DOI: https://doi.org/10.15276/aait.03.2019.2
UDC 004.4'2

ANALYSIS AND SYNTHESIS OF THE RESULTS OF COMPLEX EXPERIMENTAL RESEARCH ON REENGINEERING OF OPEN CAD SYSTEMS

Stanislav S. Velykodniy
ORCID ID: https://orcid.org/0000-0001-8590-7610, velykodniy@gmail.com
Odessa State Environmental University, 15, Lvivska Str. Odesa, 65016, Ukraine

ABSTRACT

The article presents the final results of scientific research on the development of models and methods of reengineering, as well as technologies of multilingual recoding of open systems of automated design. The common feature of all software systems lies in the fact that there is an evolutionary aging of the types of support under the influence of time and other integral factors of information, namely, updating: operating systems, programming languages, principles of the operation of distributed data processing systems, etc. Such a tendency leads to deterioration of speed, information and communication, graphic, time and other characteristics, up to a complete system failure. Reengineering is a process that allows creating quickly and easily new, improved software systems, using the experience of previous software products. The purpose of the article is to systematize the results of the integration of reusable component, which have been accumulated by developers over a certain period of development of sectoral computer-aided design systems in updating the software structures of ready-made resources. Based on the obtained scientific and practical results, the analysis of the developed models and methods of reengineering of types of support for open computer-aided design systems is performed. In general, reengineering includes the processes of reorganization and restructuring of a software system, conversion of individual system components into another, more modern programming language, as well as the modification or modernization processes of the structure and data system. The study involved the following methods: assembly, specifying, synthesizing and compositional programming, methods of generative and recognizing grammars. At present time, the process of a new software products design is not very effective without the use of the UML methodology, but when it is applied, the speed of development increases by times. UML as a language for a graphical description for object modeling, in addition to simple design, supports also the function of generating and reengineering code based on model data, as discussed in the article. The distinctive feature of this research is the ability to support the work of more than ten most popular programming languages. In applying these technologies, it is possible to automate the process of software components recoding and, therefore, to free the working time of programmers from routine reprogramming and reduce the probability of occurrence of structural errors inherited from the previous system. The use of the obtained results will improve significantly the efficiency of the application of automated design systems in such fields of their use as: mechanic engineering, telecommunications, production and transport management, education, etc. The developed models and methods will be useful to system architects and program engineers involved in redesigning software already being in their multi-year operation.

Keywords: CAD Systems Reengineering; UML Methodology; Multilingual Transcending; Linguistic Structures; Generative Grammar

***For citation*: Stanislav S. Velykodniy. Analysis and Synthesis of the Results of Complex Experimental Research on Reengineering of Open Cad Systems. *Applied Aspects of Information Technology.* 2019; Vol.2 No.3: 186–205. DOI: https://doi.org/10.15276/aait.03.2019.2**

INTRODUCTION

The main goal set before the computer-aided design (CAD) for any purpose is to reduce the design time of the object and reduce the personnel required for this design, and as a consequence, the cost of the finished design object.

The common feature of all CADs is that under the influence of time and other inevitable factors of informatization (upgrades: operating systems, programming languages, principles of the operation of distributed data processing systems, etc.) there is an evolutionary aging of the types of CAD maintenance. This tendency leads to a deterioration of speed, information and communication, graphic, time and other characteristics, up to the complete system failure.

Hence, it follows that the CAD should be one that develops. According to the world tendencies of development, CAD relies on a life cycle of 3-4 years. Of course, when updating the design object – CAD is also updated. At this stage, the question arises: what to do when the system is tightly tied to the design object? There is one answer to this question: it is necessary to apply reengineering on the CAD.

CAD reengineering is the evolution of the system through radical change in order to increase the usability, maintenance or change of its functions. It includes processes for the reorganization and restructuring of the CAD, a transfer of individual components of the system to another, more modern programming language (PL), as well as processes for updating or modernizing the structure and data system. In this case, architecture the system may remain unchanged.

CAD reengineering is a target for obtaining a new component by executing a sequence of operations for making changes, upgrades or modifications [1], as well as reprogramming

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

individual components of the CAD. It is implemented by a set of models, methods and processes that change the structure and capabilities of components in order to obtain a module with updated capabilities. New components are identified by the names that are used when creating component configurations and frameworks of CAD [2].

From the technical point of view, reengineering is a solution to the problem of CAD evolution by modifying its components and adapting the architecture to a new environment [3], in which components are placed according to the configuration of the operating system. The reason for the evolution may be the change of the PL of CAD (for example, outdated: Fortran, Cobol or even C, etc.) with the transition into modern object-oriented languages (Java, C #, Python, etc.).

The goal of the article is to systematize the results of the integration of reusable component (RUC), accumulated by developers at a specified time of the industry CAD development, into new program structures of finished resources.

GOAL SETTING

One of the main tasks of modern programming is the creation of theoretical and applied foundations for construction of complex programs with simpler program elements that are written in modern PLs. In fact, solution of this problem is accomplished by collection, combination or integration of heterogeneous software resources and RUCs, including modules and programs for the implementation of a particular domain.

Linguistics, which studies language laws, models and rules, is a scientific basis of any language (including the programming language). Generative linguistics, which was founded by Avram Noam Chomsky (in the Soviet times, sometimes was interpreted as “A. N. Khomsky”), who created the revolution in language studies, is a special branch of linguistics that should be used in the structure of programming languages.

By way of the task of correct chains, formal grammars are divided into generative and recognizing. The generative grammars include those ones by which it is possible to construct any correct chain with an indication of its structure and it is impossible to construct any wrong chains. For the first time, the notion of generative grammar was proposed by A. N. Chomsky. Recognizing grammar is a grammar that allows to establish the correctness of an arbitrary chain and, if it is correct, to find out its structure. Formal languages include, in particular, artificial languages for communication between the operator and the computer (programming languages).

The lingware of CAD considers the construction of a software system with one or more (mutually agreed) PLs, each of which is based on the rules of a particular grammar and is considered by the author of the presented article in [4] and [5].

The problem of CAD reengineering of various industrial purposes has been discussed in detail in [6]. The methodological principles for the CAD reengineering have been laid down in [7]. Problems of methods formation for conversion of software for various software systems, for example, SCADA-systems, was considered in [8].

Generalization of the stages of reengineering of complex information systems and technologies is given in [9]. Formation of the method for calculating the indicators of project evaluation in the implementation of reengineering software systems is presented in [10]. Models and restrictions on the use of reengineering on software systems are identified in [11]. The method of presentation of an estimation of reengineering of software systems using project factors is formed in [12].

ANALYSIS OF RECENT RESEARCH AND PUBLICATIONS

At present, there is a large number of software that performs a large number of specialized tasks. Some of them are tied to only one branch of industry, while others are used in large numbers, but the trend goes through the specialization of software products in general [13].

Corporations that develop CAD, design a lot of specialized software products, for example – AutoDesk. They have a complete set of programs for work with engineering structure (Inventor), architecture (ArchiCad), design (3dMAx) and design in a broad sense (AutoCad) [14].

Thanks to powerful computational tools in the CAD using integrated modules containing banks and databases (DB) of ready-made design solutions, it is possible to quickly make adjustments to the necessary parameters of products (sizes, form, order of processing, etc.), which are manufactured; as well as to the sequence of technological operations, that is to reorient the whole production process [15], [16].

Such a reorientation (in the broadest sense) of a CAD from a database can be defined as reengineering of information provision [17].

Reengineering includes processes of reorganization and restructuring of the software system [18], the conversion of individual system components into another, more modern

programming language, as well as processes for updating or modernizing the structure and data system [19].

The methodology outlined in [20] is useful as a basic trajectory of research. Principles [21] and studies in [22] have suggested practical aspects of reengineering models [23].

One of the important components of the CAD is the computer graphics, which is a collection of tools and techniques through which input, transformation and output of graphic information from specialized environments are carried out [24].

Computer graphics is an actual branch of design and application of computing systems that are intensively developing in recent times [25]. The term "computer graphics" means computational processing of information, as well as the output of results in the form of various graphic images. The data necessary for results displaying in a graphic format is created based on graphical information [26].

Particular interest in computer graphics has become apparent in connection with the intensive development and introduction of currently free and open CADs not only in engineering, instrumentation, radio electronics, interior design, but also in other areas of production and training [27].

One of such CADs is BRL-CAD, a specialized cross-platform open source system. It represents a powerful 3D CAD for solid modeling using CSG methods. This CAD includes an interactive geometric editor, parallel beam tracing, rendering, and geometric analysis [28].

BRL-CAD was developed for about 40 years and has been used by the US armed forces. The entire BRL project works from the source code, thus it can be used on any platform: GNU/Linux, MacOS, Solaris, and Windows.

Here are some definitions of open source software and their design technologies.

Source code (usually just "crumbled", also "sources", "program code", "text of the program") – any set of instructions or announcements written in the programming language and in a human-readable [29]. The source code allows the programmer to communicate with the computer with the help of a limited set of instructions [30].

Program source code is a set of files that are required to convert from a human-readable form to some types of computer executable code [31]. There are two possible ways to execute a source code: compiling into a computer code using a compiler (designed for specific computer architecture) or executing directly from the text with the help of an interpreter [32].

One of the first CADs, capable of developing in both these areas, appeared because in 1979 the US Army Ballistic Research Laboratory (BRL), now the United States Army Research Laboratory, expressed a strong need for instruments and tools that could help in computer simulation and engineering analysis of weapons systems (tanks, rockets, airplanes, etc.) and their working conditions [33].

When none of the CADs existing at that time appeared to be ready for this purpose, BRL developers began to collect a set of utilities capable of interactively viewing and editing geometric models trees. Programmers began to develop their own suite of applications that were designed to display, edit and combine geometric models. In this way, the BRL-CAD, the application package for Constructive Solid Geometry (CSG), was created.

The first public release was made in 1984. In December 2004, BRL-CAD became an open source project. It is very important that BRL-CAD is licensed under the terms of BSD and GNU GPL.

From now on, this CAD has been constantly evolving, and new opportunities have emerged, but now the very linguistic provision of the database submission (C language) in the BRL-CAD environment requires the transfer (reengineering) to high-level languages (C or C#).

Today, thanks to about a million lines of C code, BRL-CAD has become the most powerful graphical modeling package that has been used by more than 2,000 organizations around the world.

METHODS OF RESEARCH

The following methods have been used during the research:

– method of assembly programming, which explores the program elements, which are completed with modules, objects, components, services, etc.;

– method of concretizing programming, used in the presence of some universal software;

– method of synthesizing programming, which is used from setting of problem, which is formed as a model of calculation and specification of the program for solving the set problem;

– method of composite programming, used in the organization of functions and data in software systems;

– method of assembly programming, used in the presence of a bank of modules and components of reuse;

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

– method of Chomsky's generative and recognizing grammars that are used in the construction of linguistic chains for formal programming languages.

MAIN MATERIAL OF THE ARTICLE

The most important action for skeletal structure and reengineering is the creation of two diagrams [34]: class diagrams and component diagrams. It is from them that the code of the future software product [35] is generated. All other diagrams have an auxiliary (relief) role [36] and should be used at someone's own discretion.

The implementation of this phase depends on a technical specification and modern market requirements, built in the so-called top-list of the most popular programming languages, which include: Java; C#; C++; PHP; Python [37] and more.

The choice of a CASE-tool depends on the user's preferences. According to the author's opinion [38], the most optimal CASE-tool that supports the import and generation of code written in the languages mentioned above is Enterprise Architect (EA). It is EA (version 14.0) to be considered by us as an effective transcoding tool.

Proceeding from the problem, in the selected simulation environment, there should be a lot of possibilities in addition to the standard set of diagrams (15 pcs.), it is necessary to carry out efficiency analysis, which is business diagrams, synchronization diagrams, etc.

In one of the most well-known environments, Rational Rose, business diagrams and all subsequent metrics are not implemented effectively [39]. At the moment, there are few CASE-tools that support the correct code generation in many languages, especially if you do not count on the language itself, and a software product that is developing in this direction and has the best prospects for learning. The convenience of work and the simplicity of the interface were equally important. Of course, in terms of interface simplicity, EA fails to keep pace with its counterparts, but it is completely overlapped by its efficiency [40].

PROCESS OF CODE GENERATION

UML, as a language for the graphical description for object modeling [41], supports, in addition to simple design, generation and reengineering of code based on model data. As noted earlier, code generation occurs from two diagrams – class and components.

The component diagram serves as a convenient link for us to connect classes and entire packages that consist of similar modules. In the EA CASE-tool, the component diagram does not have a direct effect on code reproduction from the model, it only performs auxiliary functions. Very revealing is the fact, that when creating a complex software product, it is not very convenient to reproduce a separate class diagram, so further binding to the component diagram consists precisely from the transfer of the "Class" type modules to this diagram.

Therefore, in the EA software product, there is such a convenient type of component, called "Packaging Component" – this component has a wide internal structure in the form of another diagram. This internal diagram was created precisely for the convenience of working with modules of the "Class" type, but the possibilities of EA allow us to create diagrams of any type there (if to investigate the methodology in detail, it is quite convenient because, for example, you can show the structure or methodology of business diagrams) When you create this component, a new component diagram automatically appears inside it, which is very convenient to load class modules. All modules that will be created or moved to this diagram are automatically tied to the component in which they are located.

To generate a code, we will open a physical location of modules of the "Class" type on the screen, and then select all the necessary modules that we want to play. Next, we indicate the location for each element being created, and step by step we will complete the generation (Fig. 1).

In essence, this process of code generation is complete. However, how the skeleton structure of our software product is related, it depends on what types of communication and which variables, operations, and attributes are specified in the class diagram. It should not be forgotten that this is only a general basis (template) for the code, all the main code and processes must still be written by programmers (Fig. 2).

The convenience of such a generation consist in a general form structuring, the assistance in the distribution of template tasks between programmers, and almost complete exclusion of the problem of the incompatibility of modules, because the entire structure is already connected initially.

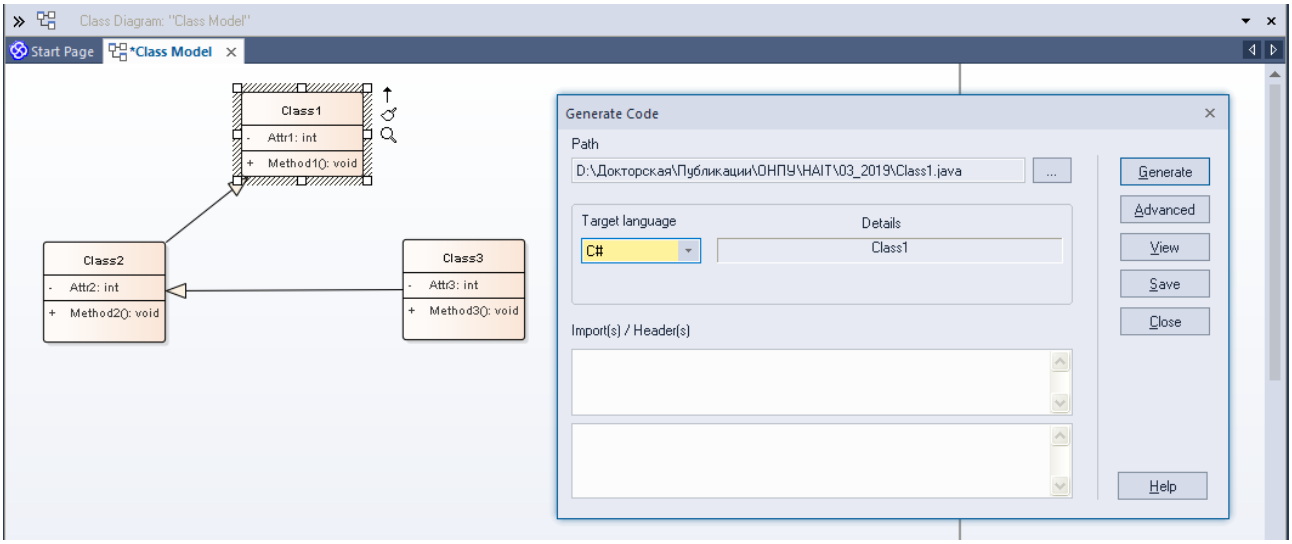

Fig. 1. **Process of code generation**
***Source*: compiled by the author**

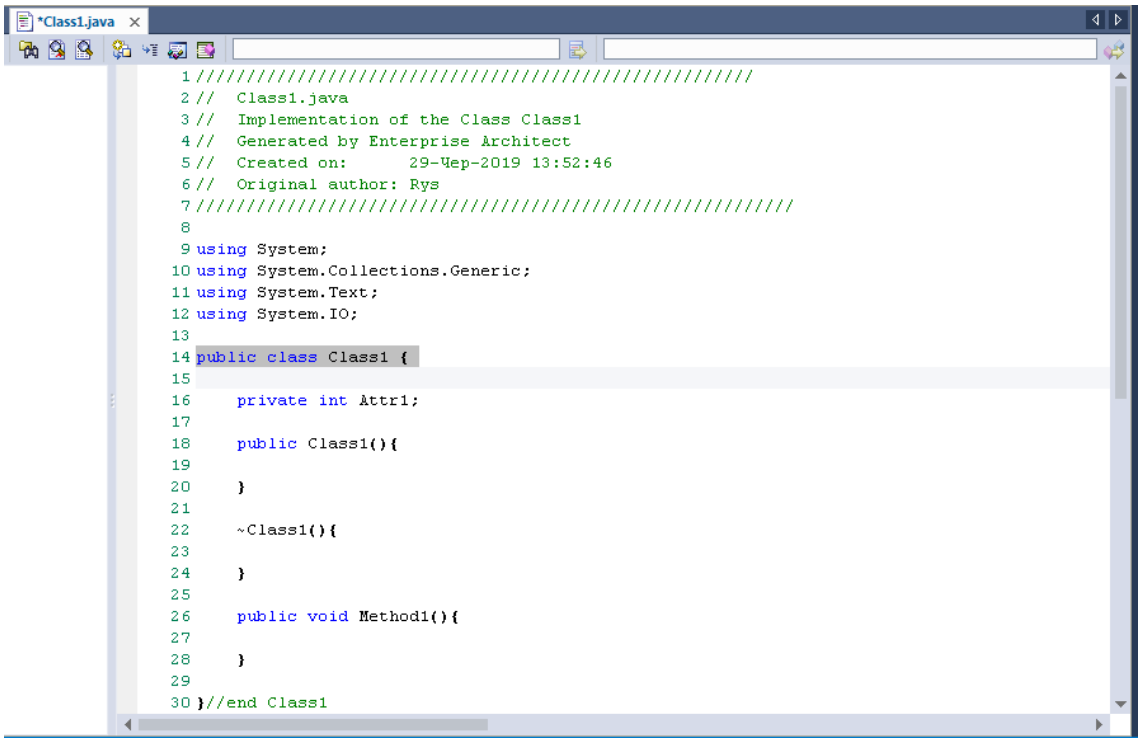

Fig. 2. **Generation result**
***Source*: compiled by the author**

DECOMPOSITION BRL-CAD SOFTWARE INTO COMPONENTS

Class Diagram

In order to create new software based on the old one, you need to analyze the structure of the primary software product. The structure of BRL-CAD is presented as program code in C language and is divided into a large number of modules, each of which contains one or more classes interrelated or related to other modules. In addition to each module understanding, the individual task is to understand the relationship between classes and make a coherent presentation. Since this is a direct work with the code, we will represent the structure on diagrams created especially for this: the diagrams of classes and components.

The very general class diagram for the primary software product is to be compiled first, so that it is rather difficult to compose the immediately connected diagram of the classes and components in terms of the complexity of the work, while, as

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

experience has shown, the effectiveness of such action does not exceed the time costs for work with individual diagrams. That is, when decomposing any product, the best decision will be decomposing into components, while, when generating a new product, it is better to create an exact and complex interrelation.

To begin with, we need to find a "generalizing" module (if, of course, it is present). Module, in which all the main working submodules are represented, is called generalizing. This kind of representation is almost always used in the work of open source software products for more understandable and rapid parsing and further modification of the code. It was decided to adhere to this rule, and since it is precisely the same as building an open source code, then you need to structure the code as best as possible, that is to make it "clean".

In BRL-CAD, the entire C programming language is located in the "incude" folder. Open the file "brlcad.h", which is our "generalizing" module and see what kind of connection it has:

```
/* system headers presumed to be available
/* basic utilities */
/* vector mathematics */
/* non-manifold geometry */
/* basic numerics */
/* database format storage types */
/* raytrace interface constructs */
/* trimmed nurb routines */
/*   the   write-only   database   library
interface */
/*   in-memory   representations   of   the
database geometry objects.  these
 * are subject to change and should not be
relied upon.
/* database object functions.
```

There is a connection of submodules responsible for certain operations here. Moreover, as we have already mentioned above, even explanatory comments are provided for improving the convenience of working with the code. Consequently, it is from here that the generation of a new structure will begin. Each submodule represents a whole set of files with related classes associated with them, so all of these files will be presented as "Interacting Packets" in the easiest way on the diagram.

When creating the first package, which is called "include" (in the name of the folder where the entire executable code is located), it is proposed to select the type to be used in the subsequent, internal, diagram (Fig. 3).

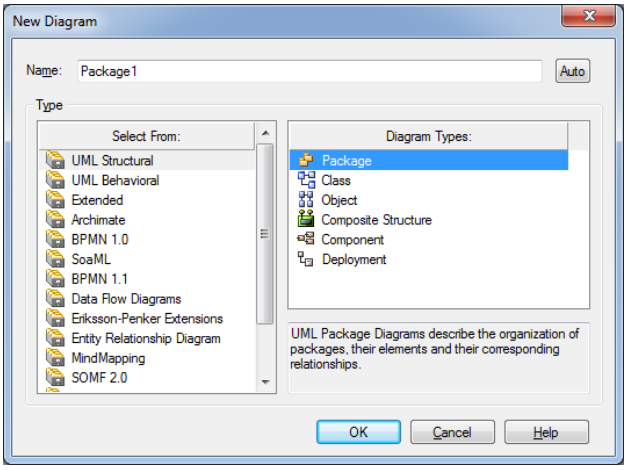

Fig. 3. **Selection of the diagram type inside the package**

***Source*: compiled by the author**

Since all subsequent modules will be separate (interacting only at the data package transmission levels) classes of relationships, it is more likely to create them in the form of the same packages. When creating these packages, you must again select the type of diagram that will be used internally. Next, choose a class diagram, so the interaction inside these modules already takes the form of classes with their operations and attributes. After adding all the major packages, the project browser looks like it is shown in Fig. 4.

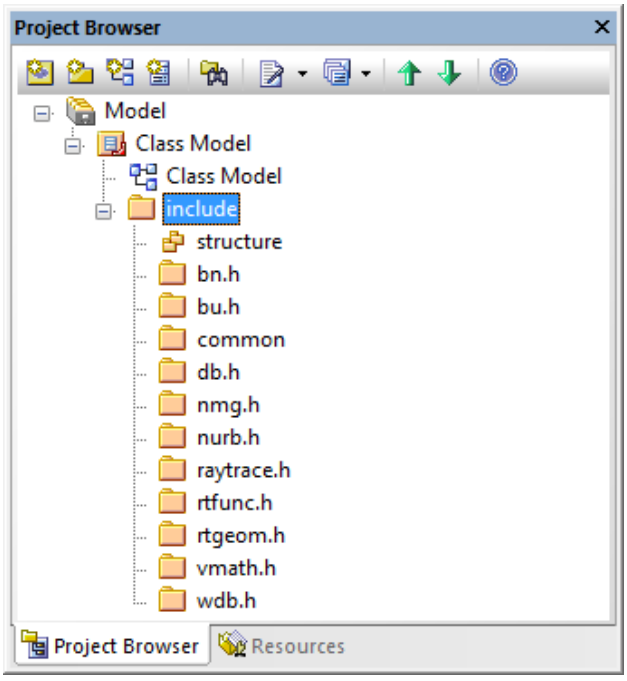

Fig. 4. **Project browser after adding packages**

***Source*: compiled by the author**

Now when the main structure is complete, proceed in particular to each package, with its breakdown by classes and interconnections.

Relations between packages will be considered after their internal structure is determined.

Let's start with the packages. The first package is called “bn.h” and on the hard disk it is represented by a single file with the same name. The first thing to do is to restore its structure using the command “Import C File” from the “Source Code Engineering” graph.

In the end, we get the result, from which it is evident that just one class has recovered, which is strange. The file we tried to recover was not fully read, as evidenced by the error graph. The report window was also opened (Fig. 5).

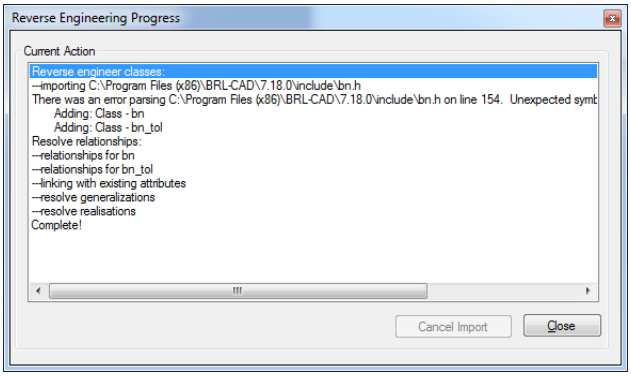

Fig. 5. **Reverse engineering report**
***Source*: compiled by the author**

In it, we see that besides reproduction of this class, all the possible types of connections are reproduced in the same way and all types of relations are restored. However, this is not enough, so when we open the window of the physical file of the code location, we see that there are many more classes there. The very first difference that arrests attention is that the class played on the screen is the only one that defines some variables, while all the following use the #define command, which serves to declare any constant. This constant can be taken from other modules.

The file that we are interested in consists mainly of constructs such as:

a) Struct bn_tol – class declaration;

b) BN_EXPORT BU_EXTERN (void anim_tran, (mat_t m)) – process declaration;

c) #Define bn_cx_add (ap, bp) {(ap) -> re + = (bp) -> re; (ap) -> im + = (bp) -> im;} – declared constants.

Since we are not satisfied with the results of the reverse engineering, we have to complete the analysis on the diagrams manually. To do this, we need to understand what exactly to look for in the files. First of all, of course, the declared classes and their variables are of interest, the general structure and the shape of the diagram depend on it. After the classes and variables are recreated, it is necessary to take on the processes that the given classes will take.

The process of reproduction of a full-fledged structure up to each variable is not necessary, in the reengineering it is important to understand the process of software modules work and to make the most understandable scheme, which will be convenient to work not only on someone’s own, but also to explain the principles to the programmers who will reproduce the new modules. Therefore, we will not specify many small classes and processes or we will combine them into larger diagram processes.

The example of a schematic association of processes.

The program code contains many small processes with explanations such as:

```
BN_EXPORT BU_EXTERN(void anim_dy_p_r2mat,
                    (mat_t m,
                     double y,
                     double p,
                     double r));

/**
 * @brief Make a view rotation matrix,
given desired yaw, pitch and
 * roll. (Note that the matrix is a
permutation of the object rotation
 * matrix).
 */
BN_EXPORT BU_EXTERN(void anim_dy_p_r2vmat,
                    (mat_t m,
                     double yaw,
                     double pch,
                     double rll));

/**
 * @brief Make a rotation matrix
corresponding to a rotation of "x"
 * radians about the x-axis, "y" radians
about the y-axis, and then
 * "z" radians about the z-axis.
 */
```

The comments clearly indicate: what each process replies for (once again remember that this is one of the conveniences of working with open source software, although this convenience has a number of shortcomings). In our example, these processes are responsible for rotation of the selected object in the given coordinate system: *X, Y, Z*, under which it is monitored, any deviation.

In the diagram, we generally call this a Rotation process and will not go into details, because at least later, when working with programmers we will have to develop a new model, taking into account the specificity of the programming language. By doing this, we will facilitate and reduce the process of developing the diagrams, because there are more than thirty such processes in one such “bn.h” file.

We begin to construct the diagram, taking into account combination of non-essential classes. Let's show the stages in detail on one of the classes (Fig. 6; Fig. 7):

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

```
struct bn_unif {
    unsigned long magic;
    long msr_seed;
    int msr_double_ptr;

    double *msr_doubles;
    int msr_long_ptr;
    long *msr_longs;};
```

```
#define    BN_CK_UNIF(_p)        BU_CKMAG(_p,
BN_UNIF_MAGIC, “bn_unif”)
#define     BN_CK_GAUSS(_p)      BU_CKMAG(_p,
BN_GAUSS_MAGIC, “bn_gauss”)
```

The final representation of the class diagram for the file “bn.h” is shown in Fig. 8.

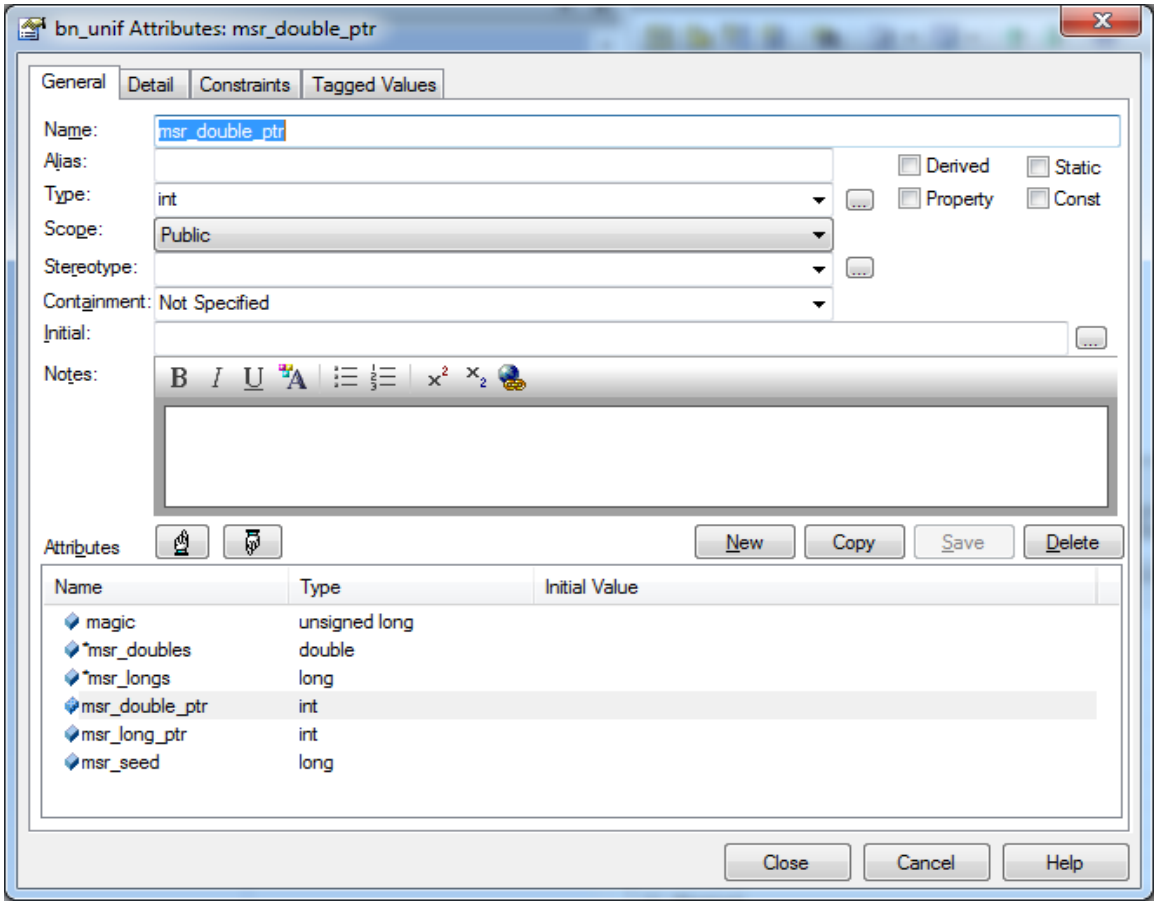

Fig. 6. **Filling in the attributes of bn_unif class**
***Source*: compiled by the author**

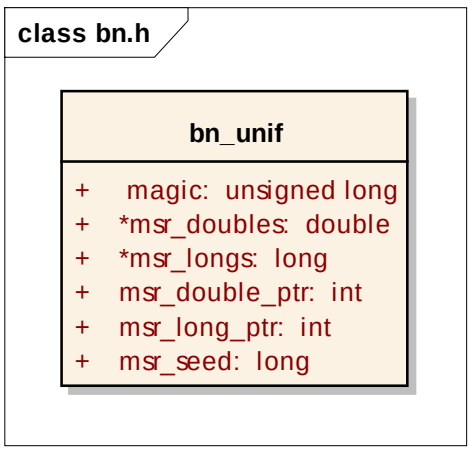

Fig. 7. **Final representation of bn_unif class**
***Source*: compiled by the author**

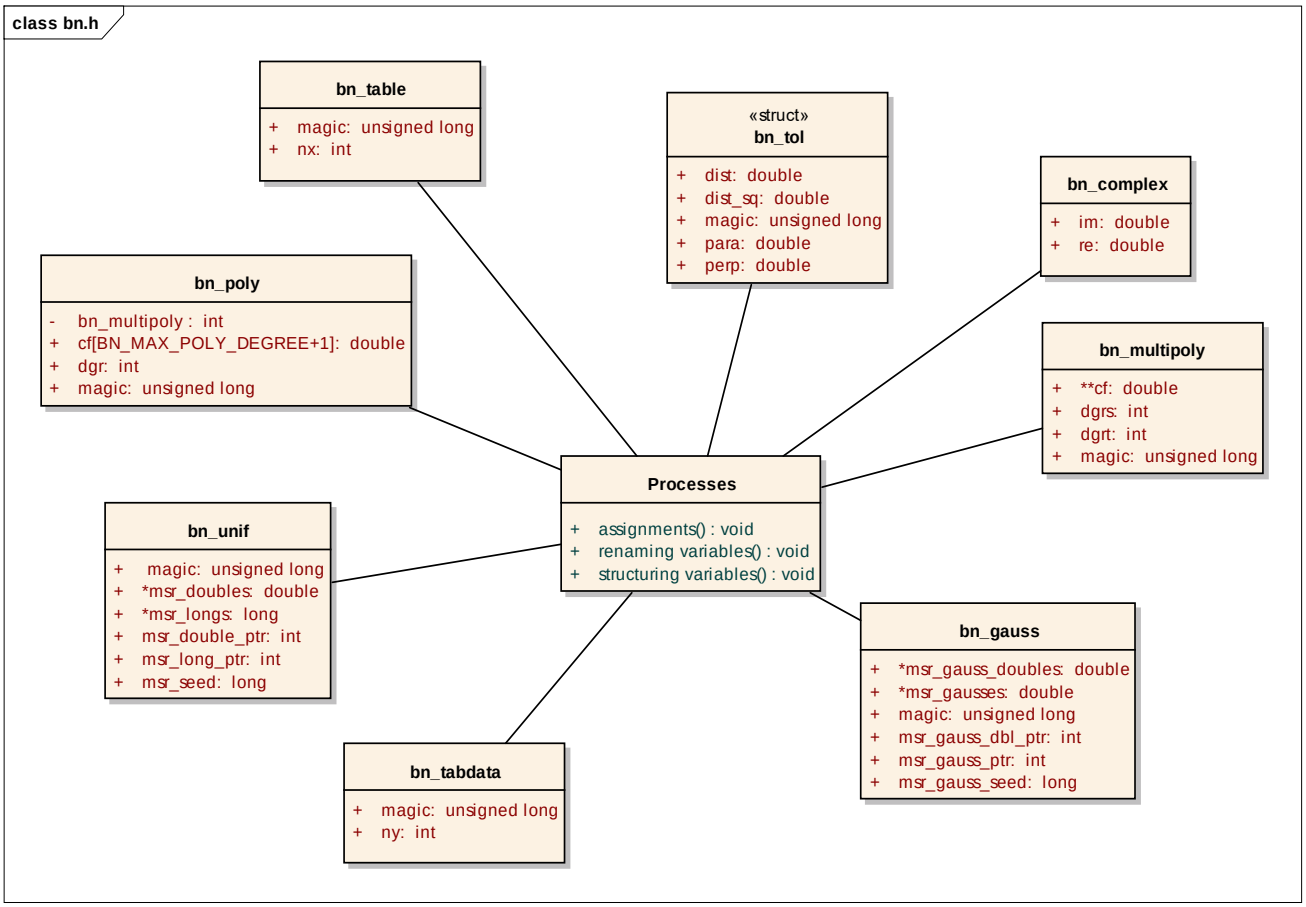

Fig. 8. **Class diagram for "bn.h"**
***Source*: compiled by the author**

It should be taken into account that this diagram is not a complete reflection of the entire file, therefore less significant classes take a lot of space, but they are not essential in the reflection. The same applies to processes – there are more than two hundred of them, while they only announce or structure the data, so they are schematically displayed, but due to the fact that each class is inclined to influence any processes, it is necessary to show it on the diagram (Fig. 9).

"Association" is chosen as communication, because this communication carries information about the relation between objects within the software. Associations can be specified and display which class is it and how it is related to others.

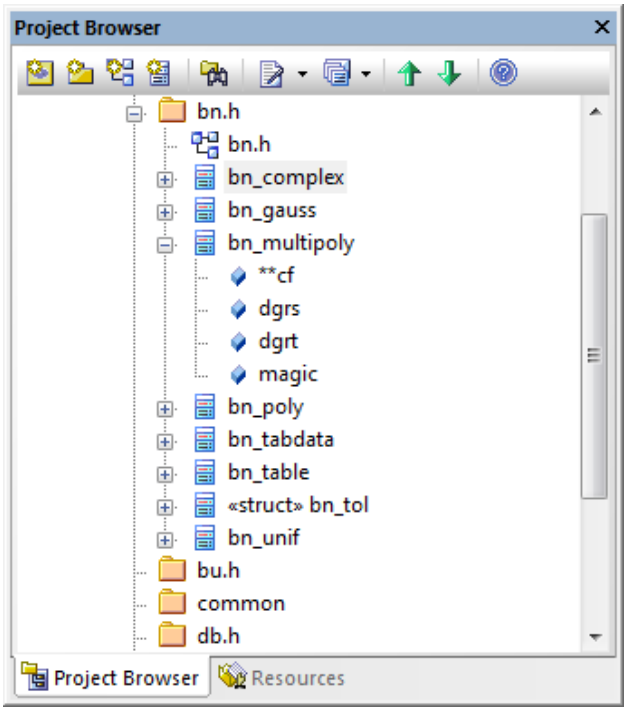

Fig. 9. **Diagram in the form of a browser**
***Source*: compiled by the author**

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

After completing the analysis of the first file called “bn.h”, the last step in working with it is to show the connection of this “package” with others. At the beginning of the file, there are the following lines:

```
/* interface headers */
#include “bu.h”          /* required for
BU_EXTERN, BU_CKMAG */
#include “vmath.h” /* required for mat_t,
vect_t */
```

This entry indicates that we have a relation with other packages included in this diagram, so this link should also be displayed. It is necessary to use the type of connection “dependency”, so the execution of mathematical and other functions in the file depends on these two connected components. As a result, in the package diagram, this will look the same as in Fig. 10.

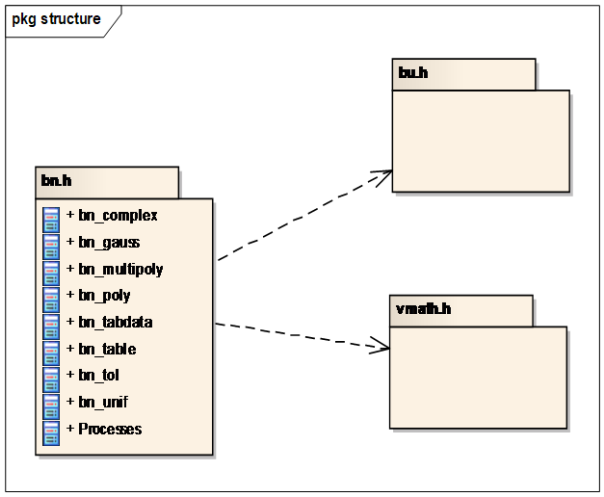

Fig. 10. **Relationship of “dependence” between packages**

***Source*: compiled by the author**

In some cases, the relationship between classes can be neglected because the work with them is at the presentation level of the files (packages), which means that access to the class from the “bmath.h” file will not occur from the inside, but from the outside, for example from the “bn.h” file. Similarly, in some cases, the links will be displayed in the title, for example, the relationship between super- and child classes (Fig. 11).

That is, the “rt_revolve_internal” class is the descendant of the “rtgeom” class.

At this stage of the study, the “bu.h” and “vmath.h” packages are not yet filled with classes and functions, so their reflection, so far, is purely schematic.

Thus, it was investigated how to restore files from C language manually. All files in this software project were considered similarly to display the complete diagram (Fig. 12).

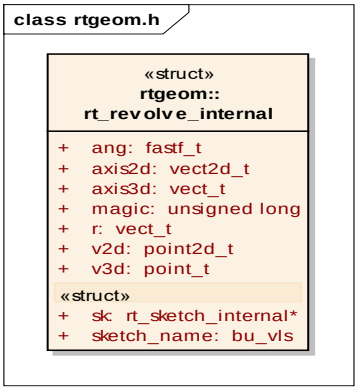

Fig. 11. **Descendant class**

***Source*: compiled by the author**

Component diagram

Component diagram is the second diagram that participates in generating the code of the future software product and for this purpose it should be associated with the first diagram – a class diagram. We will analyze a component diagram for the primary software product according to the classical canons [42], with a breakdown into 3 parts:

a) deployments that ensure the direct execution of the system's functions – such components may be shared libraries with the “*.dll” extension;

b) work products are usually files with source code programs, for example, with the extension “*.h” for the C language;

c) executive, representing modules with the extension “*.exe”.

In the first step, not so much the program code is necessary as the physical location and the overall software architecture (Fig. 13).

There are 4 folders in this structure in addition to executable files (“archer” and “brlcad”). The “bin” folder contains the main and auxiliary “exe”-files that run separate modules (the specifics of a software product: each module does not integrate into the interface but connects to a separate window).

The folder “include” contains a work product - files with texts in “C” programming language. The folder “lib” contains the library files, and the folder “share” contains text documents with descriptions, license agreements, pages on the Internet, generally auxiliary files.

Having considered this structure, you can already build a “skeleton” of the diagram and it will look like on Fig. 14.

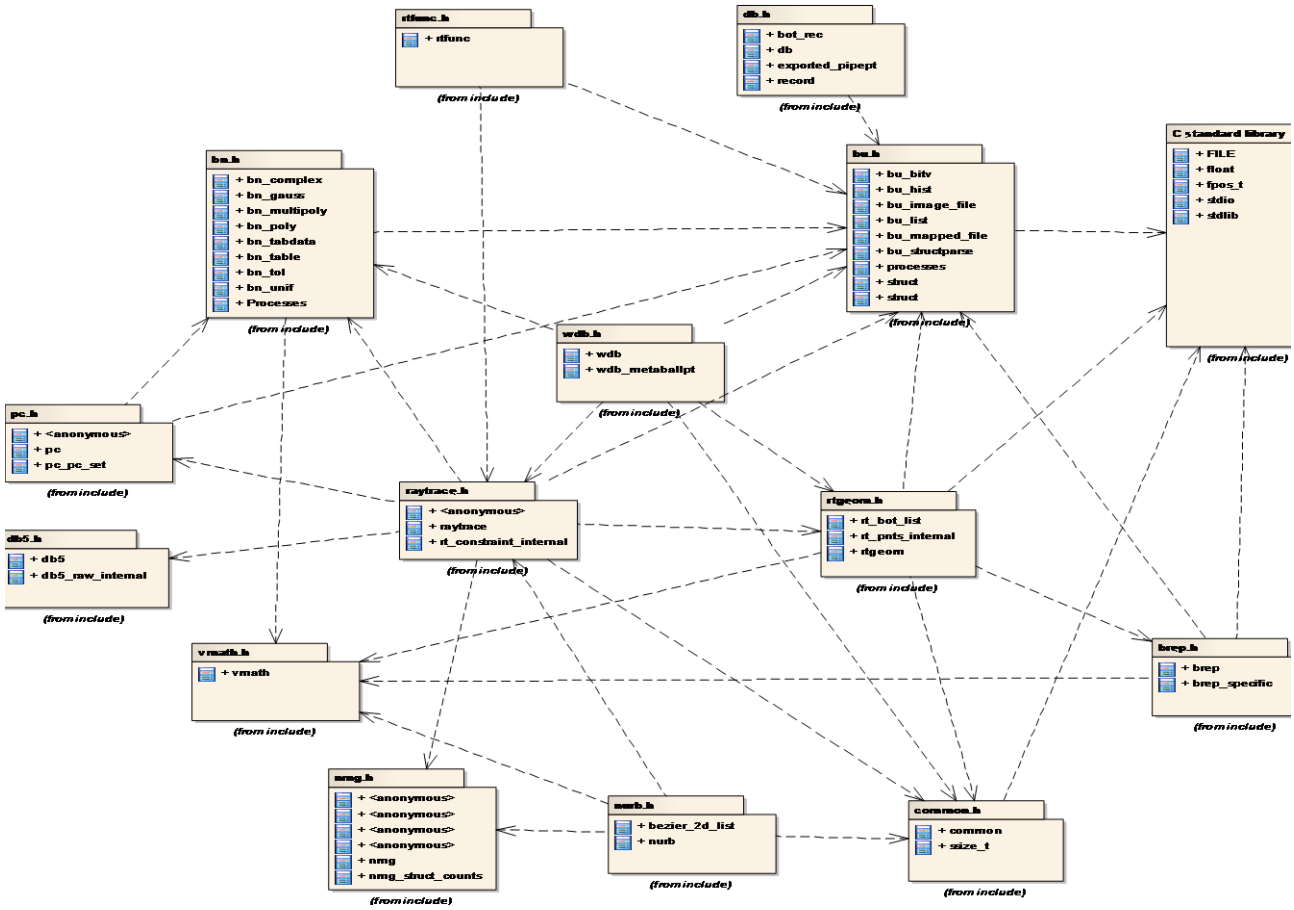

Fig. 12. **Full diagram of project packages**
***Source*: compiled by the author**

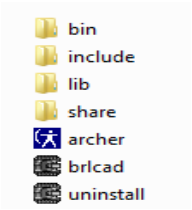

Fig. 13. **Physical location of the application**
***Source*: compiled by the author**

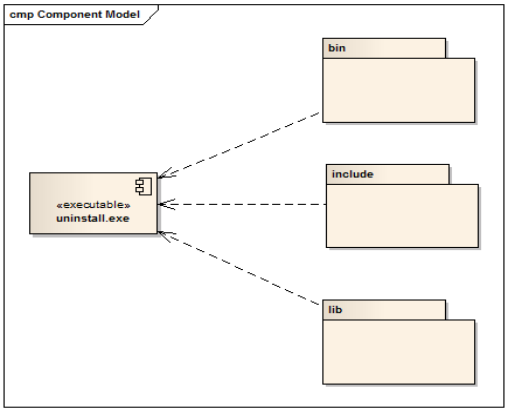

Fig. 14. **Primary structure**
***Source*: compiled by the author**

The next step in working on this diagram is to fill the packets with data. Let's start with the most important package – “include”, which contains executable files.

Let’s rebuild the structure inside the folder and bind the entire class diagram to it. Since all executable files are written in C language, that is, in one format, then we will not create a large number of components – we limit ourselves to one. We will name it the general name “Source code” and indicate in the specification that it is the language “C” and then we bind the class diagram, which contains the entire code of the component.

The overall sequence is:

a) in the project manager, choose a diagram containing the required classes;

b) find the component to which the diagram will be attached;

c) transfer the diagram to the batch component.

At the moment of the transfer, the following process takes place: the diagram itself goes into the component hierarchy, and all classes remain in the old place, in their subdivision of the diagrams (Fig. 15).

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

The convenience of such a structure is that the diagrams have been connected, that is, the transition to a class diagram can be done only through a dedicated component, but the classes themselves have not moved, which has greatly facilitated readability, and in the future, a change in the structure.

All major modules have been upgraded and only some of the details have to be completed, so the folder besides the code contains another folder with configuration files that directly affect the code. Therefore, we will create a folder called "config", fill it with the components of the corresponding file and continue to divide the links in the data packet (Fig. 16).

In the "config" folder, the files do not interact with each other, so internal connections are not required, but in general, each of these files interacts with the component "Source code", which means their connection. It remains only to determine the type of connection.

In any of the files in the "config" folder there is a set of numbers to which executable files are referred, that is, when changing the number, the code itself changes. For this kind of interaction, there is a special type of connection, called "dependency". Therefore, we will place it on the diagram, indicating that the "Source code" batch component depends on the "config" folder (Fig. 17).

In the same way, we implement other 3 folders on the diagram – "lib", "bin" and "share". However, there are some differences. In the "lib" folder, there is a huge amount of libraries, each of which can be represented as an independent component, but it can be done only manually and it takes a lot of time, so let's go by the path of least resistance – transferring all the libraries to the diagram as "artifacts".

Artifact is any artificially created element in the system. Artifacts can be any type of file, even elementary pads, quite apart from code files and libraries. In this case, it would be entirely justified to apply structuring exactly in this way, but even though such a move makes it easier to work, you need to specify a type, or as it is called in UML, a stereotype of the file (Fig. 18). Reproducing each artifact, we set its specification.

After playing the modules in the diagram, we set up the links between them. As it was seen from Fig. 13, the folder contains the "Uninstall.exe" file. If it is activated, then the uninstallation of the CAD occurs, that is, this file affects every other physical (and not only) location of the software system.

As a result, all other folders and files are associated with this component as a "dependency" connection. All we have to do is to link our main packages. Therefore, the "bin" package is a folder with the main and auxiliary "exe" files, and two other packages – "lib" and "include" (libraries and executable files with code) "realize" the "bin" package. For this, there is a special type of connection with the similar name "realize".

Now the packages are recreated, the links are also binded to the class diagram executed. The result of the work in general is shown in Fig. 19.

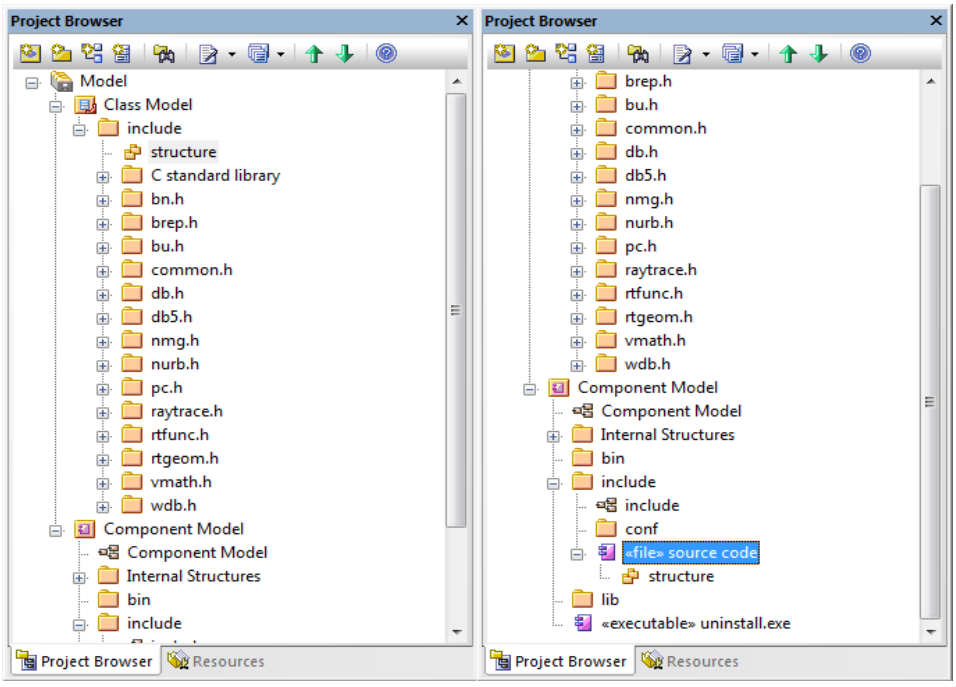

Fig. 15. **Changes in project manager**
***Source*: compiled by the author**

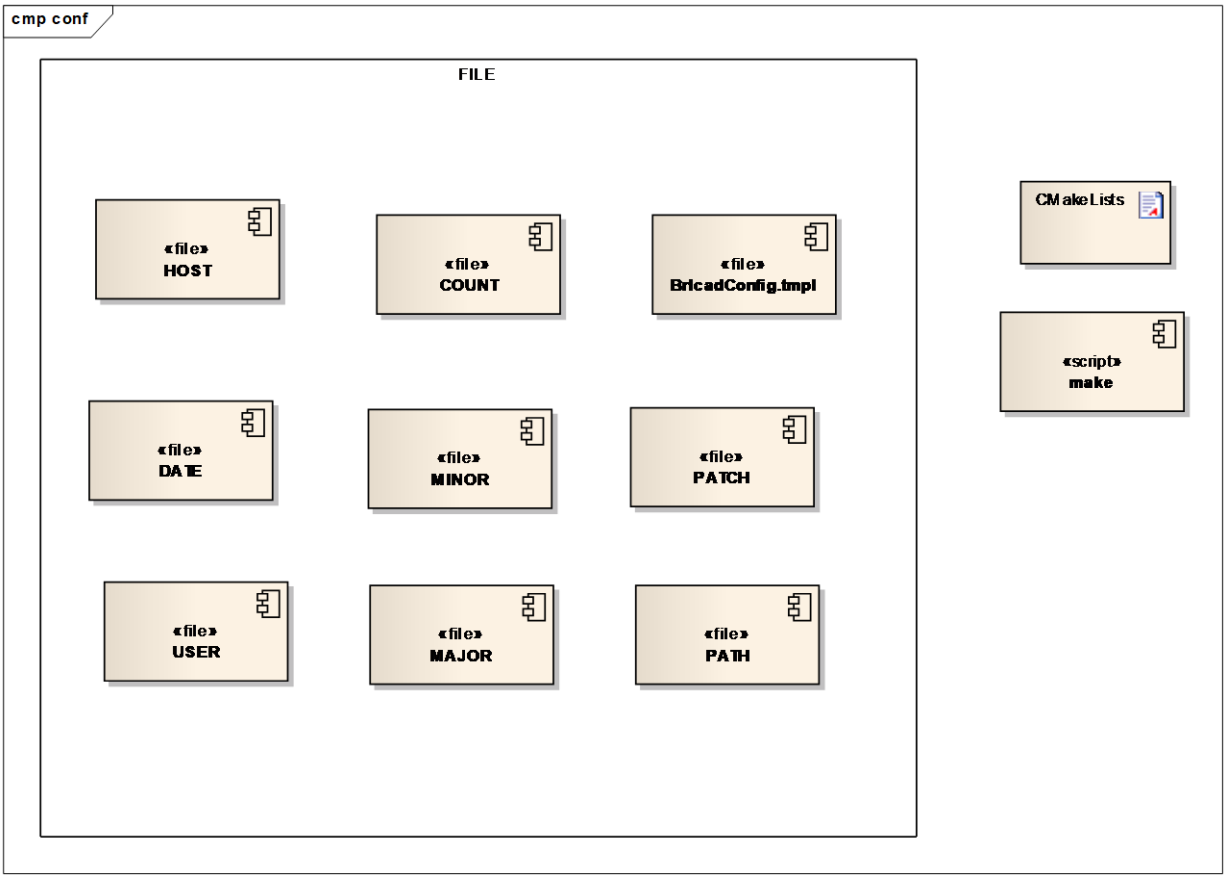

Fig. 16. **Presentation model of the "config" folder**
***Source*: compiled by the author**

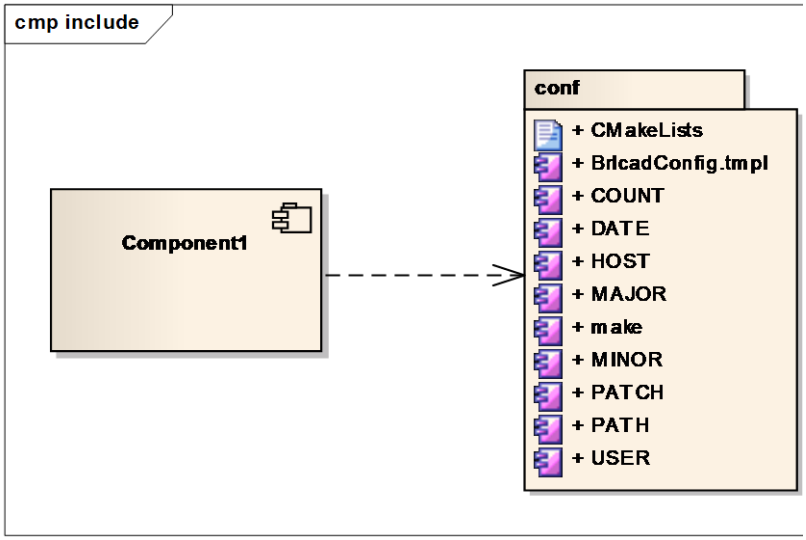

Fig. 17. **Interaction within the "include" folder**
***Source*: compiled by the author**

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

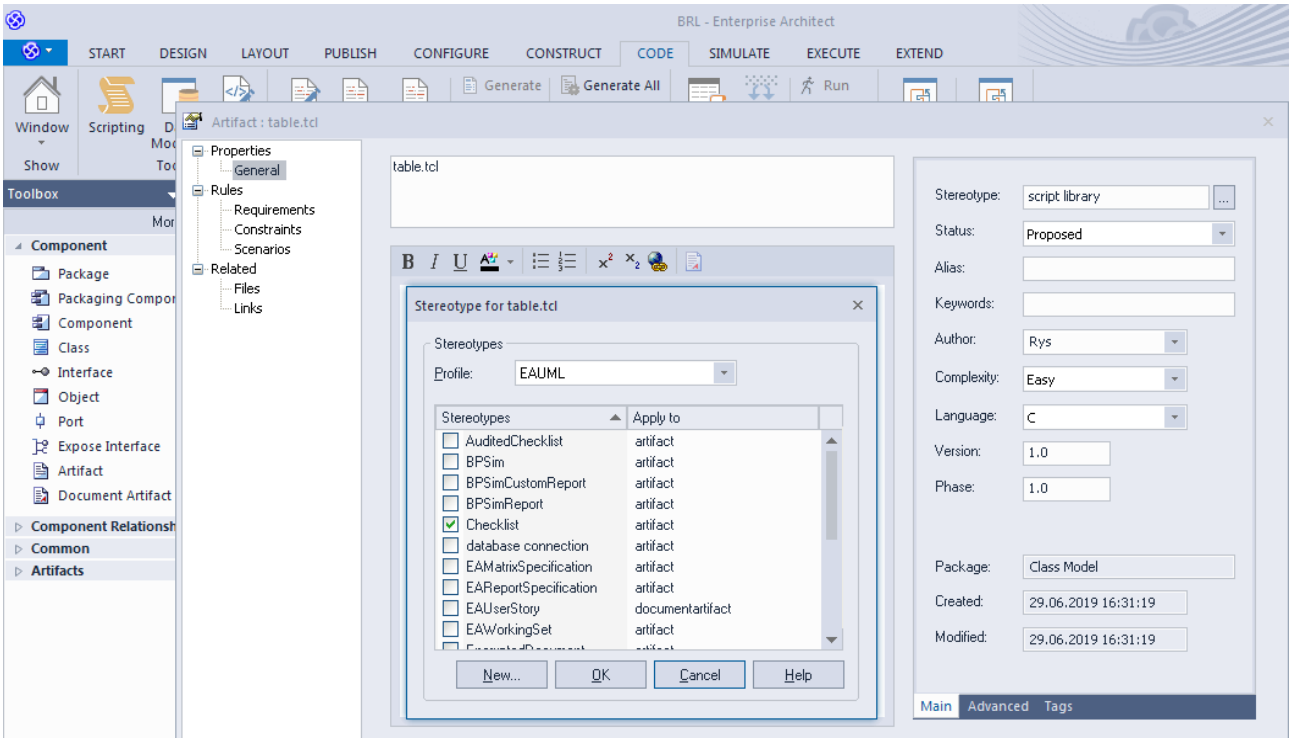

Fig. 18. **Stereotype and program profile selection**
***Source*: compiled by the author**

DISCUSSION

The present article summarizes the process of reverse engineering on the example of the open BRL-CAD. The study has been carried out using the UML methodology using Enterprise Architect CASE-tool. The UML methodology is quite voluminous and the project has considered several diagrams that are used to design a new product.

The main focus was on the class and component diagrams. This is due to the fact that the code generation and subsequent work of programmers will occur directly from these diagrams, while other auxiliary diagrams serve only to explain complex project specifications, which, however, does not deemphasize their significance in the project.

In the most progressive countries of the world, new products have not been developed "from scratch" for a long time, for them systems that help to create any necessary structure much more quickly and efficiently are used. The UML methodology and related software products serve to improve the design and structuring of data. This methodology has been actively used since recently, but very quickly integrated into the overall design structure. The convenience of the reengineering methodology is that it is not tied to any of the development methods and is very flexible in use.

The development of the UML methodology for reverse engineering is typical for the West and parts of Europe. At the beginning of the research (that is, in 2010), specialists from our country have just begun to work on the active exploitation of this methodology in the form in which it is presented now.

The open and free BRL-CAD has served as an excellent prototype for work. The advantage of such systems is that they are distributed under a free license and there are no legal issues with copying, modification or other actions related to the software code. Similarly, it should be noted that because the code was made open, the developers tried to make it also understandable. This is due to the large number of comments in the program code.

One of the major disadvantages of BRL-CAD is the lack of a clear graphical product interface, corrected and improved in the prospects for software product reengineering. However, developers and designers usually develop the interface from the middle of the project, or even closer to the end, when the full full functionality and working principles are already known.

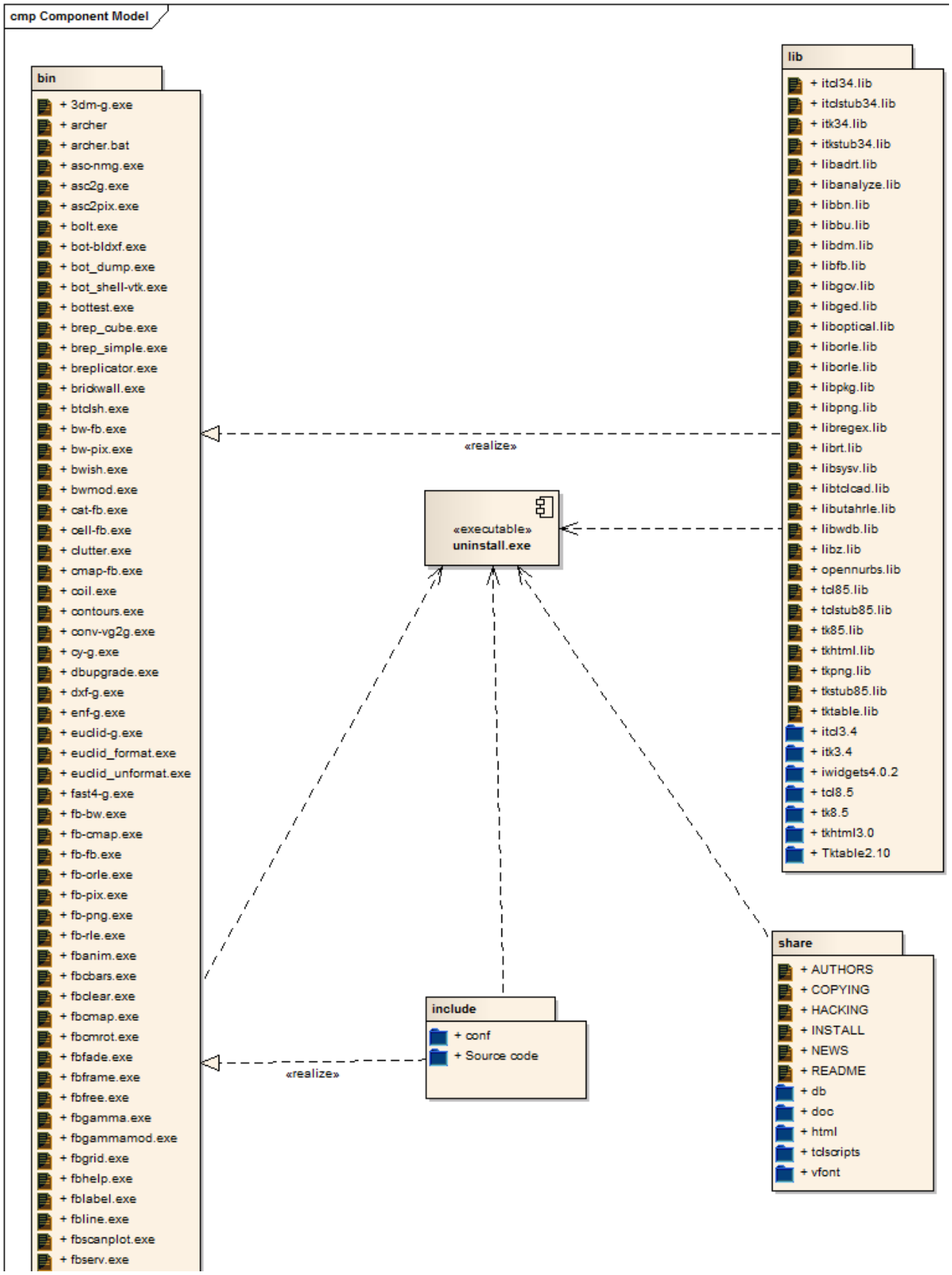

Fig. 19. **Complete diagram of BRL-CAD components**
***Source*: compiled by the author**

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

CONCLUSIONS

The article analyzes and summarizes the results of complex experimental research of new methods of multi-linguistic transcoding of open source software. The distinctive feature of the studies under review is the ability to support the work of more than ten most popular programming languages. When applying these technologies, it is possible to automate the process of recoding software components and, therefore, to free programmers from routine reprogramming and reduce the probability of occurrence of structural errors inherited from the previous system.

In the article, the technology of multi-linguistic transcoding has not been only given, but also analyzed. The problem of this field for our country is the lack of any educational materials; although this situation allows to use software products and build or restore code, but it hides some interesting and useful features from users. Therefore, the task was to consider and systematize the process and the logic of reverse engineering and to form the basis of a scientific paradigm, which allows the system architect to understand the principles of this reengineering.

Reengineering is a process that allows to create new, improved software systems quickly and easily, using the experience of previous software products. With the introduction of this methodology, we can conclude that the efficiency and speed of work has grown considerably also because of the fact that UML is a convenient language that coordinates the actions of all employees and helps to distribute tasks between performers.

From an economic point of view, reengineering is generally advantageous – it is a significant time and effort savings for programmers, it helps in project coordination, and optimizes the number of employees, although there are some situations where software reengineering is not the best solution. In any case, before performing the evolutionary improvement, it is necessary to evaluate the feasibility of a software project reengineering - this is also a series of publications by the author of this article.

At this time, the process of designing new software products is not very effective without the use of the UML methodology, but with its use – the speed of development increases at times.

Summing up the results, we can state that in the article:

1) Generalization of the results of experimental studies at the level of presentation of classes and components presented using a unified modeling language – UML, with processing and interpretation of results at CASE-tools level has been performed;

2) The results of source code conversions have been analyzed and summarized, the main of which is the reduction of labor productivity of the CAD creation;

3) The generation of new linguistic structures has been improved, based on reconstructed unified diagram models, which allows to preserve the properties of relations between classes and between components.

4) Methods of importing encapsulated components of the CAD that allows re-encoding the components regardless of the programming language have been developed further.

Using the results will significantly improve the efficiency of the CAD use in such fields of their use as: mechanic engineering, telecommunications, production and transport management, education, etc.

The developed models and methods will be useful to system architects and program engineers involved in redesigning software already being in their multi-year operation.

Properly executed reengineering is characterized by the achievement of practical results:

a) Reducing the risk of errors in the future update of the CAD;

b) Reduction of the product cost due to the repeated use of software components in the development of a new CAD;

c) Reduction of the labor productivity of the creation of the CAD due to the almost complete elimination of routine reprogramming operations of many already identified components.

Thus, the main problem of the study was the systematization of methods of reengineering software components into new software structures, systems and ready-made information resources accumulated by humanity at a specified time. This new direction does not yet have standard solutions to the problem of the gradual transformation of the multi-linguistic description, the implementation of generation, debugging, and integration for the final software system.

Therefore, the possible directions of the research continuation and the prospects for the development of the following studies on the following topics are:

1) Industrial testing for fault tolerance and practical testing of implementation of software systems multi-linguistic transcoding that will allow them to be improved on an industrial basis;

2) Creation of reengineering models for each other types of CAD provisioning that will be redesigned.

Complete reengineering of the CAD will overcome the contradiction between the pace of science and technology and design processes development, improve the efficiency of technical support of software systems, and reduce operating costs.

REFERENCES

1. Link, D., Behnam, P., Moazeni, R. & Boehm, B. "The Value of Software Architecture Recovery for Maintenance" (Submitted on 23 Jan 2019 in Cornell University). – Available from: https://arxiv.org/abs/1901.07700. – Active link – 27.06.2019.
2. Lavrishcheva, E. M., & Grishchenko, V. N. "Sborochnoe programmirovanie. Osnovy industrii programmnyh produktov". [Assembly programming. Fundamentals of the software industry] (in Russian). *Naukova dumka.* Kiev: Ukraine. 2009. 372 p.
3. Subriadi, A. P., Muqtadiroh, F. A. & Dewi, R. S. "A model of owner estimate cost for software development project in Indonesia". – Available from: https://onlinelibrary.wiley.com/ doi/abs/10.1002/smr.2175. – Active link – 27.06.2019.
4. Velykodniy, S. & Tymofieieva, O. "The paradigm of linguistic supply submission by generative grammar assistance". *American Scientific Journal.* 2017; No.17: 4–7.
5. Velykodniy, S. & Tymofieieva, O. "Reengineering Models of Linguistic Providing Software Systems". Advances in Quantum Systems in Chemistry, Physics and Mathematics, Ser.: Progress in Applied Mathematics and Quantum Optics, Eds. A. Glushkov, O. Khetselius, V. Buyadzhi. *FOP Panov.* Kharkiv: Ukraune. 2017. p. 385–388.
6. Velykodniy, S. S. "Problema reinzhiniringa vidov obespecheniya sistem avtomatizirovannogo proektirovaniya". [The reengineering problem of ensures types CAD/CAM/CAE-systems] (in Russian). *Control Systems and Computers.* 2014; No.1: 57–61. 76.
7. Velykodniy, S. S. "Metodo-logicheskie osnovy reinzhiniringa sistem avtoma-tizirovannogo proektirovaniya. [The method-logical bases of reengineering CAD/CAM/CAE-systems] (in Russian). *Control Systems and Computers.* 2014; No.2: 39–43.
8. Velykodniy, S. & Tymofieieva, O. "Multilingual recording method designed for SCADA-system's software upgrade". *Automation of technological and business-processes.* 2017; Vol.9, Iss.1: 17–22.
9. Velykodniy, S. & Tymofieieva, O. "Sposib multylinhvistychnoho perekoduvannia prohramnoho zabezpechennia skladnykh informatsiinykh system ta tekhnolohii". [The way of multilingual software transcoding for complex information systems and technologies] (in Ukrainian). O. S. Popov's ONAT Scientific Works. 2017; No.2: 153–159.
10. Velykodniy, S., Tymofieieva, O. & Zaitseva-Velykodna, S "Metod rozrakhunku pokaznykiv otsinky proektu pry vykonanni reinzhynirynhu prohramnykh system". [The calculation method for indicators project estimation in the implementation of software systems reengineering] (in Ukrainian). *Radio electronics, computer science, control*. 2018; No.4: 135–142. (Web of Science). DOI: https://doi.org/10.15588/1607-3274-2018-4-13.
11. Velykodniy, S. "Idealizovani modeli reinzhynirynhu prohramnykh system". [The idealized models of software systems reengineering] (in Ukrainian). *Radio electronics, computer science, control.* 2019; No.1: 150–156. (Web of Science). DOI: https://doi.org/10.15588/1607-3274-2019-1-14.
12. Velykodniy, S. "Method of presenting the assessment for reengineering of software systems with the project coefficients help". *Innovative technologies and scientific solutions for industries.* 2019; No.1(7): 34–42. DOI: https://doi.org/10.30837/2522-9818.2019.7.034.
13. Afshari, A. R., Brtka, V. & Ćoćkalo-Hronjec, M. "Project risk management in Iranian software projects". *Journal of Engineering Management and Competitiveness (JEMC).* 2018; Vol.8 No.2: 81–88.
14. Blum, B. "Software engineering: a holistic view". – Available from: https://dl.acm.org/citation.cfm?id=SERIES9569.128915. – Active link – 24.06.2019.
15. Klein, M. "Reengineering metho-dologies and tools. A Prescription for Enhancing Success". [Electronic resource]. – Access mode: https://www.tandfonline.com/doi/abs/ 10.1080/10580539408964633. – Active link – 23.06.2019. DOI: https://doi.org/10.1080/10580539408964633.
16. Boehm, B. "Software Risk Management". – Available from: https://link.springer.com/chapter/10.1007%2F3-540-51635-2_29. – Active link – 24.06.2019, DOI: https://doi.org/10.1007/3-540-51635-2_29.

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

17. Grover, V. & Malhotra, M. “Business process reengineering: A tutorial on the concept, evolution, method, technology and application”. – Available from: https://www.sciencedirect.com/science/article/abs/pii/S0272696396001040. – Active link – 23.06.2019. DOI: https://doi.org/10.1016/S0272-6963 (96)00104-0.

18. Manganelli, R. & Klein, M. “The Reengineering Handbook: A Step-by-Step Guide to Business Transformation”. [Electronic resource]. – Access mode: https://www.sciencedirect.com/science/article/pii/S00https://journals.lww.com/jhqonline/Citation/1995/03000/The_Reengineering_Handbook__A_Step_by_Step_Guide.11.aspx. – Active link – 26.06.2019. DOI: https://doi.org/10.1097/01445442-199503000-00011.

19. Jacobson, I., Ericsson, M. & Jacobson, A. “The Object Advantage: Business Process Reengineering with Object Technology”. *ACM Press*. – Available from: http://eaststemcell.com/files/storage.cloud.php?id=MDIwMTQyMjg5MQ==. – Active link – 20.06.2019.

20. Boehm, B. “Spiral Development: Experience, Principles and Refinements”. Special Report: CMU / SEI-2000-SR-008. 2000. 37 p.

21. Hammer, M. & Champy, J. “Reengineering the corporation: A manifesto for business revolution”. [Electronic resource]. – Access mode: https://www.sciencedirect.com/science/article/pii/S0007681305800643?via%3Dihub. – Active link – 21.06.2019. DOI: https://doi.org/10.1016/S0007-6813(05)80064-3.

22. Selby, R. W. “Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management and Research”. *Publ. John Wiley & Sons*. New Jersey. 2017. 818 p.

23. Boehm, B. “A Spiral Model of Software Development and Enhancement”. ACM SIGSOFT Software Engineering Notes. 1986; Vol.11 Iss.4:14–24. DOI: https://doi.org/ 10.1145/12944.12948.

24. Nevlyudov, I. Sh., Velykodniy, S. S. & Omarov, M. A. “Ispol'zovanie CAD/CAM/CAE/CAPP pri formirovanii upravlyay-ushchikh programm dlya stankov s ChPU”. [Using CAD / CAM / CAE / CAPP when forming control programs for CNC machines] (in Russian). *Eastern-European Journal of Enterprise Technologies*. 2010; Vol.2 Issue 2(44): 37–44.

25. “Unigraphics Direct Interface: Reference Manual”. *Southampton: ICEM Ltd*. 2014. 392 p.

26. Werner, J. “The Case for Verifying and Optimizing Tool Paths”. Irvine: *CGTech.* 2003. 5 p.

27. Velykodniy, S. S., Tymofieieva, O. S., Zaitseva-Velykodna, S. S., & Niamtsu, K. Ie. Porivnyalniy analiz vlastivostey vidkritogo, vilnogo ta komertsiynogo programnogo zabezpechennya. [A comparative analysis of the properties of open, free and commercial software] (in Ukrainian). *Information Technology and Computer Engineering* 2018; No.1(41): 21–27.

28. Velykodniy, S. S., Burlachenko, Zh. V. & Zaitseva-Velykodna, S. S. “Reinzhyniryng grafichnyh baz danyh u seredovyshhi vidkrytoi' systemy avtomatyzovanogo proektuvannja BRL-CAD. Modeljuvannja povedinkovoi' chastyny”. [Graphic data-bases reengineering in BRL-CAD open source computer-aided design environment. Modeling of the behavior part] (in Ukrainian). Transactions of Kremenchuk Mykhailo Ostrohradskyi National University 2019; No.2(115): 117–126. DOI: https://doi.org/10.30929/1995-0519.2019.2.117-126.

29. Miles, R. & Hamilton, K. “Learning UML 2.0”. *O'Reilly Media.* 2006. 288 p.

30. Hay, D. C. “UML and Data Mode-ling: A Reconciliation Technics Publications”. 2011.242 p.

31. Haigh, T. “How Data Got its Base: Information Storage Software in the 1950s and 1960s”. *IEEE Annals of the History of Computing.* 2010; Vol.31 Iss. 4: 6–25. DOI: https://doi.org/ 10.1109/MAHC.2009.123.

32. Date, C. J. “Date on Database: Writings 2000 – 2006”. *Apress.* New York City: 2006. 566 p.

33. Velykodniy, S. S., Burlachenko, Zh. V. & Zaitseva-Velykodna, S. S. “Reinzhyniryng grafichnyh baz danyh u seredovyshhi vidkrytoi' systemy avtomatyzovanogo proektuvannja BRL-CAD. Modeljuvannja strukturnoi chastyny”. [Graphic data-bases reengineering in BRL-CAD open source computer-aided design environment. Modeling of the structural part] (in Ukrainian). Transactions of Kremenchuk Mykhailo Ostrohradskyi National University. 2019; No. 3(116): 130–139. DOI: https://doi.org/10.30929/1995-0519.2019.3.130-139.

34. Carroll, E. R. “Estimating Software Based on Use Case Point”. OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, San Diego: CA. 2005. p. 257–265. DOI: https://doi.org/10.1145/1094855.1094960.

35. Cohn, M. “Agile Estimating and Planning”, *Publ. Prentice Hall.* NY. 2005. 368 p.

36. Clemmons, R. “Project Estimation with Use Case Points”. *Cross Talk.* 2016; Vol.2 Iss.: 18–22.

37. Kalnauz, D., & Speranskiy, V. “Productivity estimation of serverless computing”. *Applied Aspects of Information Technology*. 2019; Vol.2 No.1: 20–28. DOI: https://doi.org/10.15276/aait.02.2019.2.

38. Velykodniy, S. S., Burlachenko, Zh. V. & Zaitseva-Velykodna, S. S. “Software for automated design of network graphics of software systems reengineering”. *Herald of Advanced Information Technology*. 2019; Vol.2 No.2: 95–107. DOI: https://doi.org/10.15276/hait.02.2019.2.

39. Boggs, W. & Boggs, M. “UML & Rational Rose”. *Lori.* SPb.: Russian Federation. 2008. 600 p.

40. Fauler, M. “UML. Osnovy. Kratkoe rukovodstvo po standartnomu yazyku ob’ektnogo modelirovaniya” [UML. Basics. A quick guide to the standard object modeling language]. *Simvol-Plyus.* Moscow: Russian Federation. 2011. 192 p.

41. Robinson, S. Kornz, O. & Glinn, J. “C# dlya professionalov”. [C# for professionals]. *Lori.* Moscow: Russian Federation. 2005. 1000 p.

42. “Object Management Group. OMG Unified Modeling Language (OMG UML). Version 2.5”. *Object Management Group.* 2013. 831 p.

Conflicts of Interest: the authors declare no conflict of interest

Received 26.04.2019
Received after revision 03.06.2019
Accepted 18.06.2019

DOI: https://doi.org/10.15276/aait.03.2019.2
УДК 004.4'2

АНАЛІЗ ТА УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ КОМПЛЕКСНИХ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ З РЕІНЖИНІРИНГУ ВІДКРИТИХ СИСТЕМ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ

Станіслав Сергійович Великодний
ORCID ID: https://orcid.org/0000-0001-8590-7610, velykodniy@gmail.com
Одеський державний екологічний університет, вул. Львівська, 15. Одеса, 65016, Україна

АНОТАЦІЯ

У статті подаються заключні результати наукового дослідження з розроблення моделей та методів реінжинірингу, а також технологій мультилінгвістичного перекодування відкритих систем автоматизованого проектування. Спільної рисою для усіх програмних систем є те, що під впливом часу та інших невід’ємних факторів інформатизації, а саме оновлення: операційних систем, мов програмування, принципів дії розподілених систем обробки даних тощо, відбувається еволюційне старіння видів забезпечення. Така тенденція призводе до погіршення швидкісних, інформаційно-комунікаційних, графічних, часових та інших характеристик, аж до повної відмови системи. Реінжиніринг – це процес, який дозволяє зручно й швидко створювати нові, удосконалені програмні системи, використовуючи досвід попередніх програмних продуктів. Мета статті – систематизувати результати інтеграції компонентів повторного використання, що накопичено розробниками за визначений час розвитку галузевих систем автоматизованого проектування у оновлені програмні структури готових ресурсів. За отриманими науковими та практичними результатами виконується аналіз розроблених моделей та методів реінжинірингу видів забезпечення відкритих систем автоматизованого проектування. Загалом, реінжиніринг містить у собі процеси реорганізації та реструктуризації програмної системи, переведення окремих компонентів системи в іншу, сучаснішу мову програмування, а також процеси модифікації або модернізації структури і системи даних. В досліджені задіяні наступні методи: складального, конкретизуючого, синтезуючого та композиційного програмування, методи породжувальних й розпізнавальних граматик. На цей час, процес проектування нових програмних продуктів є не надто ефективним без використання UML-методології, але при її застосуванні швидкість розробки підвищується у рази. UML, як мова графічного опису для об'єктного моделювання, окрім простого проектування, підтримує ще й функцію генерації та реінжинірингу коду на основі даних моделей, саме які розглянуто у поданій статті. Відмінною особливістю наведених досліджень є можливість підтримки роботи більше десяти найпопулярніших мов програмування. При застосуванні наведених технологій вдається автоматизувати процес перекодування компонентів програмного забезпечення та, за рахунок цього, вивільнити робочий час програмістів від рутинного перепрограмування і зменшити вірогідність виникнення структурних помилок, що успадковуються від попередньої системи. Використання отриманих результатів надасть значне підвищення ефективності застосування систем автоматизованого проектування у таких галузях їх використання як: машинобудування, сфера телекомунікацій, управління виробництвом та транспортом, освіта тощо. Розроблені моделі та методи стануть у нагоді системним архітекторам та інженерам-програмістам, які задіяні у перепроектуванні програмного забезпечення, що вже знаходяться у кількарічній експлуатації.

Ключові слова: реінжиніринг систем автоматизованого проектування; методологія UML; багатомовне транс кодування; лінгвістична структура; породжувальна граматика

DOI: https://doi.org/10.15276/aait.03.2019.2
УДК 004.4'2

АНАЛИЗ И ОБОБЩЕНИЕ РЕЗУЛЬТАТОВ КОМПЛЕКСНЫХ ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ ПО РЕИНЖИНИРИНГУ ОТКРЫТЫХ СИСТЕМ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

Станислав Сергеевич Великодный
ORCID ID: https://orcid.org/0000-0001-8590-7610, velykodniy@gmail.com
Одесский государственный экологический университет ул. Львовская, 15.Одесса, 65016, Украина

АННОТАЦИЯ

В статье представлены заключительные результаты научного исследования по разработке моделей и методов реинжиниринга, а также технологий мультилингвистического перекодирования открытых систем автоматизированного проектирования. Общей чертой для всех программных систем является то, что под влиянием времени и других неотъемлемых факторов информатизации, а именно обновление: операционных систем, языков программирования, принципов действия распределенных систем обработки данных и др. происходит эволюционное старение видов обеспечения. Такая тенденция приводит к ухудшению скоростных, информационно-коммуникационных, графических, временных и других характеристик, вплоть до полного отказа системы. Реинжиниринг – это процесс, который позволяет удобно и быстро создавать новые, усовершенствованные программные системы, используя опыт предыдущих программных продуктов. Цель статьи – систематизировать результаты интеграции компонентов повторного использования, накопленных разработчиками за определенное время развития отраслевых систем автоматизированного проектирования в обновлённые программные структуры готовых ресурсов. По полученным научным и практическим результатам выполняется анализ разработанных моделей и методов реинжиниринга видов обеспечения открытых систем автоматизированного проектирования. В целом, реинжиниринг включает в себя процессы реорганизации и реструктуризации программной системы, перевод отдельных компонентов системы на другой, более современный язык программирования, а также процессы модификации или модернизации структуры и системы данных. В исследовании задействованы следующие методы: сборочного, конкретизирующего, синтезирующего и композиционного программирования, методы порождающих и распознающих грамматик. В настоящее время, процесс проектирования новых программных продуктов не слишком эффективен без использования UML-методологии, однако при ее применении скорость разработки повышается в разы. UML, как язык графического описания для объектного моделирования, кроме простого проектирования, поддерживает еще функции генерации и реинжиниринга кода на основе рассмотренных в данной статье моделей. Отличительной особенностью приведенных исследований, является возможность поддержки работы десяти самых популярных языков программирования. При применении указанных технологий удается автоматизировать процесс перекодирования компонентов программного обеспечения, и за счет этого, высвободить рабочее время программистов от рутинного перепрограммирования, а также уменьшить вероятность возникновения структурных ошибок, которые наследуются от предыдущей системы. Использование полученных результатов позволит значительно повысить эффективность применения систем автоматизированного проектирования в таких областях их использования как: машиностроение, сфера телекоммуникаций, управления производством и транспортом, образование и др. Разработанные модели и методы пригодятся системным архитекторам и инженерам-программистам, которые задействованы в перепроектировании программного обеспечения, находящегося в многолетней эксплуатации.

Ключевые слова: реинжиниринг систем автоматизированного проектирования; методология UML; многоязычное транскодирование; лингвистическая структура; порождающая грамматика

ABOUT THE AUTHORS

Stanislav Sergeyevich Velykodniy, Dr. Sci. (Eng), Associate Professor of the Department of Information Technology, Odessa State Environmental University, 15 Lvivska Str. Odesa, 65016, Ukraine
velykodniy@gmail.com. ORCID ID: https://orcid.org/0000-0001-8590-7610.
Research field: Software Design, Software Analysis and Testing, Software Project Management

Станіслав Сергійович Великодний, доктор техніч. наук, доцент кафедри Інформаційних технологій, Одеський державний екологічний університет, вул. Львівська, 15. Одеса, 65016, Україна