

# Conception of Programs Factory for Representing and E-Learning Disciplines of Software Engineering

Ekaterina Lavrischeva<sup>1</sup>, Artem Dzyubenko<sup>1</sup> and Andrey Aronov<sup>1</sup>

<sup>1</sup> Taras Shevchenko Kiev National University, Kiev, Ukraine

lavryscheva@gmail.com, asmer@asmer.com.ua

**Abstract.** The paper presents a new idea of knowledge representation for students studying software engineering by developing artifacts and software components accumulated in libraries or repositories for further reuse. The idea is based on the concept of assembly line by V. Glushkov, further elaborated by Soviet and foreign specialists (A. Ershov, V. Lipaev, J. Greenfield, G. Lenz, Y. Bai, M. Fowler). The paper introduces the elements of program factory: reusable components, their interfaces, and assembly lines for designing and assembling complex software products from components. It is shown that modern operating environments provide prerequisites for such factories. The students' program factory, implemented at Taras Shevchenko Kiev National University, is then described. The technologies behind the factory, as well as its goals are studied.

**Keywords.** Artifact, reusable component, applied system, interface, assembling line

**Key terms.** Methodology, Technology, Process, Qualification

## 1 Introduction

The paper presents the elements of programs factories: reusable components (RC) and their interfaces, assembly lines for designing and assembling complex programs from RCs. It is shown that modern operating systems provide tools for creating specialized programs factories (MS.NET AppFabric, SOAFab, SCAFab, IBM VSphere, CORBA, etc.). Factories are built by different commercial structures for software product development, as well as for studying purposes (e.g., the programs factory implemented in Taras Shevchenko Kiev National University for the assembling artifacts from disciplines that are studied at the university). The purpose of such factories is to study computer science, software engineering, programming technology and software systems with electronic textbooks, develop applied projects and thus train highly competent professionals in software industry.

Over last decades a huge amount of various programs has been accumulated in the informational world that may be used as end products for complex programs devel-

opment. Therefore, a new approach has been formed in programming, namely reusability – reuse of ready-made software resources (reuses, assets, services, components, etc.), hereafter referred to as reusable components (RC). This term is used in informational world to represent new knowledge acquired over researches in certain fields of computer science. Being needed for somebody, it may be used in solving certain problems concerning similar artifacts as well as for development of new software systems (SS), applied systems (AS) or software product families (SPF).

All software artifacts and RCs may be stored in public warehouses (libraries, repositories) that may be searched by professionals to identify the necessary data for implementation in their own research activities. That is, reuse of ready-made resources becomes a capital-intensive activity in the field of software engineering and it is particularly important that universities possess so-called factories for scientific artifacts, programs and RCs needed by other students and professionals. With these factories, students may participate in industry development of scientific artifacts for mass use [1–3].

Based on such innovative ideas, Prof. E. Lavrischeva proposed fourth-year students at KNU to establish the first programs factory over the course of theoretical and practical labs on software engineering. This factory is focused on artifacts, software development, and repository maintenance. Students' programs factory operates on the web site (<http://programsfactory.univ.kiev.ua>) since December 2011. The web site has been visited by more than 5,000 people – students, scientists and teachers.

## 2 Establishing Programs Factory

**History of Software Industry in USSR.** An idea of industry for computers and supporting software has been formulated by Academician V. Glushkov at Cybernetics Institute of NAS of Ukraine in 1960s-1970s. Under his guidance, a family of small computers 'Mir' (1967–1975) has been developed together with a language of analytical and formula transformations for solving differentiation, integration and formula calculus problems. In addition, other computers (Dnepr, Dnepr-2, Kyiv-67, 70, macro-conveyor etc.) have been elaborated with auto code-typed programming languages (PL) to develop information processing programs and automated management systems (AMS) for various organizations and enterprises. For their development, the problems of software quality and increase of production of reusable components have been investigated with computers at state institutions [4–6].

In 1975 V. Glushkov first formulated the concept of assembling conveyor consisting from technological lines for software products (SP). The core of Glushkov's paradigm is to accelerate the transition from programming as an art to industrial methods of SP production in order to solve various economical, business, scientific problems with automated management systems.

Then (1978), software development as joint scientific-technical production and the projects requesting for automating SP creation have been decreed. The first pilot factory for software engineering was established for mass production of various AMSs (1978, Kalinin); but, because of lack of ready-made programs and immaturity of pro-

programming technology for industry production, the factory had lasted for two years and was closed [7]. Nevertheless the experiments aiming to elaborate programming automation tools were lasting until the collapse of the USSR.

V. Glushkov's idea concerning programs factories is now running at the several industrial factories explored by different authors theoretically and in practice [8–13]:

- Conveyor by K. Czarnecki and U. Eisenecker
- Software factories for assembling applications by J. Greenfield, K. Short et al.
- Continuous integration by Martin Fowler
- EPAM assembly line for building various types of software, improving software quality and reducing risk
- Automated assembling of the multi-language programs in heterogeneous environments by Y. Bai (VC++, VBasic, Matlab, Java, Visual Works, Smalltalk and others)
- Command development and assembling programs for software projects in MS Visual Studio Team Suite based on contracts
- G. Lenz's program factory utilizing UML in .NET
- Assembling, configuration and certification of global scientific-technical software on the European Grid factory
- Compositional (assembling) programming by E. Lavrischeva for developing software products from reuses, services, artifacts, and so on
- Experimental KNU factory

Careful analysis allowed singling out the key components of programs factories [13]:

- Prepared software resources (artifacts, programs, systems, reuses, assets, components, etc.)
- Interface as a mediator between two components, containing passport information for heterogeneous resources in a certain specification language (IDL, API, SIDL, WSDL, RAS, etc.)
- Operating environment with system facilities and tools supporting assembly lines with heterogeneous software resources
- Technological lines or product lines for mass production and assembling products.
- Methods of product development

The elements listed are the fundamentals of SP production industry at the factories that operate at major foreign software companies such as Microsoft, IBM, Intel, Apple, Oberon etc. At the KNU students' programs factory, these fundamentals are implemented as certain lines for programs and scientific artifacts, which are the best among student-developed programs factories.

The authors of the KNU programs factory consider it an integrated infrastructure for organizing production of mass usage SPs that are needed for customers and users from the fields of computer science, state government, commerce etc. The factory is equipped with technology lines (TL) or product lines [7, 12], as well as a collection of products, tools and services needed for automated processes execution over these

lines in modern operational environments (MS.NET, IBM, Sun Microsystems and so on).

**Web Site of KNU Programs Factory.** The main objectives for the web site are: improving students' skills in software development using the system for exchange of KNU students' certified software products and scientific artifacts; increasing SP quality and reliability; and learning to support the methods of SS industrial production.

The web site presents lifecycle models, SP building lines, the line for program production with the help of MS.NET platform, and examples of student programs. Obligatory requirements are maintained during certification to store software products in the repository of the factory (the library pool).

The main activities on the factory site are:

- Organizing program and artifact development
- Familiarizing students with tools and methods for program and SS development
- Representing students' software products in the repository
- Citing excerpts from articles and textbook materials concerning various disciplines

The factory is equipped with tools that allow broadening its functionality by specifying new software artifacts and storing them in the repository for further reuse. Each artifact is uniformly documented based on WSDL, IDL standard used in Grid global project.

### 3 Factory Lines and Components

**Development of Technological Lines.** Technological lines are created at the technological pre-production stage [7, 12] before SS production. They include activities for designing the TL scheme from processes and actions that determine the processing order of SS elements with appropriate technological modules (TM) or programming systems. The basic requirement of TL engineering imposed on production of programs and components is to assemble TLs from lifecycle processes meeting problem domain goals using standard tools, TMs, and the system of regulatory documents. The TL is then supplied with ready-made components, tools and instruments that generate and implement specific functions or elements, as well as the management plan for processes for changing states of the elements and providing quality evaluation [13-16].

The *RC model* for component-based development has the following specification:

$$RC = \{T, I, F, Imp, S\}, \quad (12)$$

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

Basic operations over components are:

- Specifying components and their interfaces (pre- and post-conditions, which must be satisfied by the caller) in such languages as IDL, API, WSDL, etc.

- Maintenance of components, reuses and artifacts in the component repository for search, change and future integration into applied systems
- Integration of components into applications, domains, applied systems, software product families, etc.

All aspects of RC development and their usage in SP and software product families are goals for reusability disciplines and building material for these systems.

*Applied system* is a collection of software means (or functions of SS), including general tools (DBMS, protection systems, system services, etc.), constructed subsystems or components together with the tools for marshalling from one AS to another [14].

**Product Lines at SEI.** Product lines and product family (PF) is defined in ISO/IEC FDIS 24765:2009 (E) – Systems and Software Engineering Vocabulary: “*Product line* is a set of products or services that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. Synonym: *product family*”.

SEI professionals propose the two models for representation of activities for SS development, namely engineering and process ones.

The engineering model assumes the three activities, namely RC development, PF development through RC configuration and management of the both above activities.

The activity for RC development presupposes PF scoping and production planning of SS collection accounting for the context of SS usage, production limitations and the chosen strategy. The PF development activity includes designing each specific SS implementation based on the set of developed RC, and building software systems according to the PF implementation plan. The management activity is based on balancing activities for RC development and PF maintenance tasks, and includes both organizational and technical management.

According to the process model, a set of processes is performed at the two levels, namely domain engineering, being also referred to as the development “for reuse”, and application (or SS) engineering – the development “with reuse” [15]. The last one is performed over assembly line using ready-made RCs to shorten time and increase SS availability. Therefore, configuring the product family from RCs according to the specific requirements and needs of a particular market segment is the final one in the cycle of production activities.

#### 4 KNU Students’ Programs Factory

From the theoretical standpoint, program factories are based on the assembling conveyor that includes a collection of various more or less complex production lines for software artifacts, programs and RCs. Conveyor lines contain process execution using system tools or technological modules that automate process execution for obtaining interim or final results.

From the perspective of information technology, the factory provides the data processing toolset for the transition from individual programming of particular re-

sources to the industry of mass-usage SP. The factory increases SP development productivity during each lifecycle process due to use of RCs that possess the necessary functionality with due quality guaranteed by their developers. The assembling (composition, configuration) line may benefit through reduction of efforts because of use of readymade artifacts or RCs stored in the repository.

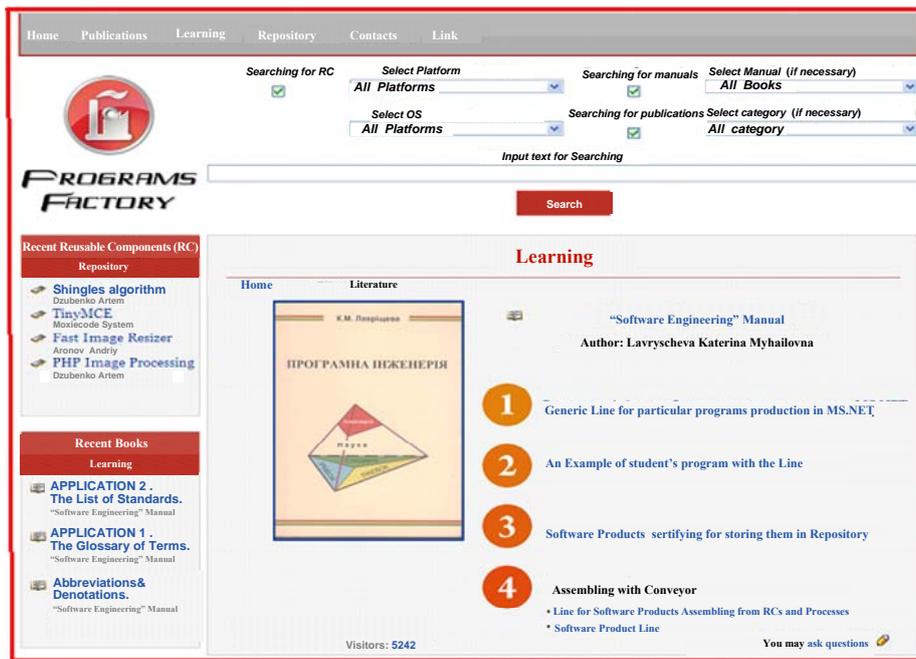


Fig. 1. Main page of KNU programs factory

**Programs Factory Technological Lines.** The TL development is as a rule matched with some lifecycle, e.g., implemented in the MS.NET environment with guides, frameworks, programming languages, common libraries templates, system tools that support new subject-oriented languages such as Domain Specific Language, etc. For complex SP development, an *assembly (compositional) line* is proposed, as a tool for composing RCs using their interfaces from various interim libraries within development environments, as well as from RC repositories (Fig. 1). The figure shows the main page of the web site of the factory: lines 1, 2, 3, 4 and a text-book for e-learning fundamental aspects of software engineering [2].

The assembling conveyor has four implemented technological lines [1, 2] that, since 2011, aid in artifacts and program development. The structure of these lines is designed according to the technology explored by the author (Prof. E. Lavrischeva) in 1987–1991 [7, 12-16]. Simple TLs in factory are as follows:

- E-learning C# in VS.NET environment (Fig. 2)
- Saving components into corresponding repositories and selecting them from the repository to meet market demands on specific SPs

- Assembling or configuring RCs into complex program structures (SP, FS)
- E-learning basic knowledge on software engineering with the dedicated e-textbook

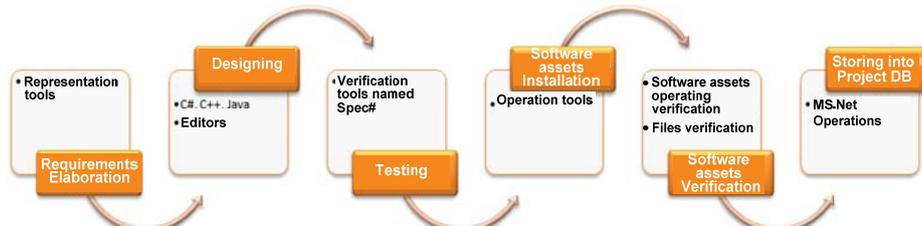


Fig. 2. Chart for components design in VS.NET environment

**Web Site Lines.** The depicted TL for learning C# programming is designed according to the ISO/IEC 12207 standard of lifecycle processes while allowing for .NET specifics as to how to perform the following design tasks:

- Exploring demands on software products, singling out features and methods for automated generation
- Fixing requirements on implementations of SP functions for the domain
- Specifying software elements or artifacts, documenting their passport data and interfaces with a WSDL-like language
- Storing created software artifacts and programs in the repository

The line for repository maintenance includes the mechanisms for stored RCs being uniformly documented with their WSDL passports, as well as the tools for selecting ready-made RCs and artifacts from the repository based on their passport data, functions and relevant solution examples (see Fig. 3). This line is pertained to the processes for quality assessment of artifacts or programs created, verifying them from the perspective of reliability and quality.

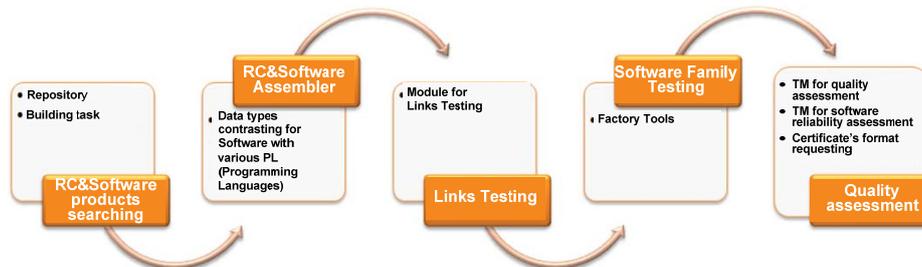


Fig. 3. Flowchart of the line for component search in repository

The line for AS development at the factory includes the tools for engineering of specific required components using various PLs, programs and artifacts, both just developed and selected from repository; the line also supports building multilingual RCs and marshalling non-relevant types of data exchanged by the components [11]. The constituents of this line include standard tools for building or configuring multilingual components, testing both the sample components and the links between RCs

being composed together, as well as the tools for reliability and quality assessment and certification of the product obtained.

The line for remote e-learning with the dedicated *Software Engineering* textbook [14, 15] is now actively used in studying topics of this discipline with KNU students. This line may also be used for independent studying in other high school institutions where software engineering or computer science courses are established.

**Design of Programs Factory.** At the students' factory, Visual Studio .NET licensed by KNU is used as the foundation for the programs factory; capabilities of MS.NET platform in providing tools for multilingual AS development and support using the components in C#, C++, Basic, etc., are utilized as well. Consequently, third-party software developers for the factory may not be limited to the choice of a single PL.

The factory web site is developed using PHP programming language, and, for its external representation, HTML5, CSS3 and JavaScript. The system core is independently engineered by the authors with relying on known web frameworks [2].

## 5 E-learning SE Disciplines

Teaching students the aspects of software industry at Ukrainian universities is at its initial stage. To solve several education problems, we have introduced a new approach to e-learning various aspects of SE, which assists in acquiring knowledge on software industry.

A new concept for breaking down the software engineering disciplines (Fig. 4), which is necessary in industrial factory production, was proposed [7, 14]. Basic goals of software engineering disciplines are as follows:

- Scientific discipline consists of the classic sciences (theory of algorithms, set theory, logic theory, proofs, and so on), lifecycle standards, theory of integration, theory of programming and the corresponding language tools for creating abstract models and architectures of the specified objects, etc.
- 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
- 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
- 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.
- Product discipline consists of product lines, utilizing software resources (reusable components, services, aspects, agents, etc.), taken from libraries and repositories; it also contains assembling, configuring and assessing quality of software

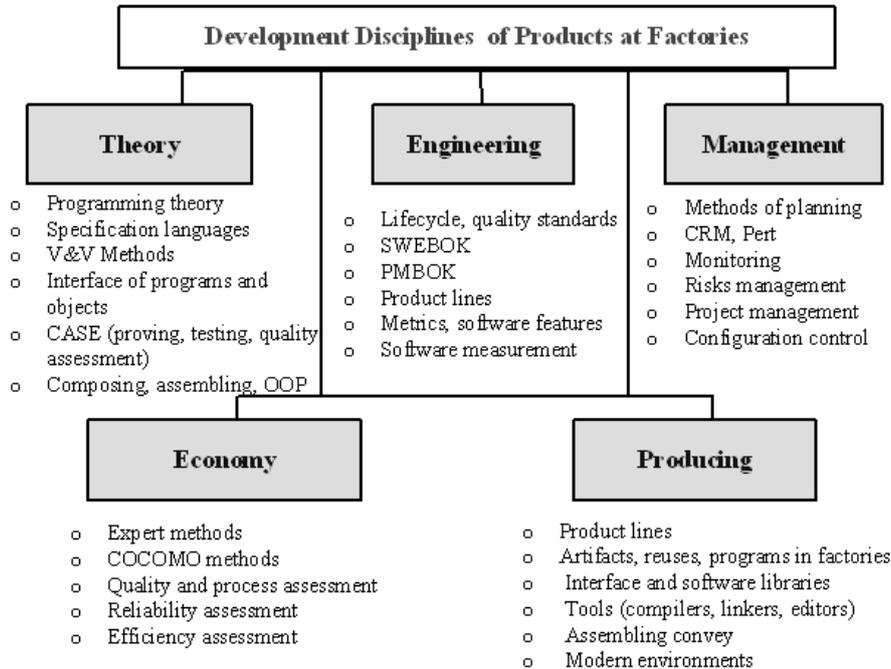


Fig. 4. Software engineering disciplines

In our opinion, all the SE disciplines considered above and their theoretical foundations must become the independent subjects to be taught to students specializing in the field of SE with the orientation toward the industrial production of SPs from readymade components (reuses, services, assets, and so on) [17]. The production cycle will be repeated in case of bringing changes in the product structure as it is done in technology of continuous integration by M. Fowler and in AppFabric in MS.NET (Fig. 5) [18].

**Approach to Teaching SE.** The SE educational course must include intertwined theoretical and practical fundamental positions and achievements in software development and integration [15, 17, 19]. The directions of the SE teaching course are as follows:

- Base concepts, principles and methods that constitute the basis of SE knowledge and technology of programming (e.g., the five SE disciplines, life cycle, project management, quality, configuration) and proved their productivity in practice
- Mathematics of systems analysis for subject domain with the use of elements of theory of algorithms, logic and semantics of programming for the formal design of key notions of the domain, reflection of their communications and relations in case of formal task of their models and SP architecture

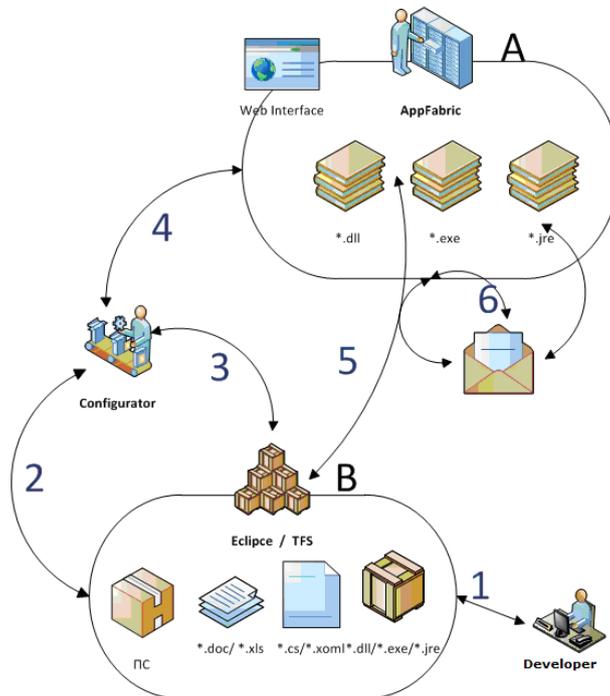


Fig. 5. Structure of Microsoft AppFabric

- General principles and methods of designing programs, software products and software product families using ready-made objects, components, services, aspects, etc.
- Modern applied tools for representation of software products, which are widely used by professionals in research and development (systems analysis, decompositions, architecture, design, OOP, ontology, etc.)
- Methods of measuring quality of software products
- Development environments (MS.NET, IBM, CORBA, Protégé, Eclipse, etc.)
- The directions of e-learning are summarized in the *Software Engineering* textbook. The students learn and apply them on practice with the use of system tools, artifacts or programs developed by other students. Certain students' achievements are certified by the teacher and can be added to the repository of the factory

In terms of teaching SE, the author's first textbook (2001), written in Ukrainian [18], is dedicated to the foundations of SE from Curricula-2001; the second textbook in Russian (2006) teaches the basics of Curricula-2004 [12]; the third textbook is developed for modern approach towards teaching SE [15], including topical outlines of some of the above-mentioned disciplines and fundamental aspects such as reliability and quality engineering. In the new textbook, basic elements and engineering tools are presented for development of various target SE objects and lifecycle processes, methods for design and management of collectives of executors, quality, terms, and

cost. The textbook on the web site describes new SE disciplines and fundamental aspects of SE. The structure of the textbooks corresponds to the typical Curricula-2004 program, and to the modern requirements on the subject imposed by the program of the Ministry of Education and Science of Ukraine (2007).

## 6 Conclusions

The result of the authors' work is an experimental programs factory web site, accessible on the Internet using address <http://www.programsfactory.univ.kiev.ua/>. This web site is proposed for e-learning product line development at the high school institutions on the specialties of informatics, computer sciences, information systems and technology.

The following concrete lines are established at the factory:

- Production of reusable components and artifacts
- Development of console applications, DLL component libraries, local Windows applications with C# in VS.NET
- Developing Java programs (a manual by I. Habibulin)
- Assembling programs from RCs in MS.NET environment
- E-learning software engineering with dedicated textbooks on the web site in Ukrainian ([sestudy.edu-ua.net](http://sestudy.edu-ua.net)) and in Russian ([intuit.ru](http://intuit.ru))

The prospects of future factory evolution are its further adjunction with new resources in the field of software engineering being yet prepared by the students, namely:

- Description of the process of development of complex programs and SS using DSL language (Eclipse-DSL, Microsoft DSL Tools)
- Transformation of general data types into fundamental data types from the perspective of the standard ISO/IEC 11404-2007 generation tools
- Ontological representations of new disciplines for study (e.g., computational geometry, lifecycle domains, verification)
- New applied product lines for business developed with appropriate mechanisms;
- SEI product lines approach, and so on

## References

1. Lavrischeva, E.: Concept of Scientific Software Industry and Approach to Calculation of Scientific Problems. *Problems in Programming*, 1, 3–17 (2011) (in Ukrainian)
2. Aronov, A., Dzyubenko, A.: Approach to Development of Students' Program Factory. *Problems in Programming*, 3, 42–49 (2011) (in Ukrainian)
3. Anisimov, A., Lavrischeva, E., Shevchenko, V.: On Scientific Software Industry. Technical report, Conf. Theoretical and Applied Aspects of Cybernetics. (2011) (in Ukrainian)
4. Glushkov, V., Stogniy, A., Molchanov, I.: *Small Algorithmic Digital Computer*. MIR, 11 (1971) (in Russian)

5. Glushkov, V.: Basic Research and Technology Programming. *Programming*, 2, 3–13. (1980) (in Russian)
6. Kapitonova, U., Letichevsky A.: Paradigms and Ideas of Academician Glushkov. *Naukova Dumka*, Kiev (2003) (in Russian)
7. Lavrisheva, E.: Formation and Development of the Modular-Component Software Engineering in Ukraine. V. Glushkov Institute of Cybernetics, Kiev (2008) (in Russian)
8. Czarnecki, K., Eisenecker, U.: *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston, MA (2000)
9. Bai, Y.: *Applications Interface Programming Using Multiple Languages: A Windows' Programmer's Guide*. Prentice Hall Professional, Upper Saddle River (2003)
10. Greenfield, J., Short, K., Cook, S., Kent, S.: *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, Hoboken (2004)
11. Lavrisheva, E., Koval, G., Babenko, L., Slabospitska, O., Ignatenko, P.: *New Theoretical Foundations of Production Methods of Software Systems in Generative Programming Context*. Electronic Monograph, UK-2011, 67, VINITI RAN, Kiev, Moscow (2012) (in Ukrainian)
12. Lavrisheva, E., Grischenko, V.: *Assembly Programming. Basics of Software Industry*. 2nd ed. *Naukova Dumka*, Kiev (2009) (in Russian)
13. Andon, P., Lavrisheva, E.: Evolution of Programs Factories in Information World. *News of NANU*, 10, 15–41 (2010) (in Ukrainian)
14. Lavrisheva, E.: Classification of Software Engineering Disciplines. *Cybernetics and Systems Analysis*, 44(6), 791–796 (2008)
15. Lavrisheva, E.: *Software Engineering*. *Akademperiodika*, Kiev (2008) (in Ukrainian)
16. Lavrisheva, E.: Theory and Practice of Software Factories. *Software–Hardware Systems. Cybernetics and Systems Analysis*, 47(6) 961–972 (2011)
17. Lavrisheva, E., Ostrovski, A., Radetskyi, I.: Approach to E-Learning Fundamental Aspects of Software Engineering. In: Ermolayev, V. et al. (eds.) *Proc. 8<sup>th</sup> Int. Conf. ICTERI-2012*, CEUR-WS, vol. 848, pp.176–187, CEUR-WS, online (2012)
18. Kolesnyk, A., Slabospitskaya, O.: Tested Approach for Variability Management Enhancing in Software Product Lines. In: Ermolayev, V. et al. (eds.) *Proc. 8<sup>th</sup> Int. Conf. ICTERI-2012*, CEUR-WS, vol. 848, pp.155–162, CEUR-WS, online (2012)
19. Babenko, L., Lavrishcheva, E.: *Foundations of Software Engineering*. *Znannya*, Kiev (2001) (in Ukrainian)