Global Crisis as Enterprise Software Motivator: from Lifecycle Optimization to Efficient Implementation Series

Dr. Sergey V. ZYKOV, Ph.D.
School of Software Engineering, National Research University Higher School of Economics
33 Kirpichnaya Str., Moscow, 105187, Russia

ABSTRACT

It is generally known that software system development lifecycle (SSDL) should be managed adequately. The global economy crisis and subsequent depression have taught us certain lessons on the subject, which is so vital for enterprises. The paper presents the adaptive methodology of enterprise SSDL, which allows to avoid "local crises" while producing large-scale software. The methodology is based on extracting common ERP module level patterns and applying them to series of heterogeneous implementations. The approach includes a lifecycle model, which extends conventional spiral model by formal data representation/management models and DSL-based "low-level" CASE tools supporting the formalisms. The methodology has been successfully implemented as a series of portal-based ERP systems in ITERA oil-and-gas corporation, and in a number of trading/banking enterprise applications for other enterprises. Semantic network-based airline dispatch system, and a 6D-model-driven nuclear power plant construction support system are currently in progress. Combining various SSDL models is discussed. Terms-and-cost reduction factors are examined. Correcting SSDL according to project size and scope is overviewed. The so-called "human factor errors" resulting from non-systematic SSDL approach, and their influencing crisis and depression, are analyzed. The ways to systematic and efficient SSDL are outlined. Troubleshooting advises are given for the problems concerned.

Keywords: Enterprise Software System, Software Lifecycle Model, Software Development Methodology.

1. INTRODUCTION

We are going to focus on the reasons for the "crisis" and "depression" in software development area. The "crisis" phenomena occurred in software development relatively long ago, approximately since 1960-s. Let us analyze the reasons for the current "crisis" and the subsequent "depression". Software product lifecycle, which the industry had just started moving toward, was anarchic in many ways, since a uniform, systematic approach had been absent. At that time, software development did not allow precise variation of the "project triangle" parameters. In fact, the software had been developed in an "artisan" way, with a build-and-fix approach as the core "methodology". Thus, systematic approach to SSDL, and responsibility for the deliverables should be required.

The following decade revealed that the software development process started becoming rather a science than an art; however, it had not become a production yet, due to imperfect technologies. The era of unique, "hand-made" software projects from certain gifted programmers had passed away. Large software R&D centers appeared, one of the most well known examples of which is the Software Engineering Institute of the Carnegie-Mellon University (www.sei.cmu.edu/).

The value of software, as compared to hardware, had increased tremendously. Mission-critical software systems appeared (e.g., for military and life-support applications).

However, the software crisis, which started in the 60-s, lasted much longer and had a deeper nature than that in material manufacturing industries (construction, automobile production etc.). Lacking "universal" methodology, the so-called "silver bullet" for software development, explicitly indicated that the crisis has not been overcome, and that the "depression" started.

To conquer the crisis, we need to optimize the SSDL by systematically approaching all of its processes. The methods of software engineering (SE) methods and tools can help in this case, since the discipline approaches software development issues in a systematic way.

The SE approach is chiefly oriented on "serial production" of large-scale (with terabytes and petabytes of data), complex (thousands of files, tens of modules, hundreds of components), high quality, architecturally heterogeneous, interoperable software systems (online multiuser distributed interaction, virtualization, data warehousing etc.) [4,6]. Other architectural aspects include portal-based software systems, remote services, etc. The system quality is measurable, by the following "dimensions": reliability, security, fault tolerance, ergonomics, usability, reuse, documentation, maintainability. Heterogeneous software systems imply versatile architectures, data bases and warehouses, as well as structure degree ("flat" relationships, scanned documents, multimedia data etc.). Let us focus on optimizing SSDL on the basis of SE methods and tools.

2. SOFTWARE DEVELOPMENT AND MATERIAL PRODUCTION: COMPARING THE LIFECYCLES

Let us treat constructing a software system as an "industrial" production of a large and complex material object (such as an automobile, a bridge, a skyscraper, etc.). Although there is a certain similarity, a number of significant differences also take place. Such a conclusion has been drawn by a NATO Software Engineering Conference back in 1960-s [7]. The conference revealed that software cannot be built according to the same principles as material products, since their lifecycles are fundamentally different in a number of ways. However, the major software parameters (project terms, product cost, size, quality) can be measured formally, and SSDL can be managed by means of scientifically approved methodologies.

On the whole, we recommend following a distinct and logically sound process of step-by-step software functional specification, including conceptual modeling, analysis and design (with software operational parameter monitoring), prototyping, implementation and maintenance. It is worth to emphasize that the software quality and reliability are determined by the rate of residual (rather than that of fixed) errors, and by the total expenses for restoring the product state after a failure.

More specifically, the software model and its prototype should not necessarily be reliable. The software release may contain errors; however, it does not catastrophic for the product, or its resign, as in case of a certain automobile model. Even after a software crash, it is often sufficient to just restart it rather than reconstruct/ reproduce it. Software errors are accumulated in time. Error detection is challenging, and building am error-free software systems requires different methods from the material object construction. Finally, the "brute force" approach is not quite applicable to software. For example, doubling data channel throughput would not guarantee its reliability. However, making bridge trestle two times wider would result in a deliberately reliable product.

Analyzing the SSDL, we arrive to a conclusion that maintenance is its most specific phase, as compared to material production. Software reuse processes (e.g., for design and implementation) are essentially iterative and incremental. Software changes are more serious and radical than the changes in material objects (e.g., it is known that a building or a bridge often can be used for many decades at negligible maintenance costs).

The software on the whole (both on the mass scale, and on the enterprise scale) also has got a number of fundamental differences as compared to material production. For example, the SSDL is often essentially shorter than a material object lifecycle, since software ages much faster, and its retirement is economically preferable to maintenance.

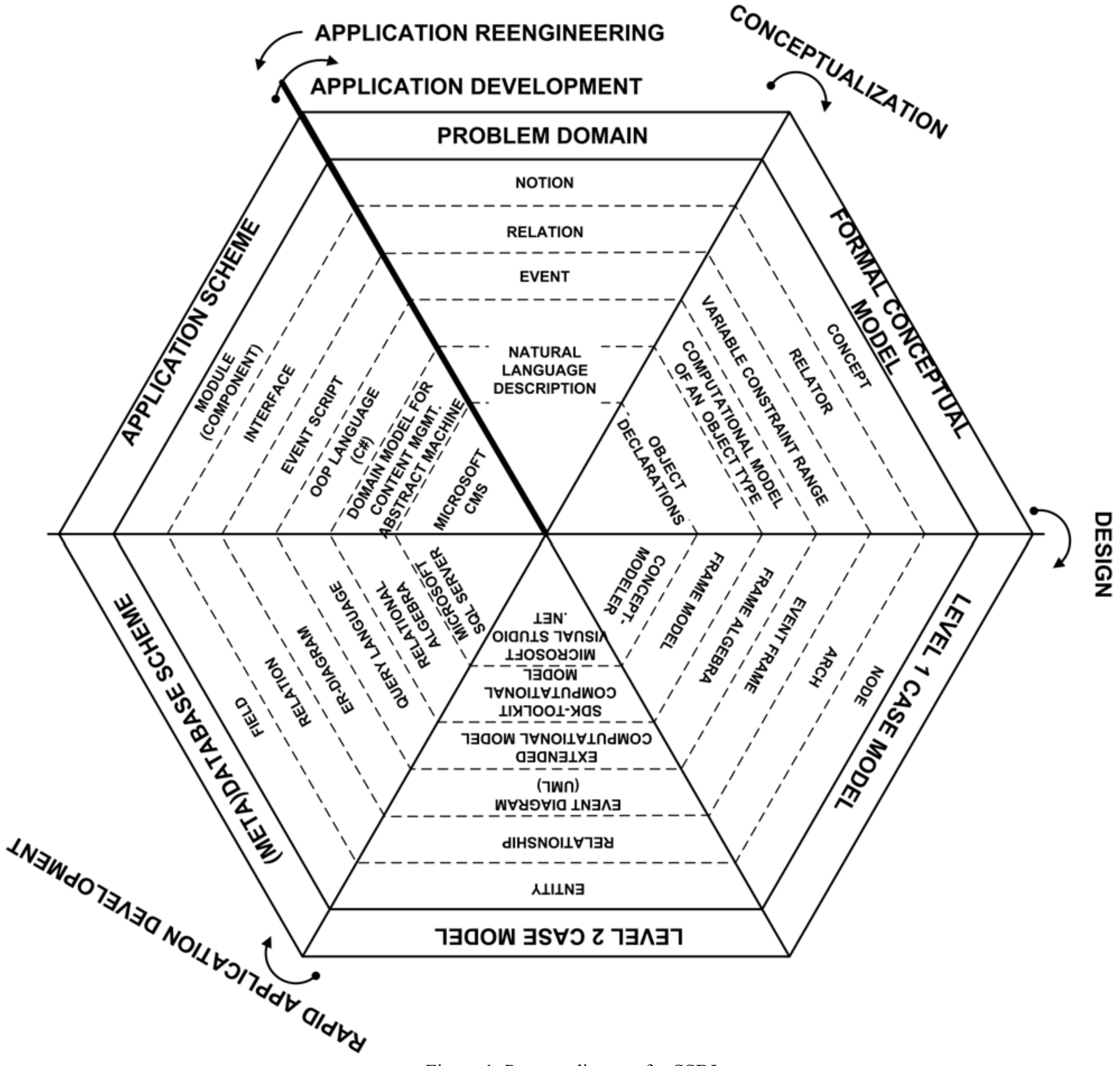


Figure 1. Process diagram for SSDL

The accumulated individual/team experience in software development (irrespective of the project role) does not always result in system quality increase due to rapid changes of complex platforms.

So, software development has both similarities and essential differences as compared to material production. However, industrial production of large and complex software systems with high quality, reliability, efficiency (due to product artifacts reuse), usability and other operating parameters is possible under a rigorous science and technology basis of SE methods and tools.

3. SOFTWARE DEVELOPMENT: OPTIMIZING THE LIFECYCLE MODELS

Every SDDL stage can be optimized, including requirement analysis, product specification, design, implementation, maintenance and retirement.

The SSDL optimization methodology is based on close integration of models and supporting enterprise-level methods and service tools. The models for problem domain and computing environment are built on rigorous formal theories, while those for other lifecycle stages are more heuristic and pragmatic. Therewith, the family of the software service tools used contains both traditional CASE and "lower" level instruments, which integrate the model and the software components in heterogeneous software systems.

The generalized methodology of integrated software system development (fig.1) provides iterative bidirectional component-wise development of open, expandable heterogeneous software systems in global environment, allowing data consistency and integrity control.
During the SSDL, the transformation of the heterogeneous information systems are transformed from problem domain concepts to mathematical model data entities. Further, by means of original software toolkit (ConceptModeller [14] and content management system [10,12,13]), the model is transformed into a complex semantic network and object-relational warehouses, managed by an abstract machine (and at the CASE level – by a virtual machine). Finally, we arrive to a well-formed layout of software system component interfaces and to an internet portal superstructure. The development levels are detailed in terms of entities, relationships, content definition/manipulation languages, and software tools.
Thus, the SSDL methodology is supported by a family of formal object models for content representation/management. The models incorporate fundamentals and methods of finite sequences, variable domains, semantic networks, and other theories [1-3,5,8].

The approach suggested is the first to provide the following features:

1) rigorous object modeling of heterogeneous software, their elements and, families, as well as the environment for system objects and families;

2) integration of "abstract" formal models and "specific" industry-standard technologies and CASE tools for development software systems (due to innovative "middleware" tools).

Both advantages have been implemented for enterprise content (i.e. integrated data and metadata) representation and management.

Thus, the central aspect of mathematical and conceptual modeling, analysis and design of the software systems is shifting the SSDL paradigm from object-oriented to pure object approach, i.e. from IT to computing. Computing is a relatively new scientific subject, adequately representing complex, heterogeneous, changeable, and interactive problem domains in terms of objects and their environment [9].

4. ORGANIZING THE SOFTWARE DEVELOPMENT LIFECYCLE: SEQUENTIAL ELABORATION

The major purpose of the methodology is the multi-factor optimization of the SSDL model, which is often critical for both product quality and project success. Under such an improved approach to SSDL, the major optimization factors are, terms, costs, quality and maintainability of the product.

Therewith, the specific features of our understanding of the term "optimization" are as follows. First, we do not mean optimization is as a mathematically rigorous as it is commonly meant (we choose the best variant out of a finite number of discrete choices rather than a maximum of a continuous function). Second, the priority of the factors is dependent on project scale and scope. Third, all of the factors mentioned are measurable and have certain numeric metrics (number of code lines, residual faults, etc.). The optimization indicator values can be calculated for each possible software solution scenario on the basis of such prioritized metrics. Reasonable project management decisions could be made on the basis of the indicative values and preliminary estimates from the projects plan.

Naturally, in a number of cases (especially for large-scale, complex, heterogeneous SSDL) it seems reasonable to use the above models for content representation/management at the analysis and conceptual design stages.

Therewith, both formal models (content representation/management) and methodological processes of the SSDL phases (analysis, design, implementation, integration, maintenance etc.) are supported by either standard or specific visual CASE tools. The SSDL processes are supported not only by CASE tools, but also by workflow management tools based on document management system. During each SSDL phase, certain types of documents (e.g., project plan, requirements checklist, unit test report, etc.) are generated and processed depending on the lifecycle model type, and on the project scale/scope.

Since the paper is not aimed at detailed description of versatile and complex interrelated SSDL management processes, let us limit our scope to certain examples and short descriptions of the methodology (i.e. models, metrics, methods, and tools). Detailed description of the methodology is given in [11-13].

During the requirement analysis phase, optimization often results in generating requirements checklist, which is a simplified and less formal document, than the detailed product specification. However, irrespective of the type of the specification document, it should contain the chosen SSDL model as a "global parameter", which is, essentially, the project management outline.

Let us note that the entire product development process (and the project management as well) is a step-by-step elaboration of the software functional description. In the above case, the instantiation is directed from conceptual (formal) model to project specification description with subsequent elaboration of the lifecycle scheme. Further, we move on to the structures of databases and information systems, which make up the software system. Therewith, large-scale, heterogeneous SSDL is primarily iterative, evolutionary, and incremental: every iteration contains further elaboration of the product functions (fig. 1). In essence, the conceptual outline is an improved spiral SSDL model. However, in a number of cases, the general outline may be instantiated, depending on the product scale and scope or on the "project triangle" correction (terms, costs, functions). For example, such a correction may result in SSDL model simplification down to waterfall, where the software development is limited to single pass through all of the lifecycle phases or even to "build-and-fix" model with incomplete lifecycle and simplified product documentation.

Further product elaboration and optimization is performed in terms of system architecture, key technologies, supporting CASE tools and programming languages. Therewith, we should consider the existing clients' software environment. The product developed should be predictable, reliable, maintainable, usable, and, ideally, reusable.

Let us note that lifecycle phases influence project economics differently. Maintenance is the most expensive and challenging phase: it requires over 60% of project time and budget (as coding contribution into the product expenses is minimal). Combining software development methods and tools is essential for low-cost product development.

For a number of cases (e.g., test termination when approaching the satisfactory error threshold) the decision is made solely by the project manager, while for the others (e.g., software retirement) require multi-side project economy evaluation.

The major approaches for creating state-of-the-art (interactive, distributed, open, expandable) products are object-based ones: from classical object-oriented to active objects and "pure" objects. Under such approaches, SSDL phases are flexible, with dynamically adaptive, "floating" borderlines. However, even under state-of-the-art object-based approaches SSDL management is still possible on the basis of strict quantitative SE metrics.

To validate software and to satisfy the product specifications, specific CASE tools should be used (e.g., those which are based on reliability statistical analysis, higher order logics, and other mathematical foundations). Such CASE tools require minimal mathematical training and it is designed for analysts and developers of middle qualification level.

Essential preconditions of project success are functional prioritizing and step-by-step, incremental implementation.

Project specifications should be rigorous, logically sound, non-contradictory, complete in critical functional coverage, they should provide transparent integrity tracing.

Software development requires rigorous procedures for all SSDL phases, which should be strictly followed. Documentation standards should be taken special care of. Otherwise, developing a huge, complex, heterogeneous, distributed product is at risk of becoming an unmanageable, informal anarchy with an unpredictable result. That is why the paper suggests a set of methodologically interrelated processes, which provide development of a predictable, requirement-matching, high quality software, even under such challenges as changeable requirements, budget/terms correction "on the fly", etc.

The suggested methodology is based on a thoroughly selected and practically tested set of models, SE methods and tools.

The methodology has been practically implemented in a number of enterprises, including software development for a large ITERA International Group of Companies (150 companies of over 20 countries, and over 10,000 employees; www.iteragroup.com). The methodology has been also implemented in the Institute of Control Problems

of Russian Academy of Science, Russian Ministry for Industry and Energy, and other enterprise structures.

5. IMPLEMENTATION FEATURES OVERVIEW

Let us overview how the methodology was implemented. First, let us summarize the specific features of the enterprise-scale implementations.

Currently, the multinational enterprises possess large, geographically distributed infrastructures, aimed at the same business goals. Each of the enterprises has accumulated a tremendous and rapidly increasing data burden, comparable to an avalanche. In certain cases, the data bulk exceeds petabyte size, and it tends to double every five years.

Undoubtedly, management of such data is a serious challenge. The problem becomes even more complicated due to heterogeneous nature of the stored data, which varies from well-structured relational databases to non-normalized trees and lists, and to weak-structured multimedia data. The technology presented in the paper is focused at more efficient heterogeneous enterprise and uniform data management procedures.

The technology involves a set of novel mathematical models, methods, and the supporting software engineering tools for object-based representation and manipulation of heterogeneous enterprise data.

6. APPLYING THE METHODOLOGY TO SSDL

Brute force application of the so-called "industrial" enterprise software development methodologies (such as IBM RUP, Microsoft MSF, Oracle CDM etc.) to heterogeneous enterprise data management, without an object-based model-level theoretical basis, results either in unreasonably narrow "mono-vendor" solutions, or in inadequate time-and-cost expenses. On the other hand, the existing generalized approaches to information systems modeling and integration (e.g., category and ontology-based approaches, Cyc and SYNTHESIS projects – [15-17,20] – do not result in practically applicable (scalable, robust, ergonomic) implementations since they are separated from state-of-the-art industrial technologies (CASE, RAD etc.).

Thus, the suggested technology of integrated development and maintenance of heterogeneous internet-based enterprise software systems has been created. The approach is based on rigorous mathematical models and it is supported by software engineering tools, which provide integration to standard enterprise-scale CASE tools, commonly used with software development methodologies. The approach eliminates data duplication and contradiction within the integrated modules, thus increasing the robustness of the enterprise software systems (ESS). The technology integrates a set of ESS development levels: data models, software applications, "industrial" methodologies, CASE, architecture, and DB management. The novel technology elements are:

(i) conceptual framework of ESS development;
(ii) a set of object models for ESS data representation and management;
(iii) CASE tools for semantic-oriented ESS development (ConceptModeller) and intelligent content management (ICMS);
(iv) ESS implementations [11].

For adequate modeling of heterogeneous ESS, a systematic approach has been developed, which includes object models for both data representation and data management [11,21]. The general technological framework of ESS development provides closed-loop, two-way construction with re-engineering.

The general technological framework of ESS development contains stages, which correspond to data representation forms for heterogeneous software system components, communicating in the global environment.

The object nature of the "class-object-value" model framework provides compatibility with traditional object-oriented analysis and design approach (OOAD), as well as with other certain promising approaches (such as D.S.Scott's variable domains [5], V.E.Wolfengagen's conceptual method [8]) and helps to extend the mentioned approaches to model the ESS internet-based environments. The following technological transformation sequence is suggested: (i) a finite sequence object (e.g., a lambda calculus term); (ii) a logical predicate (higher order logic is used); (iii) a frame (as a graphical representation); (iv) a XML object (class definition generated by the ConceptModeller engineering tool); (v) an UML diagram (CASE tool data scheme) in the ESS (meta)data warehouse [11].

Therewith, the warehouse content representation is based on semantic network situation model, which provides intuitive transparency for problem domain analysts when they construct the problem domain description. The model can be ergonomically visualized through a frame-based notation. Warehouse content management is modeled as a state-based abstract machine and role assignments, which naturally generalizes the processes of similar engineering tools, such as (portal page template generation, portal page publication cycle, role/access management etc. Therewith, the major content management operations (declaration, evaluation, personalization etc.) are modeled by the abstract machine language. The language has a formal syntax and denotation semantics in terms of variable domains. The transformation sequence of the model is:

(v) a term of variable domain algebra (D.S. Scott's computations theory is used)[5];
(vi) a domain-based function (higher order logic is used) [5];
(vii) a frame (a graphical notation);
(viii)a XML object (a template for a ICMS portal page);
(ix) HTML code (ICMS portal page code) of the ESS portal.

The architecture of the integrated heterogeneous enterprise content warehouse provides unification due to generalized object association-based relationships at the data at metadata levels. Uniform heterogeneous ESS content management is based on a uniform portal foundation, which serves a meta-level enhancement over the enterprise data warehouse. Assignments act as code scripts; they change ICMS machine states, and provide dynamical, scenario-driven content management.

The ConceptModeller tool assists in semantically-oriented visualized development of heterogeneous ESS data warehouse scheme [21]. A semantic network-based model is suggested, which works in nearly natural-language terms, intuitively transparent to problem domain analysts. Model visualization is based on frame representation of the ESS data scheme.

Deep integration with mathematical models and ESS CASE tools provide a closed-loop, continuous lifecycle with reengineering. The ICMS tool is based on an abstract machine, and it is used for problem-oriented visualized heterogeneous ESS content management and portal publication. ICMS features a flexible scenario-oriented management cycle and role-based mechanisms. ICMS provides a unified portal representation of heterogeneous (meta)data, flexible content processing by various user groups, high security, ergonomics and intuitively transparent complex data object management.

7. PATTERN-BASED DEVELOPMENT FOR SSDL

The general ESS development framework [5] potentially allows the following benefits:

(i) applying a "spiral-like" lifecycle to the general ESS development framework;
(ii) ESS "tuning" by applying a "spiral-like" lifecycle and subsequent verification;
(iii) requirement "tracing";
(iv) building a repository of ESS "meta-snapshots", with which the system and/or warehouse could be "reincarnated" to virtually any previous state using component-wise strategy;
(v) building a "pattern catalogue" [19] for heterogeneous ESS, based on the integrated repository of various ESS state "meta-snapshots";
(vi) developing a repository of "branches" for "cloning" slight ESS variations for the "basis;
(vii) developing a formal language specification (e.g, a DSL technology-based one) [18];

(viii)"adjusting" the existing ESS "meta-snapshot" repository components to match requirements;
(ix) reuse of the desired components.

The preferable ESS development framework tends to be iterative; in certain cases waterfall is an option.

An essential feature of the general ESS development framework is its two-way organization. The approach provides reverse engineering possibility both for ESS in general, and their components in particular. The practical value of the approach is provided by the verifiability of heterogeneous ESS components at the uniform level of the problem domain model, which is practically independent upon the hardware and software environment of the particular component. Therewith, a major theoretical generalization is a possibility of mathematically rigorous verification of the heterogeneous ESS components by a function-based model [5,2]. The ESS engineering models are oriented at a promising "pure" objects approach, which is a strategy of .NET and Java technologies, where any program entity is an object.

An essential benefit of the approach suggested is a possibility of adaptive, sequential "fine tuning" of ESS heterogeneous component management schemes in order to match the rapidly changing business requirements. Such benefit is possible due to the reverse engineering feature of the integrated general iterative framework of ESS development. The reverse engineering is possible down to model level, which allows rigorous component-wise ESS verification. Thus, conventional reengineering and verification can be enhanced by flexible correction and "optimization" of the target ESS in strict accordance with the specified business requirements. This is possible due to the suggested model-level generalization of the iterative, evolutionary ESS development framework.

Another benefit of the suggested ESS development framework is a possibility of building a "catalogue of templates for heterogeneous ESS", which is based on an integrated metadata warehouse, i.e., a "meta-snapshot" repository. Thus, the software development companies get a solution for storing relatively stable or frequently used configurations of heterogeneous enterprise software systems.

The solution potentially allows avoiding the integration problems of "standard" ESS components and/or combinations, which have been obtained previously. The approach allows serious software engineering project savings for clients, provided the ESS developer's "meta-snapshot" repository already stores a similar or an analogous integrated solution to the system required. The above consideration clears the way for "meta-snapshot" repository development, which stores the chronological sequence of ESS solutions as a tree with the "baseline" and slight variations of ESS "branches".

This is similar to version control CASE tools. The approach allows a reasonable selection of most valuable deliverables of the ESS lifecycle phases, and organization of similar solution "cloning". Therewith, the "clones" may be created both for different client enterprises, and for different companies of a single enterprise.

Further discussion could cover the prospective areas of "meta-snapshot" repository development. First of all, to describe the metadata warehouses and the related enterprise-level business requirements it seems reasonable to develop new DSL-type problem-oriented meta-languages. Let us call them the MetaWarehouse Description Language (MWDL) and the Requirement Specification Language (RSL) respectively. Further, the formal models, outlined in the paper and given a more detailed coverage [11], allow interrelation of the RSL and MWDL entities.

Semantic-oriented search mechanisms assist in revealing ESS "meta-snapshot" repository components, which provide the closest matching to the new requirements. The approach potentially allows terms-and-cost-effective and adequate transforming of the existing ESS components in order to match the new requirements with minimum corrections effort and, consequently, with minimum labor expenses.

Therewith, the global perspective it becomes possible to reuse certain ESS components for current or new clients. Selection criteria for such "basic" components may be percentage of reuse, ease of maintenance, client satisfaction, degree of matching business requirements etc.

8. IMPLEMENTATION SUMMARIES

ITERA Oil-and-Gas Group: a Portal-Based Solution

The suggested methodology has been practically approved by development of Internet and Intranet portals in ITERA International Group of Companies. During the design stage, problem domain model specifications are transformed by the innovative ConceptModeller SDK to UML diagrams, then by Oracle Developer/2000 integrated CASE tool – to ER diagrams and, finally, into target IS and enterprise content warehouse storage schemes.

Using the suggested data model, the architectural and interface solution has been customized for enterprise resource management IS with content personalization for a wide spectrum of user and administrator types.

To provide the required industrial scalability and fault tolerance level, the integrated Oracle design and implementation toolkit has been chosen to support UML and business process reengineering.

A set of models have been constructed including problem domain conceptual model for enterprise content dynamics and statics as well as a model for development tools and computational environment in terms of state-based abstract machines, which provide integrated object-based content management in heterogeneous enterprise portals. For the model set, a generalized development toolkit choice criteria set has been suggested for information system prototyping, design and implementation. A set of SDKs has been implemented including ConceptModeller visual problem oriented CASE-tool and the CMS. According to the approach, a generalized interface solution has been designed for Internet-portal, which is based on content-oriented architecture with explicit division into front-end and back-end sides. Portal design scheme is based on a set of data models integrating object-oriented methods of management of data and metadata (or knowledge).The major implementations of portals in ITERA Group were: CMS for network information resources, official Internet site, and enterprise Intranet portal.

Distributed Trading Company: a Domain-Driven Messaging System

A trading corporation used to commercially operate a proprietary Microsoft .NET-based message delivery system for information exchange between the headquarters and the local shops.

The system was client-server based. The client included a local database and a Windows-based messaging service, while the server side consisted of a Web service and central database. The operation/maintenance challenges were: complicated client-side code refactoring (with recompiling and reinstallation); difficult error localization/reduction (due to high coupling, non-flexible and non-transparent architecture); inadequate documentation (due to frequent code updates); and decentralized configuration monitoring/management for remote shops (due to distributed and non-transparent system administration).

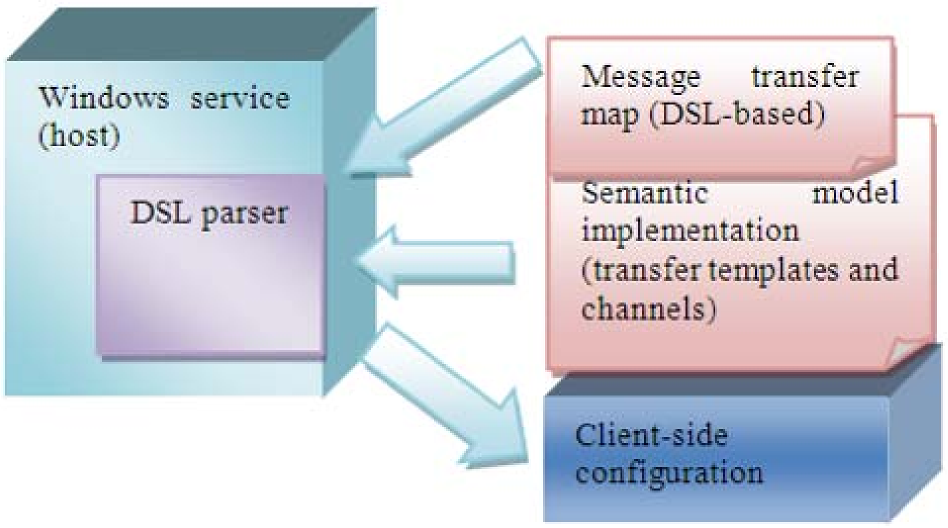

Windows service (host)
DSL parser
Message transfer map (DSL-based)
Semantic model implementation (transfer templates and channels)
Client-side configuration

Figure 2. MES configuration development

To solve the problems mentioned, an approach based on domain-driven development [18] and Domain Specific Languages (DSL) has been suggested. The approach included problem domain modeling and DSL development for managing objects of the problem domain.

The DSL-based model helped to conquer problem domain complexity, to filter and to structure the problem-specific information. It also provided a uniform approach to data representation and manipulation. We used an external XML-based DSL, which extended the scope of the enterprise application programming language. The methodology instance included the following steps: DSL scope detection, problem domain modeling, DSL notation development, DSL restrictions development, and DSL testing.

The approach was client-side focused, since this is the most changeable and challenging task. The lifecycle model is iterative, and it the solution is based on a redesigned architecture pattern (see Figure 2). The Windows service is a constant part of the application (i.e. a host), which contains a DSL parser. The DSL parser input is a current message transfer map.

The DSL scope (i.e. the “flexible” area of the problem domain) included message transfer rules/parameters, and adding new types of messages. Different shops may have different configuration instances, which make the client-side message processing/transfer structure (and which are included into the semantic model).

The next methodology stage was building semantic model of the objects handled by DSL. We got three types of the objects: messages, message transfer channels and message transfer templates. DSL describes object metadata, i.e., configurations and manipulation rules. Templates were core elements of the model, and channels were links between template instances. Templates and channels together make message maps. DSL described the maps, i.e. the static part of the model, while messages referred to its system dynamics and store the state.

Templates define actions with messages, i.e. transform or route them. Templates were grouped into the *IMessageProcessingPattern* interface. Standard routing templates were: content-based router, filter, receiver list, aggregator, splitter, and sorter. We also produced a number of domain-specific templates for system reconfiguration, server interaction, etc.

Channels were used for message management. In the graph of map messaging, templates are represented as nodes, while channels are arcs between certain templates. In our case, two types of channels were implemented: “peer-to-peer” channel and error messages channel.

Based on DSL class model and implementation, messaging maps were built, which were later used by parser to generate system configuration. At this stage, DSL syntax and semantics were built. Each messaging map, generally, a script, was instantiated by a file. Messaging map was built as an XML document, which defined system configuration and contained templates for routing, message processing, transfer channels and their relationships.

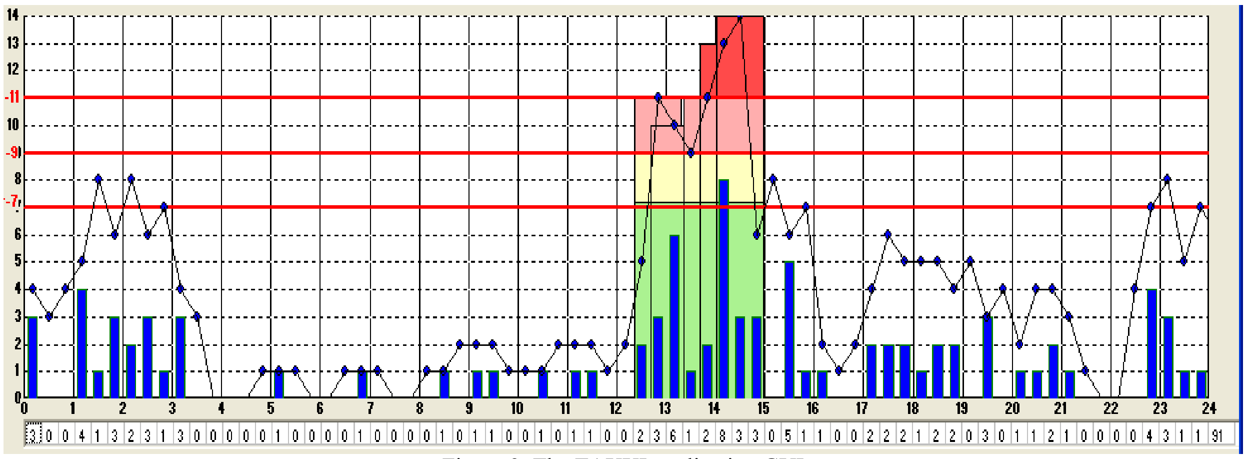
While parsing messaging map, the parser creates channel objects based on DSL channel descriptions. Then it configures the messaging system by creating message processing objects in a similar way. Finally, the parser instantiates the I/O channels, and creates the required relationships between channels and message processor. The resulting DSL-based system configuration was functionally identical to the initial, C#-based one.

Thus, the DSL-based refactoring resulted in an enterprise trade management system with transparent configuration and a standard object-based model (routing templates, channels, etc.). The DSL developed solved the problem of messaging management. Since changes are chiefly localized within the transfer configuration /map, the change management has been dramatically simplified.

The DSL-based methodology instantiation assisted in conquering complexity, made the proprietary system an open, scalable, and maintainable solution. The approach can easily be customized to fit a broad class of similar proprietary systems.

Air Transportation Planning System
Air traffic planning system is an area of work-in-progress.

The problem is to develop remote access to the planning data. An operating solution currently exists. However, it is based on an outdated TAXXI-Baikonur technology, which is no longer evolving after early 2000s. The technology involves component-based visualized assembling of the server application. The ready-made VCL library components from Borland had been integrated with proprietary TAXXI components. The client side is TAXXI Communicator, i.e. an XML browser, which is a "thin" client.



3 0 0 4 1 3 2 3 1 3 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 1 0 1 1 0 0 2 3 6 1 2 8 3 3 0 5 1 1 0 0 2 2 2 1 2 2 0 3 0 1 1 2 1 0 0 0 0 4 3 1 1 91

Figure 3. The TAXXI application GUI

The TAXXI technology is limited Microsoft Windows framework, which is the only possible basis for both client and server-side applications. According to the State Program of Planning System Updates, the Main Air Traffic Management Centre is going to create the new remote access solution. The internet-based architecture is to be implemented in Java technology and to operate on the Apache web server platform. The solution is to query Oracle-based data centre, process the query output and retrieve the results of the air traffic planned capacities to an intuitive and user-friendly GUI.

The practical application of the solution is (building a prototype of) the global enterprise-scale integrated system, which is providing a uniform and equal information access to all of the international air traffic participants (see Figure 3).

The similar globalization processes are underway in Europe and the U.S.A.
The suggested pattern-based and component-wise approach is going to unify the issues of the architecture-level update and application migration in Russia. The methodology will also simplify the integration challenges of the global air traffic management software solution. It is advisable to keep all the given values.

Nuclear Power Plant: Approaching a 6D-Model Based Implementation
Another challenging aspect of the methodology implementation is related to high-level template-based software re-engineering for nuclear power plants (NPP).

To provide worldwide competitive level on the nuclear power plant production, it is necessary to meet the following requirements:

(i) • meeting quality standards throughout the lifecycle;
(ii) • high security under long-term operation;
(iii) • term-and-cost reduction for new generation facilities development.
(iv) The above conditions could be satisfied only under a systematic approach, which combines
(v) • state-of-the-art production potential,
(vi) • advanced control methods, and
(vii) • software engineering tools.

Each stage of the NPP lifecycle (such as: technical proposal, project draft, technical project, design documentation etc.) is mapped into a set of business processes, where not only people, but also enterprise systems (CRM, SCM, ERP, PLM etc.) are interacting.

Identifying operation sequences, the systems form business process automation standards. For example, workflow mechanisms can assist in building enterprise standards on electronic documents validation and approval. During a certain NPP lifecycle, the enterprise systems acquire information on it. Finally, each of the enterprise systems reveals certain NPP aspects: design, technology, economics etc. Thus, various objects (3D-units, technological data, bank accounts etc.), the systems together describe NPP as a huge object. Heterogeneous nature of the data objects, and a huge number of units (measured in million), make NPP a high complexity information object.

A major competitiveness criterion in nuclear power industry is a set of electronic manuals, which helps to assemble, troubleshoot, repair NPP etc. Such manual set provides transparent information models of NPP (units), which allow getting information on the object without directly contacting it.

Such a versatile description, combined in a single data model is often referred to as a 6D model, which includes 3D-geometry, time and resources for operating the plant. Since mechanisms for information searching, scaling, filtering and linking, should provide complete and non-contradictory results, the information models should have well-defined semantics. The uniqueness of data entry assumes information model data acquisition by the enterprise systems throughout the lifecycle.

While a single information model can be derived out of a single system (and a 3D model – out of a CAD system), the 6D model should combine information models of a number of systems. The methodology for building a 6D model suggests portal-based system integration, which can be based on a “platform” capable of entire lifecycle support (such as Siemens Teamcenter or DS V6).

The further information model development assumes monitoring system state changes and their influence to the other parts of the system. This helps to immediately react on critical issues in NPP construction (terms growth due to late unit delivery etc.), which can be used for decision making. (A wrong decision would be made otherwise under incomplete or incorrect information).

Among major nuclear industry challenges, there is a concept of a typical optimized nuclear reactor. The idea is in selecting typical invariant units for rapid “template-based” development of a set of slightly varying versions (meeting local conditions etc.). Applying the suggested methodology to the 6D information model of the nuclear reactor, is a promising approach to pattern-based component-wise development of NPP series.

9. CONCLUSIONS

SSDL management is a challenge in case of large-scale distributed heterogeneous applications. To solve the challenge, a uniform SSDL management methodology is suggested, which includes models, methods and supporting CASE-level tools.

The methodology implementations in a number of large-scale governmental and commercial enterprises have proved essential project terms-and-costs reduction, and industrial quality level of the heterogeneous applications.

Implementation of the methodology allowed to developing a unified ESS, which integrates a number of heterogeneous components: state-of-the-art Oracle-based ERP modules for financial planning and management, a legacy HR management system and a weak-structured multimedia archive. The implementation of internet and intranet portals, which manage the heterogeneous ESS warehouse content, provided a number of successful implementations in diversified ITERA International Group of companies (approximately 10,000 employees in over 20 countries). The systematic approach to ESS framework development provides integration with a wide range of state-of-the-art CASE tools and standards of ESS development.

Other implementations and work-in-progress areas include: air transportation planning system, messaging system for a trading enterprise, a nuclear power plant and banking solutions. Each of the implementations is a domain-specific one, so the system cloning process is not straightforward, and it requires certain analytical and CASE re-engineering efforts.

However, in most cases the approach reveals patterns for building similar implementation in series, which results in substantial term-and-cost reduction of 30% and more. The series can be applied both to subsidiaries and to different enterprises.

The author is going to continue his studies of enterprise software systems, their lifecycle optimization and pattern-based development.

10. REFERENCES

[1] Barendregt H.P., **The lambda calculus (revised edition),** Studies in Logic, 103, North Holland, Amsterdam, 1984

[2] Curry H.B., Feys R., **Combinatory logic**, Vol.1, North Holland, Amsterdam, 1958

[3] Roussopulos N.D., **A semantic network model of databases**. Toronto Univ., 1976

[4] Schach S.R., **Object-Oriented and Classical Software Engineering** (5 ed.) McGraw-Hill, 2001, 744 pp.

[5] Scott D.S., **Lectures on a mathematical theory of computations**. Oxford University Computing Laboratory Technical Monograph. PRG-19, 1981, 148 pp.

[6] Sommerville I,. ***Software Engineering*** (8 ed.), Addison-Wesley, 2006, 864 pp.

[7] Tomaiko J.E., Twenty-year Retrospective: The Nato Software Engineering Conferences. **The 11th International Conference on Software Engineering**, May 15-18, 1989, Vol.I, p.96

[8] Wolfengagen V.E., Event Driven Objects. **Proc. CSIT'99.** Moscow, Russia, 1999, p.p.88-96

[9] Wolfengagen V.E., **Applicative Computing. Its quarks, atoms and molecules**. Moscow: JurInfoR, 2010. 62 pp.

[10] Zykov S.V., Enterprise Content Management: Bridging the Academia and Industry Gap **Proc. i-Society 2007**, Merrillville, Indiana, USA, Oct. 7-11, 2007, Vol.I, p.p.145-152

[11] Zykov S.V., Integrated Methodology for Internet-Based Enterprise Information Systems Development, **Proc. WEBIST2005**, Miami, FL, USA, May 2005, p.p.168-175

[12] Zykov S.V., An Integral Approach to Enterprise Content Management, **Callaos N., Lesso W., Zinn C.D., Zmazek B. (Eds.), Proc. of 11th International World Multi-Conference on Systemics, Cybernetics and**

Informatics (WMSCI 2007), Orlando, FL, U.S.A., July 8-11, 2007, Vol. I, p.p. 212-216

[13] Zykov S.V., The Integrated Methodology for Enterprise Content Management, **WMSCI 2009**, July 10-13, 2009 – Orlando, FL, USA , p.p.259-264

[14] Zykov S.V., ConceptModeller: A Frame-Based Toolkit for Modeling Complex Software Applications**. J.Baralt, N.Callaos, H.-W.Chu, M.J.Savoie, and C.D.Zinn (Eds.): Proceedings of the International Multi-Conferences on Complexity, Informatics and Cybernetics (IMCIC 2010)**, Orlando, FL, U.S.A., April 6-9, 2010, Vol.I, pp.468-473

[15] Guha R., Lenat D., **Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project**. Addison-Wesley, 1990

[16] Lenat D., Reed S., Mapping Ontologies into Cyc, **AAAI 2002 Conference Workshop on Ontologies for the Semantic Web**, Edmonton, Canada, 2002

[17] Birnbaum L., Forbus K., et al., Combining analogy, intelligent information retrieval, and knowledge integration for analysis: A preliminary report. **In: ICIA 2005**, McLean, USA, 2005

[18] Evans E., **Domain-Driven Design: Tackling Complexity in the Heart of Software**. Addison Wesley, 2003, 560 pp.

[19] Fowler M., **Analysis Patterns: Reusable Object Models**, Addison Wesley, 1997, 223 pp.

[20] Kalinichenko L., Stupnikov S., Heterogeneous information model unification as a pre-requisite to resource schema mapping. In: **ITAIS 2009**, Springer, 2009, pp. 373-380

[21] Zykov S., Pattern Development Technology for Heterogeneous Enterprise Software Systems. **Journal of Communication and Computer**, 7 (4), 2010, pp.56-61