, and chemical industries and also as the implementation support of mass production systems (for example, the automated management system "Lvov"). After the death of V. M. Glushkov (1982). The mathematical management theory was developed by his specialists'.

At the same time, complex systems management theory and especially production scheduling theory were developed abroad. In particular, in a company for planning and creating time schedules of large systems of works on the modernization of factories, the Critical Path Method (CRM) was developed whose basis is a graphic representation of works, the corresponding types of operations, and their execution times. Another method, namely, the network planning PERT (Program Evaluation and Review Technique), was approved in different AS and domain.

Some elements of management and planning theory are reflected in the standard PMBOK. In this standard, control processes of a project and fundamental knowledge domains grouped around problems such as initiation, planning, monitoring, management, and completion are specified. The main knowledge domain of this body it is integration which includes the conception of management of organizational activity of the team of executors of a project. It is based on methods of making decision on resources, general design problems, correctness check services of a project, and provision of the project cost specified by the customer.

These developed basic management and planning theories, standard principles of PMBOK, the series of standards ISO 9001 regulating quality management, and the corresponding methodical support of a project must become the basis of the management discipline in SE. In higher schools, a prepared curriculum of this discipline with the use of management theory will provide the preparation of highly skilled future project managers and other specialists in the field of organizational management of producing AS on an industry.

Management Documents in Information Systems

The new theoretical and applied results in management documents under the direction of author and dissertate of Zadora was done in work [9]. Basic ideas of method management documents such as:

1) formal case by documents turn in the informative system taking into account the academicians V.M. Gluchkov ideas frame, expounded in monograph "Informatics without Pipes" (1982), and method of estimation of all types of the programs resources;

2) method of forming a plan variant X works on the network graph, where $li \in L$ works, volume of qi, the Wi - kind, the R resource = (RL, RS) with the $NRi \in NR$ and taking into account law of distributing the accidental sizes $F = \{F_1..., F_r\}$, time of t planned period $[t0, T]$ and probability of completion of works taking into account the optimum plan $K(X^*) = min\ K(X)$.

A design of the system of management by documents turn (COURT) in the informative system (IS) of education in Ukraine is conducted on the given theory. Informative characteristic of documents are set in her – volume and temporal characteristics. Volume - it is a document (middle and maximal) size and quantity of documents, which act for the definite interval of time on treatment in COURT. The temporal characteristic of document determines time of his treatment in the different knots of network of being documents, transmission of them for networks and implementations of operations above them and etc.
Volume characteristics – it regular part of document from sequence of repeated groups of fields and irregular part of document, not containing repeated data structures.

Formulas of computations of characteristic of volume of documents are shown out:

Middle volume $V=l_h + n_s k_s l_s^{max}$;

maximal volume $V_{max}=l_h + n_s^{max} k_s l_s^{max}$,

where lh – size of irregular part of document;

n – quantity of lines, which are filled for the given type of documents;

ks _ Coefficient of filling;

l_s^{max} – maximal size of regular part of document.

The temporal characteristics *of* documents include:

total values of times of treatment of different types of documents in accordance with their route;

time of implementation of some operations above documents in the different *Pi* and *Pj* knots of the system;

time of transmission of documents between the different knots of treatment in Information systems.

Computations of descriptions of volume and temporal descriptions are executed in two stages.

On the first stage their values are calculated statically with supposition, that documents are processed autonomously and calculable resources for other works are not used. The second stage assumes existence of streams of documents of different types and setting. Computations of time of passing and treatment of documents are executed by formulas:

$T_i^c = V_i^g (1/R_i^g + 1/R_i^c + 1/R_{i+1}^g)$ – time, necessary on moving a document from the P_i knot in the P_{i+1} knot;

$T_i^d = t_i^1 + t_i^d + t_i^2$ – document treatment in the P_i knot time;

$T = \sum_{i-1}^{r1} T_i^c + \sum_{i=1}^{r} T_i^d$ – general time of document treatment in accordance with passing by circulation them of different routes between knots.

A design of documents turn in COURT is carried out on the base of model of informative streams of documents in IS distributed types. The primary objective of this model consists of determination of analytical dependences between the values of intensity of streams of documents in SYBD, general time of their treatment and facilities of data communication. These dependences are used for determination and estimation of number results of design of documents turn. In case of beginning turns are increased value of temporal characteristics, which rely on size of turns and boot of knots of treatment of documents. Therefore design of processes of passing documents and distributing them aggregates for treatment by some IS with knots correspond to implementation to the scientific–analytical and control-inspection management and etc.

Roles management specialists

The main issue is the management of the project management staff, implementing the project. Staff must be qualified to perform the appropriate tasks in the project; it should have advanced knowledge in all disciplines SE [9].

In infrastructure project in accordance with ISO / IEC 12207 consists of the following groups of performer's project:

– Technical and technological support (market research, acquisition of existing tools Case, employee training, etc.);

– Protection (ensuring protection and verification of information in the project);

– Technological services (maintenance processes project specifications, construction schedules, control, etc.);

– Quality (SQA-group), whose functions include planning and implementation of activities AS, discipline SE when creating the project, check work in control points, quality control of products and documents airplanes, etc.;

– Verification and validation, conducting qualification testing of aircraft or components of the product on accuracy, coordination of work plans with the manager of the claims to the airplane, validate performance requirements and test environment airplane;

– Project Manager responsible for financial and technical resources of the project, implementation of project agreements to the Customer Management develops components of the project;

– Project Manager, responsible for drafting based on the requirements, design solutions and work plans and their implementation;

– Designers and programmers responsible for the development of the design decisions and implementation of programs, documents and other output results;

– Head configuration that registers versions (variants) of the project, retains copies and versions in media access.

The main role of the project manager performs. It evaluates the ability of an employee to perform work on the design, programming, testing, etc., and is also responsible for emerging risks in the project and among the performers. An alternative to the standard view of the development of the airplane is extreme programming, when all members of the development team are equally responsible for the performance and quality of the product. Therefore, one of the important tasks of teaching students the problems SE is to prepare students to all aspects of management theory airplane knowledge to all aspects of the specialties of the discipline and technology development the domains.

PRODUCT DISCIPLINES
Basic Fundament Conveyor **Technologies**

There are approaches' to industry AS and SPF: Conveyor Assembly lines by academician V. M. Glushkov (1975) from RC [7, 9];
Conveyor by K. Czarnecki and U. Eisenecker [9] (http://www. prek.inf.tuititmenau .de/-czarn/generate);
Software factories of assembling applications by J. Greenfild, K. Short et al. (www.softwarefactories.com, blogs.msdn.com);
Continuous integration by Martin Fowler (http://www.martinfowler.com/articles/);
EPAM assembly line (Belarus) for building various types of software, improving software quality and reducing risk (http://www.epam.by);

Compositional (Assembling) programming by E.Lavrischeva for developing software products from reuses, services, artifacts, and so on [9] (http://www.sestudy.edu-ua.net).
The notion of the assembly conveyor by Glushkov's academic has been developed for many years: nowadays we have created the experimental program factory at Kiev National University based on product lines for reusable components [16-19]. Interfaces of these components contain the standardized description in certain programming languages (PL), as well as the communication interface to enable interoperability with other objects.

Producing lines in our web-site

– Product lines of services ready RCs, which saved on repository of environments web-site;
– Techniques for assembling RC from repositories into new AS, SPF;
– Models for supplying interoperability between the programs, systems and environments;
– Technologies programming in PL (C#, Basic, Java, TD of GDT – FDT) in program factory (http://programsfactory.univ.kiev.ua) at Shevchenko Kiev National University [18-21];
– E-learning to method of PL, verification, testing and assessment, described in the e-textbook "Software Engineering" [3] and so on.

Realization of the interoperability model

The applied models for system interconnection via Eclipse IDE have the following realization in ITC [20]:
– Visual Studio.NET, Eclipse implements interaction of software systems in C# and Java, respectively, according to the description of their interfaces and stubs/skeletons, transmitted data.
– Corba, Java, MS.Net support connections between heterogeneous programs in the Eclipse repository and processing the transmitted data.
– IBM VSphere, Eclipse support merging software systems built from heterogeneous components and two-way data transmission.
The basic parameters for the interoperability model M_{inter} are program, interface and message or protocol (model WCF).

Configuration model Appfabric

One of tasks of ITC site it is configuration out the file integration RC and AS. This function runs by means of configurators, which supports AS from RC as the fabric (Fig.3, [11, 21]).
It is designated by the letter A AppFabric (Factory AS) and letters B in Eclipse / TFS (RC data repository and AS), respectively. Repository stores RCs, which is a working data for Configuration. Factory (AppFabric) is an application server that provides an environment for services AS.

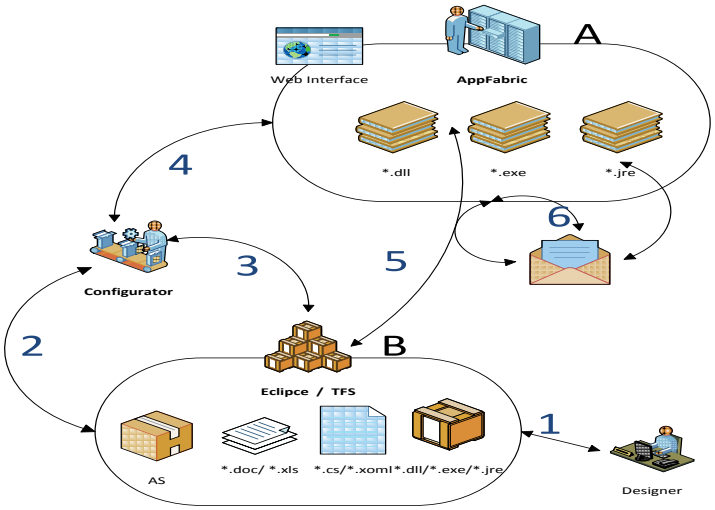


Fig.3. General Configuration model of AppFabric

The process of filling the repository is as a new component if necessary after searching Internet libraries. Developer (arrow 1) gets the action for revision or creation of a new RC, using Visual Studio and brings them to the repository. In general RC consists of two types of files - *.cs, which carry a business logic process and *. Xoml, which represent atomic or abstract objects of applied field (domain) and describe an algorithm for implementation of business logic *. cs into files. Recent arrows represent respectively:

2, 3 - action configurator by selecting an object from the repository build Eclipse;

4 - action configurator with factory programs through web-service or package orders to service;

5 – Connection repository and factory programs AppFabric;

6 - Training of package orders for fabric service.

E-LEARNING OF TECHNOLOGIES LINES

Teaching students the aspects of industry at Ukrainian universities is currently at its initial stage; to solve some education problems, we have introduced a new approach to e-learning students various aspects of SE, which assists in acquiring knowledge on software industry. We suggest learning foundation of SE with help sections of the ITC website (fig.4) [20].

E-learning Technologies lines in website:

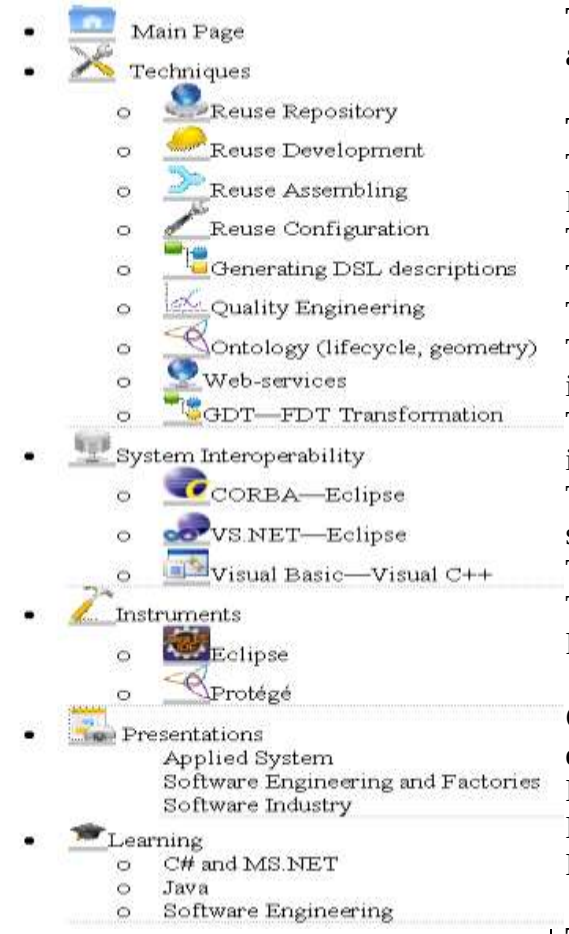
Development of components, reuses and services and their maintenance in repositories ITC;

E-assembling components on example;

E-Learning Interconnection and variability AS, SPF from RC**;**

Configuring reuses and systems into SPFs with Workflows tool in VS.Net;

E-Learning text-book “SE” on our web site http://programsfactory.univ.kiev.ua.



Technologies of developing components RC and AS.

Technologies:
Technology of service repository of components - RC,
Technology of development of RC,
Technology of assembling of RC,
Technology of configuring of RC,
Technology of generation of description of RC is in DSL,
Technology is an estimation of expenses and qual ity,
Technology of ontology will calculate. geometrie s, ISO/IEC12207 Life Cycle,
Technology of web-services,
Technology of generation ISO/IEC11404 GDT to FDT.

Cooperation of the programs, systems, environments:
Model of CORBA - Eclipse - Java,
Model of VS.Net C#- Eclipse,
Model of Basic - C#.

Tools ITC:
System Eclipse,
System Protégé.

Presentations of AS in ITC:
System of conduct of foreign business trips for N ANU,
Sliding seats about ITC, factories of the programs .
Methodologies of construction of TL.
Educating:
Technologies of programming C# VS.Net, Java,
Electronic textbook from the course "Software engineering" on website of KNU http://programsfactory.univ.kiev.ua .

Fig. 4. Keywords in the main page of the web site ITC

E-Learning Interconnection and variability AS, SPF

To study the principles of interaction between systems and environments were studied condition and capabilities of modern operating systems and development environments, AS, SPF and results of the implementation of programs and regulations. In the master work of students KNU [18, 19, 20] proposed a model interoperability of Basic and Visual, Java and C + +, Microsoft VStudio and Eclipse.

The model of interaction messages between systems and environments are realized in ITC. Model variability SPF vas developed in environment ITC on the site – Reuse Configuration [20].

E-learning variability by configuration components

Variability Model proposed is considered for simple example domain of quadratic equations solving. In classical works variants is visualized by feature diagram. It clearly demonstrates the range of possible variation points, variants and their constrains. But it's difficult to go from feature diagram to the actual implementation of specific applications in the SPF environment. Fig. 5 shows the diagram representing Variability in artifacts sub model visualized by Windows Workflow Foundation (WWF) [13, 21, 22].

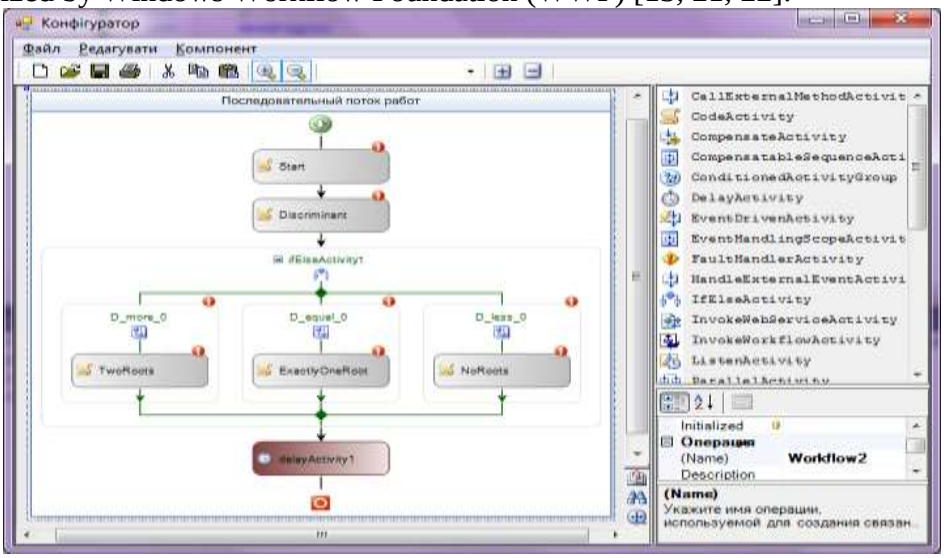


Fig. 5. Processing the configuration

In particular case quadratic equation solution the component "Discriminate" contains simple code to find discriminate ($D = (b * b) – (4 * a * c)$), implemented by developer who's responsible for producing RC in the SPF environment. Depending on the outcome of this code there are three scenarios (use case): $D > 0$, $D = 0$, $D < 0$, which are correspondent to three relevant RC "Two Roots", "ExactlyOneRoot", "No Roots".

Using WWF in Visual Studio environment the developer is allowed to insert any RC in variation points. In other words, we need a product that would support functionality of model and offer functionality to make changes in RC architecture configuration. The Configurator has been implemented within ITC, and focused on a production variety of applications configured from RC of repository (right upper corner in Fig. 5) and to add and modify the structure of AS by variation points.

E-Learning Interconnection AS, SPF

The principles of interaction between systems and environments were studied by us by examining capabilities of modern operating systems and development environments (VS.NET, Java, CORBA, and Eclipse). The objects of the study were components, AS and Spas. The main question of system adaptation across different environments was transforming different formats in the application data and usage of compiled code. In their master works, students of KNU and MIPT proposed applied interoperability models and

implemented them for pairs of systems: Visual Basic and Visual++, Java and Microsoft .NET, Microsoft Visual Studio and Eclipse [17-22]. Parameters of the interconnection model are used to exchange messages between systems and environments. Interaction between systems is based on the notion of interface, which is specified in IDL and contains definitions of data types and remote call operations. The ITC web site features three examples of interoperability between systems, which are used to demonstrate different levels of interconnection in software products:

1. Interoperability between programs created in Visual Basic and Visual C++, provided by the interface layer in form of a library, which transmits data from one program to another and transforms incompatible data types, when necessary.
2. Interoperability between Java and Microsoft .NET programming platforms, implemented by utilizing the CORBA object request broker and using interface definition language (IDL) to describe interfaces in these platforms.
3. Interoperability between Microsoft Visual Studio and Eclipse integrated development environments, provided by transmitting application data of a program, developed with Visual Studio, into the Eclipse repository, utilizing Eclipse plug-in capabilities.

E-learning SE on Web Site ITC Textbook "Software engineering"

Textbook is presented for students and management the new disciplines of SE by author and approaches' to these study and creation t for them theory and Application. General idea this text-book are the industry different domains, AS and SPF from RC, which made in different PL and Product lines. These disciplines may be creation in future designates by many specialists our information spaces [2, 20].

Textbook is presented by the languages (in Russian – www.intuit.ru and Ukraine, English – http://sestudy.edu-ua.net) for e-learning by students, magistrates and aspirants, which are specialized on department of computer science and informatics. Each areas knowledge text-book have description of general idea, question and literatures for e-studies them by students, others specialists and different users. During e-learning of this areas, examples on web site ITC and training on others instruments the students give answers on this question and put a new question in dialogs with professor. Besides that they may be elaborate a new conception and artifacts and presented them in the article or put on this web-side as he works. For sample after study many aspects of SE magistrate Alex Octrovski proposed a new aspect interconnection systems for the environments CORBA and VS.Net [18-22] and solution the task development AS from services and RC web services JEE, which done on website ITC.

CONCLUSIONS

Implemented disciplines of SE are oriented on theoretical and industrial aspects of AS development, which are necessary for e-learning of students, magistrate's and others. It is described new fundamental aspects of SE:

Essentials of disciplines SE and idea industry AS, SPF from RC.
The main thing positions of all offered disciplines of SE.
Conceptions of AS lines producing on conveyor principles.
Assembling different RC on Technological Lines on website ITC.
New models of variability, interoperability and changing RC in AS.
Configuring components and evaluation of quality and cost of the product.

Ontology's presentation knowledge about new disciplines SE;
Ontology of life suckle of standard 12207 and so on.

The e-learning fundamental aspects textbook of SE was done on website ITC and the last theoretical aspect SE was given in monographic [9].

REFERENCES

[1] E. Lavrischeva, "Classification of Software Engineering Disciplines," Cybernetics and Systems Analysis, Vol. 44, No. 6, 2008, pp. 791-796.
[2] E. Lavrischeva, "Software Engineering (in Ukrainian),"Akademperiodika, Kiev, 2008, 319 p.
[3] E. Lavrischeva, G. Koval, L. Babenko, O. Slabospitska and P. Ignatenko, "New Theoretical Foundations of Production Methods of Software Systems in Generative Programming Context," IK-2011, Software Institute NANY, 2011.- 277 p. http://www.nbuv.gov.ua/
[4] E. Lavrischeva, V. Grischenko, "Assembly Programming. Basics of Software Industry (in Russian)," 2nd Edition, Naukova Dumka, Kiev, 2009, 371 p. http://www.twirpx.com/
[5] E. Lavrischeva, "Formation and Development of the Modular-Component Software Engineering in Ukraine ", Akademperiodika, Kiev, (in Russian) 2008.- 31 p.
[6] E. Lavrischeva and V. Petruchin, "Methods and Means of Software Engineering," (in Russian) 2007, 415P., http://www.intuit.ru/ and . http://www.twirpx.com/
[7] E.Lavrischeva, "Methods Programming. Theory, Engineering, Practice", Naukova Dumka, (in Russian) Kiev, 451 P. 2006.
[8] Lavrischeva E., Ostrovski "A. General Disciplines and Tools for E–Learning Software Engineering". Conf.:9th International Conf. ICTERI–2013 "ICT in Education, Research and Industrial Applications" ISSN 1865-0929, Springer , p.212-229 (2012).
[9] Lavrischeva E.M. "Software Engineering Computer Systems. Paradigms, Technologies, CASE-tools", (in Russian), Naukova Dumka, Kiev, 282 P.(2013)
[10] Lavrischeva E., Ostrovski A.: New Theoretical Aspects of Software Engineering for Development Applications and E-Learning. In: Journal of Software Engineering and Applications, vol. 6, pp. 34–40 (2013).
[11] Slabospitskaya, O., Kolesnyk, A.: The Model for Enhanced Variability Management Process in Software Product Line. In.: Mayr H.C., Kop C., Liddle S., Ginige A. Information Systems: Methods, Models and Applications. Revised selected papers of 4th International United Information Systems Conference (UNISCON 2012), pp. 162–171. Yalta, Ukraine (2012)
[12] 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.
[13] I. Radetskyi, "One of Approaches to Maintenance interconnection Environments Visual Studio and Eclipse (in Ukrainian)," Problems in Programming, Vol. 2, 2011, pp. 45-52.
[14] K. Pohl, G. Böckle and F. J. Linden, "Software Product Line Engineering: Foundations, Principles and Techni- ques," Springer-Verlag, New York, 2005.
[15] E. Lavrischeva, A. Dzubenko and A. Aronov, "Conception of Programs Factory for Representation and E-Learning Disciplines of Software Engineering," 9th International Conference ICTERI, ICT in Education, Research and Industrial Applications, Integration, Harmonization and Knowledge Transfer, Ukraine, June 17-21, 2013, http://ceur-ws.org/Vol-1000/

[16] A. Aronov and A. Dzubenko, "Approach to Development of the Students' Program Factory (in Ukrainian)," Problems in Programming, Vol. 3, 2011, pp. 42-49. http://www.isofts.kiev.ua/
[17] V.Grischenko "Object-Component Designing Method for Software Systems"(in Ukrainian), In: Problems in Programming, vol. 2, pp. 113–125. Akademperiodika, Kiev (2007) .
[18]Lavrischeva K.M. Component Programming. Theory and Practice.(in Ukrainian)," Problems in Programming, Vol. 4, 2012, pp. 3-12, www.isofts.kiev.ua.
[19] Lavrischeva K.: "Formal Fundamentals of Component Interoperability in Programming" In: Cybernetics and Systems Analysis, vol. 46, no. 4, pp. 639–652. Springer, Heidelberg (2010).
[20] Lavrischeva E., Zinkovich V., Kolesnik A. et al.: "Instrumental and Technological Complex for Development and Learning Design Patterns of Software Systems". State Intellectual Property Service of Ukraine, copyright registration certificate No. 45292, 103 pp. (2012) (in Ukrainian)
[21] E. Lavrischeva, A. Ostrovski and I. Radetskyi, "Approach to E-Learning Fundamental Aspects of Software Engineering," Proceedings of ICTERI-2012: ICT in Education, Research and Industrial Applications, 2012, pp. 176-187. http://senldogo0039.springer–sbm.com/ocs/home/ICTERI2012.-CEUR-WS-paper-17-p-176-187.pdf/
[22] A. Kolesnyk and O. Clabospitskaya, "Tested Approach for New Theoretical Aspects of Software Engineering for Development Applications and E-Learning, Variability Management Enhancing in Software Product Line," Conference ICTERI-12. http://ceur-ws.org/Vol-848/ICTERI-2012-CEUR-WS-paper-31-p-155-162.pdf