Architecture knowledge for software generation

Ivan Stanev, Maria Koleva

***Abstract*—**The key software engineering problems nowadays are: lack of software developers, insufficient quality of software products, and high software cost. Even widespread and well defined software development methods, such as Rational Unified Process and Agile, could not solve these problems. A method improving the software development process through architecture-based and knowledge-based automated software engineering is presented. The main business objects and control objects of the method are described. The automation process ontology including business, software and infrastructure architecture objects is proposed. A tool for specification of domain area facts and knowledge, problems and problem solving algorithms is created. A technological process for software development is proposed based on the described method. The method actions include: (1) manual specification of the business objects and control objects; (2) automated verification of the specified objects; (3) automatic interpretation of the compiled control objects that reuse pre-defined repository components; (4) automated self-monitoring and self-control of the runtime process. The proposed method efficiency is evaluated based on some quality and quantity attributes.

***Keywords*—**knowledge based automated software engineering ontology; architecture-based; automated software engineering; business architecture; software architecture; infrastructure architecture; knowledge-based system; specification verification; reusable components; self-control; self-monitoring;

I. Introduction

THE huge demand for software (for e-government, e-commerce, e-business, mobile devices, embedded systems, IoT, etc.) brings the most serious crisis ever in software engineering. Issues such as lack of developers, insufficient quality of software products, rising software cost, and low customer satisfaction become increasingly common.

The IT standardization process is slowed down by the rapid development of industries that are becoming dependent on technology. As a result, developed software is substandard and high-priced.

Software vendors replace the software products deployed on own hardware, with those offered on common platforms, and cloud technologies to reduce development, infrastructure, and quality costs.

The overall IT spending in 2017 for data center systems, devices, enterprise software, and IT services is 2,1 trillion USD [9]. The enterprise software and IT services markets form 60% of the overall IT spending worldwide. Together, they are forecasted with the highest average annual growth for the 2016-2021 period, namely 13% [9].

The demand for software developers is increasing exponentially. According to the U.S. Bureau of Labor Statistics, software developer jobs are expected to grow with 24% from 2016 till 2026, which is „much faster“ than the average [24].

The existing widely used software development methods do not offer sufficiently efficient technological processes, as well as standardized and efficient development tools.

A possible solution to these problems could be the Knowledge Based Automated Software Engineering (KBASE) method presented in this paper. KBASE is a modified version of the RUP, enriched with elements from Agile, cloud computing, knowledge processing, and automated programming.

II. Related Works

This article is part of a series of works related to the elaboration of the KBASE method. In this section are summarized some important results concerning the software engineering technological processes, specification techniques, architectures and platforms. They are organized in three parts - topic review, review results and requirements to the KBASE method based on the review results. KBASE case studies are also presented.

A. Technological processes

Several methods for management of the software engineering (SE) process are studied. The selected methods can be organized in three groups:

(1) methods covering only the management aspects of the SE process, called “shell” methods such as: PRINCE2 (Projects IN Controlled Environments, [1]); TEMPO (TAXUD Electronic Management of Projects Online); PMBOK (Project Management Body of Knowledge, [21]);

(2) methods covering partially the management and technological aspects of rapid SE processes (the Agile family of methods, [17]);

(3) methods covering to a great extent the management and technological aspects of SE processes such as the Rational Unified Process (RUP, [18]).

The “Shell” group of methods does not describe in details the roles, objects and actions in the SE process. This is often the reason for serious misunderstanding between clients and developers, as well as between different developers. The low level of standardization makes these methods inappropriate for automated programming.

The presented work has been partially funded by the National Scientific Research Fund, Contract No. 02/13/12.12.2014.

I.Stanev is with the Computer Informatics Department of University of Sofia “St. Kliment Ohridski”, Sofia 1000, Bulgaria (phone: +359 2 8161 508, e-mail: instanev@fmi.uni-sofia.bg).

M.Koleva is with the Computer Informatics Department of University of Sofia “St. Kliment Ohridski”, Sofia 1000, Bulgaria, and the Informatics and Information Technologies Department of University of Ruse “Angel Kanchev”, Ruse 7000, Bulgaria (e-mail: mkoleva@fmi.uni-sofia.bg).

The Agile group of methods has two important problems if used alone: (1) the short and often incomplete design reduces to minimum the possibilities for reuse; (2) as a results of the rapid and imprecise development process, the lifetime of the realized products is too short.

The RUP group of methods is suitable for a base of the KBASE method. However RUP hinder a smooth development process due to the following problems: (1) roles, objects, and actions are defined in detail, thus making the software process too heavy and difficult to customize; (2) the development team training time is too long and expensive; (3) the complexity of the process makes it difficult to be followed by the client.

To overcome these problems in the KBASE technological process, RUP shall be simplified and extended with Agile.

B. Specification languages

The industrial languages (both for modelling and implementation) including Java, ML, Prolog, UML, BPMN, and Net are analyzed in [8].

The analysis shows that: (1) imperative languages, such as Java and C#, are of major importance for the software industry; (2) the forth-generation languages are widely used for modelling; (3) the closest object between the modeling and programming languages is the empty code, which is not convenient to work, nor industrially significant; (4) Although slower than desired, restricted natural language used for modeling gains industrial importance; (5) Tools for knowledge specification are not sufficiently developed; (6) great industrial importance acquire specification standards such as UML, BPMN, etc.

As a result of the analysis a set of requirements for the KBASE specification language is prepared. KBASE specification language(s) should enable the: (1) descriptions in restricted natural language that would facilitate a wide range of end users, who are not IT specialists; (2) formal graphical-textual specifications of the DA model; (3) interpretation of fuzzy terms, to reduce the time for describing the problem; (4) heuristic, automated search of solutions; (5) description of control components in a convenient way and control of the correctness of descriptions; (6) combination, whenever necessary, of more than one specification technique.

C. Platforms

The industrial cloud computing ([3], [19]) platforms Amazon AWS, Microsoft Azure, Google App Engine; VMWare vCloud, IBM Bluemix, HP Helion, and Oracle OCPaaS are analyzed in [6].

The analysis shows that: (1) all platforms have excellent tools for programming and automated programming for hardware, presentation and integration. (2) All platforms have good tools for the development of SOA applications. (3) Automation tools are limited for user tasks execution management at service level, development and runtime management, document and organization management. (4) There is considerable shortage of automation tools for generation of software products requested by the user based on specifications prepared using graphical interface language or natural language. (5) There is considerable shortage of ontology and knowledge processing tools.

As a result of the analysis a set of requirements for the KBASE platform is prepared. The platform should:

(1) combine techniques for automated programming from SOA, cloud computing, and the Method for Automated Programming of Robots ([4]);

(2) provide for incomplete and imprecise specification of the problem to be solved using language and tools familiar to the end user with no IT qualification;

(3) enable knowledge acquisition and knowledge interpretation (e.g. knowledge based systems, ontologies and fuzzy sets) for automated removal of deficiencies and inaccuracies in the specification and for software generation;

(4) ensure automated software generation from complete and accurate specification;

(5) allow the use of automated techniques such as Contract Testing and Quality of Service Testing to fine tune the software, to check on software performance and integration with third party components.

D. Case studies

The KBASE method is developed as result of the domain area problems analysis, based on 85 software products developed by the KBASE team. Most of the KBASE realization techniques used to automate or improve the programming process are partially or fully implemented and verified in one or more of these developments.

Architecture prototypes in the KBASE context are developed during the realization of three large state administration programs – Bulgarian e-Customs, Bulgarian e-Health, and Bulgarian e-Government. The results are summarized in [7].

The technological development of BeC, BeH and BeG spans over a long period of time. Each of the three programs at present has more than 15 business modules, 5000 workstations in intranets, and between 500 000 and 3 000 000 end users in internet. They are an integral part of the trans-European solutions developed and operated by the EU Commission, EU member states and partner countries.

During the realization of these programs two types of **problems** are recognized:

(1) ***organizational***, such as: lack of single access point with a centralized authorization solution, inefficient infrastructure use, low level of standardization of processes and objects, insufficient IT staff to develop new systems and support existing ones, low competence of business staff for business processes description and functional testing, lack of guidelines for collection and interpretation of information, huge resources are allocated to the development of inefficient business models, low competence for requirements gathering and elicitation, chaos in requirements definition, low level of business specific activities quality; and

(2) ***technical***, such as: big volume of not digitized data, low quality of digitized data, low level of semantic interoperability, implementation of reusable components to a minimum, resulting in frequent rewrite of existing functionality, lack of scalability, low level of software development automation, lack of flexibility and instruments for fast information systems adaptation to rapidly changing legal base, lack of centralized identification, relatively low quality of the developed software products, lack of standardization of processes and objects, low test coverage.

The problems identified during the realization of these programs are addressed in a number of research projects related to knowledge based automated software engineering in the area of information systems, programs for robot control [4], contract and quality of service testing [20]. The important results achieved by the KBASE team are summarized in [5]. Some of them are: (1) prototyped 3 specification techniques (formal language, graphical user interface, natural language processing); (2) prototyped 5 AI techniques (non-formal specification, code generation, self-verification, self-monitoring, self-tuning); (3) prototyped 4 KBASE components (knowledge processor, product generator, problem solver, knowledge base manager); (4) prototyped 5 technological processes (SOA interpretation process, knowledge engineering process, domain customization process, system customization process, runtime operation process).

E. The way forward

As a possible solution to the problems presented in section I the Knowledge Based Automated Software Engineering method is proposed. KBASE is a modified version of the RUP, enriched with: (1) elements from Agile - rapid application development and reusable components; (2) cloud computing – use platforms instead of products; (3) , knowledge processing - natural language processing, knowledge based systems, expert systems, ontologies; and (4) automated programming - informal specifications, software design standards.

III. KBASE Method Concept

KBASE aims to improve the quality of the developed software products, while significantly reducing the effort to develop and maintain them and thereby ensure that the development time, the size of the development and maintenance teams, and the cost of the final product are reduced.

The method achieves this goal through:

(1) ***standardization*** (standardized technological process, specification techniques, architectural patterns, and pre-defined models);

(2***) automated specification*** (manual specification of components, services, objects, problems to be solved, and problem-solving algorithms; automated verification of specification completeness and consistency based on pre-defined models; and determination of incomplete and imprecise specifications);

(3) ***automatic code generation*** (by introducing service oriented architectures and reusable components);

(4) ***infrastructure automated control*** (by integrating different product infrastructures in a common platform, by establishing automatic cloud-based platform scalability, automated virtualization, and automated quality of service);

(5) ***automated self-monitoring, self-learning and self-control*** (achieved through contract testing, adaptation to the end user, adaptation to the context of the problem solved, automated versioning, etc.).

The method concept is shown in Fig. 1. KBASE is a method that is suitable for building the following types of **products**: (1) **Information Systems** (including Enterprise information systems, EIS), (2) **Knowledge Based Systems** (KBS), (3) **Embedded Systems**, and (4) **System Services**

The objects used in the development of the method are divided into three broad categories:

(1) ***control components*** including navigation trees, business processes, state machines, multi-agent systems and components with mixed control (incl. the preceding ones);

(2) ***tasks*** that represent models of all important atomic data operations, including **business tasks** representing atomic operations derived from business architectures such as TOGAF ([23]), **system tasks** representing atomic operations of the systemic software, **knowledge tasks** - atomic operations for knowledge processing and **data tasks** - atomic operations for data processing;

(3) ***objects*** that represent data structure models, including **business objects** - typical for the domain area, **system objects** - typical for the system software, and **knowledge objects** - to describe the knowledge.

KBASE objects (models and their instances) are stored in three ***repositories*** – **control component repository**, **task repository**, and **object repository**.

KBASE basic roles are the following:

(1) ***business analyst***, who gathers information from the clients about the system to be developed, and based on that information, controls the process of knowledge generation out of incomplete and imprecise specifications of the software product;

(2) ***software architect***, who assists the business analyst by consulting them on the system generation, or prepares (in advance or on demand) control components models;

(3) ***designer*** who helps the business analyst by consulting them on system generation, or prepares (in advance or on demand) task models or new tasks of the following types: **services,** which are automatically executed, **user tasks,** which are suitable for organizing the human-machine interaction, and **business rules**, which support the navigation through the control flow, or are used for system services activities;

(4) ***data engineer*** who helps the business analyst by consulting them on system generation, or prepares (in advance or on demand) information object models;

(5) ***end user*** who uses the generated product for solving their problems based on input data.

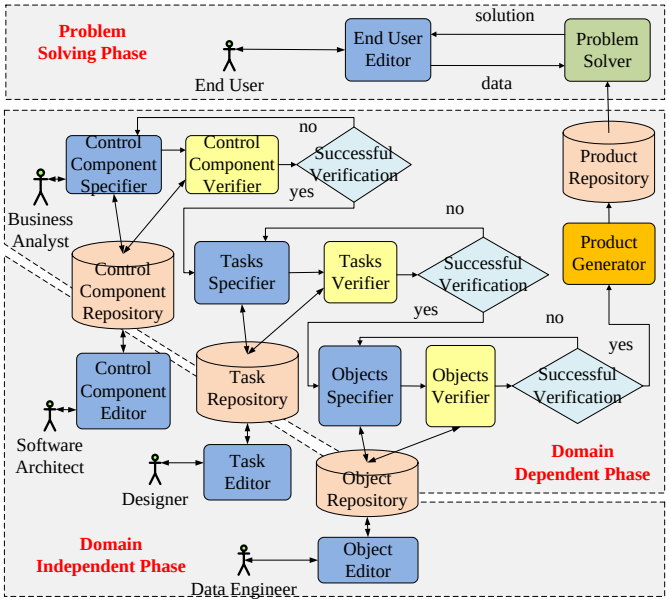

Fig. 1. KBASE method concept

The software development process is divided into three main phases:

(1) ***Domain Independent Phase*** – the main roles involved are the software architect, the designer and the data engineer who prepare those control components, tasks, information objects and their models which are suitable for various domain areas. These objects are usually developed once; they are re-usable and are subject to minor changes only in exceptional cases;

(2) ***Domain Dependent Phase*** – the main role involved is the business analyst who has to generate a new product on demand by the end users. He is supported by the software architect, the designer and the data engineer. Together with other domain area experts, they expand the domain independent models with re-usable ones convenient for the specific domain;

(3) ***Problem solving phase*** – the main role involved is the end user who uses the generated product for solving a specific user problem based on the input data he provides.

The KBASE method is provided with specification instruments for three types of information: (1) ***specification of domain area knowledge and facts*** – activity performed by all the roles involved in the process; (2) ***specification of problem solving algorithms*** – activity performed by the business analyst supported by all other roles involved in the process; (3) ***specification of the problems to be solved*** – activity performed mainly by the end user who may rarely address other IT experts or domain area experts.

All three types of specifications can be prepared by using: (1) ***formal specification standards*** such as UML, BPMN, etc.; (2) ***restricted natural language***. Specifications can be incomplete and imprecise. Based on the available domain knowledge and, if necessary, assisted by the domain area experts, KBASE instruments shall transform the incomplete and imprecise specifications into deterministic algorithms suitable for execution by a Von Neumann machine. These tasks, which cannot be specified sufficiently well (in rare cases) by the instruments stated herein, might be written in a concrete programming language (e.g. Java, C#, etc.).

In Fig. 1 are shown two business processes for software generation using incomplete and imprecise specifications: (1) ***Product Generation*** - providing specification and verification of product control components, product tasks, and product objects, as well as the product generation. The generated product is stored in the product repository after its successful completion; (2) ***Problem solving*** – provision of the input data, it's processing by the generated software product and the display of output results.

The **Product Generation** process is controlled by the business analyst who follows the instructions of the selected model, as well as the instructions of the product users. The business analyst enters the information gathered about the control components, tasks and objects. The gathered information is verified by the system against pre-set identical or similar models. The models contain data about the minimum information required for the purpose of product generation. The process is iterative. Each step could iterate until a satisfactory completeness and accuracy of the generated product is reached.

The **Problem solving** process is controlled by the end user. It can be also performed through several iterations (usually when the input data is incomplete). Rarely, the system needs knowledge provided by roles different from the end user.

On product generation, relevant components and tasks for self-monitoring, self-learning and self-control are embedded automatically in it. The purpose is to improve the quality of the software development process with regard to: improvement of the efficiency of resources, adaptation to different end-users and domain areas, etc.

IV. KBASE Ontology

KBASE is built on the: (1) use of standardized specification techniques; (2) work with architectural patterns; (3) verification and supplementing incomplete and imprecise specifications; (4) realization of Contract Tests; (5) development of models of KBASE objects; (6) use of self-monitoring, self-learning and self-organizing techniques. The successful implementation of these techniques requires a high degree of standardization in the domain area, as well as well interconnected and complementary knowledge of the domain area.

The required knowledge is summarized in the KBASE ontology in TABLE I. The ontology covers four main categories of knowledge: (1) technological process; (2) specification languages; (3) architectures; (4) platform.

TABLE I. KBASE ONTOLOGY

1 0 0	**technological process**
1 1 0	**technological roles**
1 1 1	**knowledge engineers**
1 1 2	analysts
1 1 3	developers
1 1 4	testers
1 1 5	managers
1 1 6	production & support
1 1 7	general roles
1 2 0	**artefacts**
1 2 1	project management plan
1 2 2	quality management plan
1 2 3	**ontology model**
1 2 4	requirements model
1 2 5	infrastructure model
1 2 6	platform
1 2 7	business model
1 2 8	data model
1 2 9	use case model
1 2 10	test model
1 2 11	software architecture
1 2 12	design model
1 2 13	implementation model
1 2 14	product
1 2 15	quality management results
1 2 16	user materials
1 2 17	project data
1 2 18	assessment
1 3 0	**disciplines**
1 3 1	management
1 3 2	requirements
1 3 3	infrastructure
1 3 4	analysis
1 3 5	test
1 3 6	design
1 3 7	generation & implementation
1 3 8	exploitation

2 0 0	**specification languages**
2 1 0	**specification types**
2 1 1	**NL specification**
2 1 2	formal language specification
2 2 0	**specification techniques**
2 2 1	**NL based**
2 2 2	**context based**
2 2 3	event based
2 2 4	process based
2 2 5	message based
2 2 6	service based
2 2 7	object based
2 2 8	**rule based**
2 2 9	**ontology based**
2 3 0	**specification standards**
2 3 1	**NL Combinatorial Dictionar**
2 3 2	**NET**
2 3 3	**CD**
2 3 4	**CMMN**
2 3 5	BPMN
2 3 6	UML
2 3 7	**DMN**
2 3 8	**OWL**
2 4 0	**specification processes**
2 4 1	specification processes
2 4 2	verification processes
2 4 3	**generation processes**
2 5 0	**specification phases**
2 5 1	**domain independent**
2 5 2	**domain dependent**
2 5 3	**problem oriented**

3 0 0	**architectures**
3 1 0	**structure types**
3 1 1	class
3 1 2	component
3 1 3	package
3 1 4	product
3 1 5	**knowledge model**
3 2 0	**product types**
3 2 1	information system
3 2 2	**knowledge based system**
3 2 3	embedded system
3 2 4	system services
3 3 0	**control components**
3 3 1	business process
3 3 2	**state engine**
3 3 3	**multi-agent**
3 3 4	navigation tree
3 3 5	mixed
3 4 0	**action types**
3 4 1	task
3 4 2	rule
3 4 3	event
3 4 4	policy
3 4 5	**behavior**
3 5 0	**object types**
3 5 1	attribute
3 5 2	data object
3 5 3	db object
3 5 4	message
3 6 0	**constructors**
3 6 1	structure
3 6 2	action
3 6 3	object
3 6 4	**behavior**
3 7 0	**descriptors**
3 7 1	structure
3 7 2	action
3 7 3	object
3 7 4	**knowledge model**
3 7 5	**problem**
3 7 6	**problem solving algorithm**
3 8 0	**business objects**
3 8 1	business actor
3 8 2	business role
3 8 3	business process
3 8 4	business interface
3 8 5	business function
3 8 6	business interaction
3 8 7	contract
3 8 8	business service
3 8 9	organization
3 8 10	domains
3 9 0	**application objects**
3 9 1	application component
3 9 2	application interface
3 9 3	application service
3 9 4	data object

4 0 0	**platform**
4 1 0	**platform types**
4 1 1	development
4 1 2	test
4 1 3	pre-production
4 1 4	production
4 1 5	management
4 2 0	**platform object types**
4 2 1	infrastructure service
4 2 2	network
4 2 3	node
4 3 0	**infrastructure layer**
4 3 1	hardware
4 3 2	real OS
4 3 3	virtualization
4 4 0	**cloud layer**
4 4 1	virtual OS
4 4 2	cloud cartridge instances
4 4 3	cloud cartridges
4 5 0	**control layer**
4 5 1	application servers
4 5 2	control servers
4 6 0	**ontology layer**
4 6 1	operational data
4 6 2	analytical data
4 6 3	reference data
4 6 4	**knowledge**
4 7 0	**business layer**
4 7 1	organization management
4 7 2	document management
4 7 3	activity management
4 7 4	collaboration management
4 7 5	user support
4 8 0	**development layer**
4 9 0	**system services layer**
4 9 1	runtime management
4 9 2	presentation management
4 10 0	**integration layer**
4 10 1	integration definitions
4 10 2	**TIS integrators**
4 11 0	**service bus**
4 11 1	enterprise service bus
4 11 2	**semantic enterprise service bus**

Each of these categories includes knowledge defining KBASE from a different perspective and grouped into subcategories. The black-colored components of the subcategories belong to the Software Engineering Ontology. The red-colored components of the subcategories upgrade the Software Engineering ontology to the KBASE ontology.

The ontology is designed to be used by all roles presented above in section III. During the preparation other works in the area have been taken into consideration, such as the Open Group Architecture Framework [23] and SOA ontology [22], the European Interoperability Reference Architecture [2].

The most important subcategories, their interpretation, in which parts of KBASE are intended to be used, and other useful information is presented below.

The ***Technological Process*** category includes 3 subcategories intended to standardize the KBASE development process. The subcategory **Technological roles** (including mainly the groups of roles prescribed in the RUP) outlines the main groups of participants in the KBASE development process. The Knowledge Engineer role is added to cover activities related to automated programming and knowledge processing. The **Artefacts** subcategory (including the most important and most commonly used RUP artefacts) includes the minimum full set of artefacts required for quality development and maintenance of KBASE products. The **Disciplines** subcategory (including RUP disciplines with small modifications) presents the actions performed throughout the entire lifecycle of the KBASE product. The discipline of Generation & implementation covers the process of automated generation of KBASE products from incomplete and imprecise specifications. It is expected that, based on the results of this discipline, after the first 2-3 KBASE products, a software company will have to write new code to a minimum (10-20%) since the required code will mostly be generated automatically.

The ***Specification Languages*** category includes 5 subcategories for KBASE specification preparation and processing. The **Specification Types** subcategory includes both formal and natural language specifications. The **Specification Techniques** subcategory shows which specification techniques can be used to specify the different objects in the different phases of the process. These techniques are linked in Section V with the steps of the process, the artefacts prepared and the actions taken. The specification techniques are implemented using wholly or partly some of the most common specification standards presented in the **Specification Standards** subcategory. This subcategory includes among others the Net and OWL languages. Although less widespread they have shown in practice excellent results related to automated processing of incomplete and imprecise specifications and application of SOA techniques. Subcategory **Specification Processes** includes specification processing including syntactic and semantic, as well as interoperability aspects. Subcategory **Specification Phases** introduces an automated programming technique based on reuse of components and tasks from KBASE repositories.

The ***Architectures*** category includes 9 subcategories with KBASE concepts describing ***Knowledge*** *Architecture*, ***Business*** *Architecture*, and ***Software*** *Architecture.*

The ***Structure Types*** subcategory includes all primitive and complex architectural components and models. Subcategory ***Product types*** includes the types of systems defined in KBASE. The next 5 subcategories contain the basic KBASE architecture components, which are: **control components** (including all control concepts), **actions** (including all action type concepts), **objects** (including all data concepts), **constructors** (including the rules for description of the concepts structure), and **descriptors** (including the rules for description of the concepts content). The Architecture category also includes ***Business Objects*** used to describe the business architecture and ***Application objects*** to describe the application architecture.

The ***Platform*** category includes 11 subcategories with the KBASE Platform concepts. The **Platform Types** subcategory shows which types of platforms are used in KBASE. These are: (1) development; (2) test; (3) pre-production; (4) production; (5) management. The other subcategories define the structure of the platform and are described in more detail in the next section.

V. KBASE Technological Process

The automated programming process modifying RUP is presented in TABLE II. below. The disciplines names are given in the "Disc." column. The work performed within each discipline is described in the "Discipline action" column. The specification technique/s used for the relevant action is indicated in the last column "Sp.technique" and it is further described in section VI. The resulting artefact is listed in the "Art." column.

It has to be noted that: (1) the pre-project is performed by implementing the disciplines and actions to a limited extent; (2) actions could iterate in order to achieve better results; (3) the "Management" discipline represents horizontal activities related to planning, specification and realization; (4) the "infrastructure diagram" term is used instead of the UML standard "deployment diagram" to better convey the purpose with no changes in the notation; (5) the "Infrastructure model" term is used instead of the standard RUP "Deployment model" to incorporate the development, testing, pre-production, production and management environments; (6) "Task" is used for tasks of three different control components, namely business process, state engine and multi-agent systems.

TABLE II. KBASE lifecycle disciplines

Disc.	Art.	Discipline action	Sp.technique
Mng	PMP	project management and realization	SPEM
Mng	PMP	configuration and change management and realization	SPEM
Mng	QMP	quality management and	SPEM

Disc.	Art.	Discipline action	Sp.technique
		realization	
Req	RM	gather and analyze relevant legal base	context based, NL based, ontology based
Req	RM	build organization hierarchy	object based
Req	RM	define functional areas	context based
Req	RM	build roles hierarchy for each functional area	object based
Req	RM	develop software architecture concept	event based
Req	RM	gather functional requirements	context based, NL based, ontology based
Req	RM	gather non-functional requirements	context based, NL based, ontology based
Req	RM	categorize, classify and prioritize functional requirements	context based
Req	RM	categorize, classify and prioritize non-functional requirements	context based
Inf	InfM	infrastructure management and realization	event based
A	BM	selection of control component (business process / state engine / multi-agent system)	process based / object based / event based
A	BM	control component description	process based / object based / event based
A	BM	high level description of control components tasks (task type, input, output and system objects, interfaces, interactions, messages, reports, security, actors, preconditions, post conditions)	service based, rule based
A	DatM	identify primary data objects - documents, forms, reports, messages	object based, rule based
A	DatM	develop data objects hierarchy	object based
A	BM	select candidate tasks	event based
A	BM	select tasks	event based
A	BM	refine control component with candidate tasks to control component with tasks	event based
A	UCM	define use cases for each task	service based
A	UCM	define input and output data objects for each task and select data objects storage options	object based, rule based
A	UCM	define input and output messages for each task and select messages storage options	object based
T	TM	test management and realization	BIT
D	SA	develop software architecture component diagrams	event based
D	SA	define system architecture components	event based
D	SA	define interfaces of system architecture components	interface based
D	SA	develop GUI navigation tree	object based
D	DesM	prepare class diagrams of the tasks	service based
D	DesM	define attributes and operations of classes	service based
D	DesM	develop sequence diagrams of the tasks	interface based
D	DatM	refine object data model into relational data model	object based
D	DatM	relational data model normalization (if necessary)	object based
D	DesM	develop statechart diagrams of the tasks	interface based
D	DesM	develop statechart diagrams of important data objects	interface based
D	DatM	update Data model	object based, rule based
D	ImpM	prepare package diagrams	event based
D	OM	associate use case, component, class, package and infrastructure diagrams	ontology based
G&I	ImpM	code generation	all
G&I	ImpM	code implementation	all
G&I	ImpM	code integration	all
Exp	all	deployment	all
Exp	all	exploitation	all
Exp	all	enhancement	all
Exp	all	recycling	all

Abbreviations: **A** – Analysis discipline, **BIT** - Built-in-Test ([20]) , **BM** – Business model, **D** – Design discipline, **DatM** – Data model, **DesM** – Design model, **Exp** – Exploitation discipline, **G&I** – Generation and Implementation discipline, **ImpM** – Implementation model, **Inf** – Infrastructure discipline, **InfM** – Infrastructure model, **Mng** – Management discipline, **OM** - Ontology model, **QMP** – Quality Management Plan; **PMP** – Project Management Plan, **Req** – Requirements discipline, **RM** – Requirements Model, **SA** – Software Architecture, **SPEM** - Software & Systems Process Engineering Metamodel Specification ([11]); **T** – Test discipline, **TM** – Test model, **UCM** – Use case model.

For the purpose of the KBASE method presented in this paper, the following simplifications/ extensions to RUP are proposed: (1) the RUP discipline Business Modeling is removed since low productive, and highly resource consuming; (2) the RUP discipline Analysis and Design is separated in two KBASE disciplines – first Analysis, and second Design, in order to reduce the gap between domain area experts and analysts on the one hand and between analysts and designers on the other hand; (3) the RUP discipline Implementation is replaced by Generation & Implementation; (4) the RUP horizontal disciplines Configuration and Change management, Project management, and Environment discipline process aspects are combined in one Management discipline to optimize project realization; (5) the KBASE Requirements discipline uses the Contextual design models which improve the quality of primary information collection; (6) the KBASE Analysis discipline uses the process-based, object-based and event-based specification techniques instead of the RUP Use Case Specification in order to reduce the resources for UML model completion; (7) the KBASE Design discipline uses ontologies for automation of the development process; (8) the KBASE Generation & Implementation discipline uses automated

support of models and documents; (9) the Test discipline action uses structured Use Case specifications for better compliance between use cases and test cases, improvement of the quality and efficiency of the test process; (10) the Test discipline action uses the Built-in-Test method ([20]) for automated testing; (11) the actions from the RUP Deployment discipline are incorporated in the KBASE Exploitation discipline; (12) the infrastructure aspects of the RUP Environment discipline are incorporated in the KBASE Infrastructure discipline.

VI. KBASE Specification Language

The following ***Specification Techniques*** (ST) are selected to implement the actions of the method described in TABLE II. , as well as to create ontology objects static and dynamic behavior:

(1) ***Natural Language based ST*** (standard used - Natural Language Combinatorial Dictionary, NLCD). The specification technique is used to describe domain area ontology objects and behavior, the legal base, functional and non-functional requirements. The Combinatorial Dictionary contains information for the syntax functions, semantic-syntax functions and word-order zone of each lexeme. A Linguistic Processor verify the NL description of domain area objects and behavior. If the description is successfully verified they are integrated in the domain area ontology using the morphological NL processor, semantic-syntax NL processor, and business rules.

(2) ***Context based ST*** (standard used – Contextual Design, CD, [10]). CD models are used to improve the structuring of information about the domain area. The specification technique is used to describe the domain area ontology objects and behavior, the functional areas, functional and non-functional requirements. A Context Interpreter verifies the specified objects and behavior. If the specifications are successfully verified they are integrated by a Context Manager in the domain area ontology, or they are used for generation of a new program code.

(3) ***Event based ST*** (standard used – Case Management Model and Notation, CMMN, [13]; Business Process Model and Notation, BPMN, [12]; Unified Modeling Language, UML, [16] (Component Diagram, Infrastructure Diagram, and Package Diagram)). The specification technique is used to describe complex Information Systems (IS), realized as event based Multi-Agent Systems, including their Software Architecture, Infrastructure Model, and behavior. An IS Interpreter verifies the specified models and behavior. If the specifications are successfully verified, they are used for generation of IS program code.

(4) ***Process based ST*** (standard used – BPMN). The specification technique is used to describe domain area candidate business processes. The specified candidate processes are used for identification of candidate services and selection of services. After the selection of services, candidate processes are transformed in processes. A Process Interpreter verifies the specified processes. If the specifications are successfully verified they are integrated by a Process Manager in the domain area ontology, or they are used for generation of a new program code.

(5) ***Message based ST*** (standard used – BPMN, UML (Sequence Diagram)). The specification technique is used to describe the Server to Server (S2S) direct communication, including structure of messages, typical message objects, data types, data structures, and sequence diagrams describing the S2S valid and invalid interactions. A Message Interpreter verifies the specified messages, message objects and interactions. If the specifications are successfully verified they are integrated by a Message Manager in the domain area ontology, or they are used for generation of new program code.

(6) ***Service based ST*** (standard used – UML (all diagrams)). The specification technique is used to describe domain area services including their input and output objects, classes, attributes, methods, behavior, etc. A Service Interpreter verifies the specified objects and behavior. If the specifications are successfully verified they are integrated by a Service Manager in the domain area Service Repository, or they are used for generation of new service program code.

(7) ***Object based ST*** (standard used – UML (State Engine)). The specification technique is used to describe domain area ontology objects and their behavior, including object states, state transitions, transition conditions, and transition operations. An Object Interpreter verifies the specified objects and behavior. If the specifications are successfully verified they are integrated by an Object Manager in the domain area ontology, or they are used for generation of a new program code.

(8) ***Rule based ST*** (standard used – Decision Model and Notation, DMN, [14]). The specification technique is used to describe domain area simple and complex rules applicable to class attributes and diagrams transitions/ connectors/ associations, ontology establishment and restructuring, decision making, and knowledge management. A Rule Interpreter verifies the specified rules and behavior. If the specifications are successfully verified they are integrated by a Rules Manager in the domain area ontology, or they are used for generation of a new program code.

(9) ***Ontology based ST*** (standard used – Web Ontology Language, OWL, [25]). The specification technique is used to describe formally taxonomies and classification networks, essentially defining the structure of knowledge for various domains. An Ontology Interpreter verifies the specified ontologies. If the specifications are successfully verified they are integrated by an Ontology Manager in the domain area ontology, or they are used for generation of a new program code.

VII. KBASE Common Platform for Automated Programming

The architecture of KBASE Common Platform for

Automated Programming (CPAP) (Fig. 2) is built on the following principles:

(1) CPAP components are arranged in: (1) physical layer; (2) virtualization layer; (3) system administration layer; (4) components generator layer; (5) business components layer; (6) layer for integration with external systems. The layers communicate through a standard and semantic Enterprise Service Bus.

(2) CPAP components are designed to build: (1) embedded systems – for collecting information; (2) information systems – for information processing; (3) knowledge based systems – for automated processing of large data sets, for generating new components and for monitoring and re-configuration of CPAP.

(3) CPAP models in P11 are implemented by the (1) Embedded Systems Manager, (2) Information Systems Manager, and (3) Knowledge Base Manager. Based on the models, they generate in collaboration three meta-models in P11, namely, the IS Model, KBASE Model and Embedded Systems Model, as well as the integration definitions in P20.

(4) The static and dynamic behavior of the components is formally described by P20 definitions using one or more languages and standards such as BPMN (Business Process Model and Notation, [12]), UML (Unified Modeling Language, [16]), SysML (System Modeling Language, [15]).

(5) The static and dynamic behavior of the systems is formally described by P20 definitions using one or more languages and standards such as BPMN, UML, CMMN (Case Management Model and Notation, [13]), DMN (Decision Model and Notation, [14]), Net, OWL (Web Ontology Language, [25]).

(6) The components used at CPAP central level are generated by the (1) Embedded Systems Manager, (2) Information Systems Manager, and (3) Knowledge Base Manager, using P20 integration definitions, P10 repository of services and components, and P11 knowledge.

(7) The collaboration between CPAP and its external users (e.g. economic operators) is organized by Template Information Systems (TIS) generated by the TIS Manager. The TIS Manager is managing the IS Manager, KBASE Manager, and Embedded Systems Manager for the purposes of collecting data from external systems. The TIS Manager manages the generation of the TIS meta-model in P11 and TIS integration definitions in P20 for internal and external use. The TIS Manager subsequently generates TIS in collaboration with the Embedded Systems Manager, Information Systems Manager, and Knowledge Base Manager.

CPAP architecture is composed of six layers.

L1 Infrastructure Layer organizes and manages the hardware and communication infrastructure processes at physical level (P01 Hardware) and operating system level (P02 Real OS) using virtualization tools (P03 Virtualization).

L2 Cloud Layer automatically organizes the execution of the requested work in the cloud environment. It is performed under the control of the virtual operating system (P04 Virtual OS) using scaling mechanisms (P05 Cloud instances), and allocating the required physical and virtual resources (P06 Cloud cartridges).

L3 SOA Layer manages the execution of user assignments at service level (through P07 Application Servers) and process level (through P08 SOA Servers).

L4 Ontology Layer contains data (P09 Data), components and services (P10 Repository), and knowledge (P11 Knowledge) necessary for generating the required software products by using BPMN and/ or UML specifications, graphical interfaces, natural language, etc. The models included in P11 are subject to various automated verifications such as verification and modification based on predefined standardized knowledge bases [5], business process model quality assessment, and evaluation of business process semantic correctness.

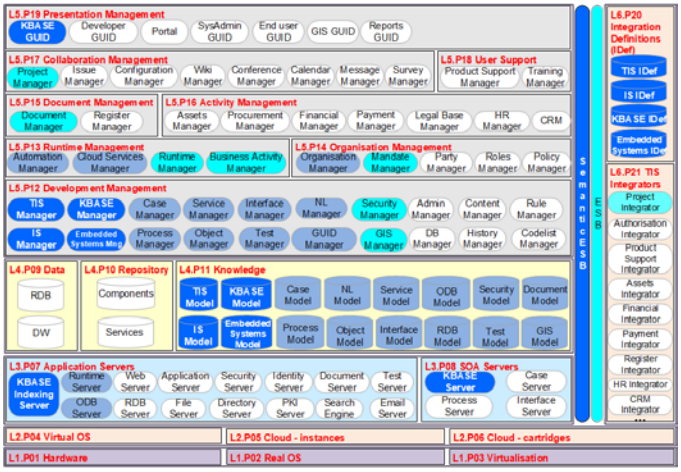

Fig. 2. CPAP Technological framework

L5 Business Layer contains a rich set of ready-made tools that are provided on demand by the user to assemble their software product. Components in L5 are organized in three groups of packages. The first group consists of components for development management (P12 Development management) responsible for managing the process of creating new software products in the context of CPAP and updating them on demand (possibly in real time) [5], [4]. The second group consists of components for runtime management (P13 Runtime management, P19 Presentation management). The third group consists of components for business logic management (P14 Organization Management, P15 Document Management, P16 Activity Management, P17 Collaboration Management, and P18 User Support).

L6 Integration Layer includes integration definitions and products for integrating systems, processes, and services developed within the organization by partners or third parties. Package P20 is composed of integration definitions (IDef) for TIS, Information Systems (IS), KBASE, and Embedded Systems. IDef represent integration conventions (e.g. communication protocols, XML schema definitions, communication sequences, information semantics, etc.) generated by P12 components. Integration takes place at

architectural or execution level. KBASE integration is performed to integrate the knowledge from various external systems. Embedded systems integration is performed at architectural or at business information flow level. Integration of components and services for information processing is based on IS IDef. TIS IDef for a specific domain area are generated by using IDef for IS, KBASE and Embedded systems. P21 TIS Integrators comprise TIS server components (TIS Server Level 1).

VIII. KBASE Efficiency Evaluation

Different combinations of the solutions incorporated in KBASE are evaluated in a large number of applications and prototypes. Some of the important results demonstrating the problem solving and efficiency improvement capabilities of KBASE are presented in TABLE III. below. These improvements are either quantitative or qualitative. Relevant improvements for the specific products are marked with √ at their intersection as applicable.

The products are the following: (No.1) the Module for Automated Programming of Robots (***MAPR***) [4] generates programs for Robot Control based on a natural language specification; (No.2) the Intelligent Product Manual (***IPM***) [5] is a software system which creates Product Manuals for industrial products; (No.3) the Built-in-Test Adaptive Document Display (***BIT***) [20] realizes the automation of Contract tests process, during the integration of a Commercial-off-the-shelf component with hosting system and the automation of Quality of Service tests process during real-time operation; (No.4) The Information Objects Manager (***IO.Man***) [5] is a tool which generates software components for document management through information objects structure and behavior formal specification; (No.5) Bulgarian e-Customs (***BeC***), (No.6) Bulgarian e-Health (***BeH***) and (No.7) Bulgarian e-Government (***BeG***) represent three enterprise information systems.

Quantitative improvements are related to considerable decrease of the necessary time and effort for software development (modelling, implementing, testing, customization) and modification as well as for the development of the integrated platform.

Qualitative improvements are in three aspects: (1) improvement of product adaptation to the domain area (adaptation to different users, standards, media, etc.); (2) improvement of product stability including prevention of system failure and failure solving; (3) quality of service improvement including product response time reduction, increased real-time product and product documentation update and synchronization.

TABLE III. Improvements in the KBASE applications

Type	Improvement	1	2	3	4	5	6	7
quantity	product specification time reduced	√				√		
quantity	programming time	√	√	√	√	√		
	reduced							
quantity	COTS (Commercial off-the-shelf) components integration time reduced			√		√	√	√
quantity	testing time reduced	√	√	√	√	√		
quantity	COTS components testing time reduced			√		√		
quantity	IT team reduced	√			√	√		
quality	adaptive to various domain areas	√	√	√	√			
quality	adaptive to different end users	√	√					
quality	adaptive presentation to various standards		√			√	√	√
quality	adaptive presentation to different media	√	√			√	√	√
quality	real time code synchronization		√					√
quality	real time documents synchronization		√			√	√	√
quality	prevent emergency system failure			√	√	√	√	√
quality	real time performance improvement			√	√			√

Below are presented examples on the achievement of the KBASE goals presented in section III. Measurements are based on project data.

(1) Following the KBASE process, the implementation shall start after 100% completion of the design. This modification to the RUP process resulted in 6 times increase of the number of reusable components (32% vs 5%), and 2 times decrease of the number of change requests related to the functional requirements. The improvements are observed on the eID system. It is compared to the eVote system. They are both part of the BeG program.

(2) Structural use case algorithmization style shall be used in the KBASE process. The comparison of two versions of the transit management system, part of the BeC program, demonstrates the following improvements: 5 times decrease of the number of use cases (1350 vs. 250), more than 3 times increase of the test coverage percentage (25 vs. 87%).

(3) Two main improvements of the development process are achieved with IO.Man. The first improvement is the reduction of the implementation and testing staff. For the functionality realized in IO.Man (8 state engines, each having 10-15 states, 30-40 arrows, and 60-80 objects) the business analysts staff was increased by 4 persons, and the implementers staff (GUI engineers, code engineers, and testers) was decreased by 9 persons. The second improvement is the time decrease for product development by more than 40%.

(4) Another effect of introducing the KBASE technology and tools is the change of IT development team profile. Although the application of the domain area ontology requires more business analysts in the team, the introduction of automation in the implementation process considerably reduces the required number of implementers and testers. In

the MAPR 61,9% of the effort is dedicated to the domain independent phase (performed once), 37,26 % - to the domain dependent phase (performed only during migration to a new subdomain or a new organization), and only 0,84% to the problem solving phase (performed daily during the lifetime of the system).

(5) Following the KBASE process the business processes shall be described with the BPMN standard instead of UML use cases. The introduction of event, process and object based specification techniques resulted in 2,5 times decrease of the number of classes in the Design model – 630 versus 245. Compared are two versions of a TIS Hospital care, part of the BeH program.

(6) The improvement obtained during the prototyping of two Product Manual versions for the same Fork Lift Truck is a reduction with 60% of the realization effort. Approximately 24 man/months are required for the realization of a non-automated and non-adaptive Product Manual version. The time for the realization of a more powerful, automated and adaptive PM version is reduced to 9 man/months.

IX. Conclusion

The suggested KBASE method integrates the RUP and Agile process frameworks, the technologies Software Engineering, Service Oriented Architectures, Automated Programming, Knowledge Processing, and Cloud Computing platforms.

Based on the KBASE method the following solutions are proposed:

(**1**) Software engineering ***standardization*** is improved by introducing KBASE ontology;

(**2**) ***Nine Specification Techniques*** based on different combinations of nine international standards for IS specification are introduced to improve the technological process;

(**3**) ***Contextual Design Method*** is introduced to improve the systematic collection of the initial information;

(**4**) ***Process-based, object-based and event-based specification techniques*** are introduced as more efficient replacement of large part of the RUP Use Case Specification;

(**5**) ***Automated programming*** is introduced to allow code generation after completion of the design model, which considerably increases components reusability level and the quality of the developed product;

(**6**) ***Built-in-Test method*** is introduced to improve test coverage and the efficiency of the development process.

The proposed solutions have been proved efficient in numerous developments and have important contribution to the improvement of the software development process.

Future work on the KBASE method is related to: (1) development of a new Unified Specification Language (USL) to integrate and optimize the selected specification techniques in the KBASE method; (2) build a first generation USL specification tool; (3) build a second generation of industrial Common Platform for Automated Programming; (4) improvement of knowledge processing intelligence.

References

[1] Axelos Limited, Managing Successful Projects with PRINCE2. The Stationery Office, 2017

[2] European Commission , ISA Programme, European Interoperablity Reference Architecture (EIRA) v.2.0.0, 2017. https://joinup.ec.europa.eu/asset/eia/description#EIA.

[3] F.Liu et. all., NIST Cloud Computing Reference Architecture, Gaithersburg: National Institute of Standards and Technology Special Publication 500-292, US Department of Commerce, 2011

[4] I.Stanev, "Method for Automated Programming of Robots," Knowledge Based Automated Software Engineering, Cambridge Scholars Press, Cambridge, pp.67 – 85, 2012.

[5] I.Stanev, K.Grigorova, "KBASE Unified Process," Knowledge Based Automated Software Engineering, Cambridge Scholars Publishing, Cambridge, pp. 1 – 19, 2012.

[6] I.Stanev, M.Koleva, "KBASE Technological framework – Requirements," 2015 17th International Conference on Semantic Interoperability and Integration (ICSII), Rome, 2015.

[7] I.Stanev, M.Koleva, "Method for information systems automated programming," 2017 11th Mediterranean Conference on Information Systems (MCIS), Genoa, 2017.

[8] I.Stanev, Method for automated programming of Robots, PhD Thesis, Department Informatics and Information Technologies, University of Ruse, 2014

[9] J.-D. Lovelock et all., "Forecast alert: IT spending, worldwide", 4Q17 update, https://www.gartner.com/doc/3841567, December 2017.

[10] K.Holtzblatt and H.Beyer, Contextual Design, Second Edition: Design for Life. Morgan Kaufmann Publishers. San Francisco, 2016.

[11] Object Management Group (OMG), Software & Systems Process Engineering Metamodel Specification (SPEM) v.2.0. http://www.omg.org/spec/SPEM/2.0/, 2008

[12] OMG, Business Process Model And Notation v.2.0.2. http://www.omg.org/spec/BPMN, 2014

[13] OMG, Case Management Model And Notation v1.1. http://www.omg.org/spec/CMMN, 2016

[14] OMG, Decision Model And Notation v1.1. http://www.omg.org/spec/DMN, 2016

[15] OMG, System Modeling Language v.1.5. http://www.omg.org/spec/SysML/, 2017

[16] OMG, Unified Modeling Language v.2.5.1. http://www.omg.org/spec/UML/, 2017

[17] P.Abrahamson, O.Salo, J.Ronkainen, J.Warsta, Agile software development methods: Review and analysis (Technical report). VTT. 478., 2002

[18] P.Kruchten, The Rational Unified Process: an Introduction, Addison-Wesley, 2000.

[19] P.Mell and T.Grance, "The NIST Definition of Cloud Computing," Gaithersburg: National Institute of Standards and Technology NIST Special Publication 800-145 US Department of Commerce. p.7, 2011.

[20] Project EU IST-1999-20162 Development and Applications of New Built-in-Test Software Components in European Industries, Software Architecture, 2003

[21] Project Management Institute, A Guide to the Project Management Body of Knowledge Sixth Edition, 2017.

[22] The Open Group, Service-Oriented Architecture Ontology v.2.0, http://www.opengroup.org/soa/source-book/ontologyv2/index.htm, 2014.

[23] The Open Group, The Open Group Architecture Framework (TOGAF) v.9.1, http://pubs.opengroup.org/architecture/togaf9-doc/arch/.

[24] U.S. Bureau of Labor Statistics, Occupational Outlook Handbook, https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm#tab-6.

[25] W3C, Web Ontology Language v.2, https://www.w3.org/standards/techs/owl#w3c_all, 2012.