DOI: https://doi.org/10.15276 /aait.04.2021.4
UDC 004.4'24

Automated object-oriented technology for software module development

Oleksii B. Kungurtsev[1)]
ORCID: http://orcid.org/0000-0002-3207-7315; akungurtsev19@gmail.com. Scopus Author ID: 57188743440
Nataliia O. Novikova[2)]
ORCID: http://orcid.org/0000-0002-6257-9703; nataliya.novikova.31@gmail.com. Scopus Author ID: 57212034123
Svitlana L. Zinovatna[1)]
ORCID: http://orcid.org/0000-0002-9190-6486; zinovatnaya.svetlana@opu.ua. Scopus Author ID: 57219779480
Nataliia O. Komleva[1)]
ORCID: http://orcid.org/0000-0001-9627-8530; komleva@opu.ua. Scopus Author ID: 57191858904
[1)] Odessa National Polytechnic University, 1, Shevchenko Ave. Odessa, 65044, Ukraine
[2)] Odessa National Maritime University, 34, Mechnykov Str. Odessa, 65029, Ukraine

ABSTRACT

It is shown that most technologies for creating information systems are based on an object-oriented approach and provide for the presentation of functional requirements in the form of use cases. However, there is no general agreement on the format of the use cases and the rules for describing script items. The work has improved the classification of items of use cases basing on the analysis of a great number of existing descriptions from different subject areas. New rules have been introduced and the existing rules have been clarified for describing use cases, which made it possible to further formalize and automate the process of describing use cases. It is also proposed to automate the process of forming a model of program classes by introducing additional information linking the class with use cases. Thus, the programming class model contains significantly more information for coding than the existing models in UML diagrams. A method for constructing a model of program classes has been developed. Methods for the automated description of use cases and the construction of a model of program classes are linked into a single process. The level of information richness of the class model also makes it possible to automate the debugging process associated with changing requirements. Since the decisions made cover most of the steps in the software module creation process, they collectively represent a new technology. The proposed model, methods and technology were implemented in the ModelEditor and UseCaseEditor software products. Approbation of the method for automating the description of use cases demonstrated a decrease in the number of errors compared to the traditional method of describing more than two times, and shortening the time – more than one and a half times. Testing the method for constructing a model of program classes showed its advantage over the existing technology: errors and time reduction – almost one and a half times. The proposed technology can be used in the development of any information systems.

Keywords: Use case; model of program classes; information technology; object-oriented technology

***For citation*: Kungurtsev O. B., Novikova N. O., Zinovatna S. L., Komleva N. O. Automated object-oriented for software module development. *Applied Aspects of Information Technology*. 2021; Vol. 4 No. 4: 338–353. DOI: https://doi.org/10.15276/aait.04.2021.4**

1. INTRODUCTION

Most technology for creating information systems (IS) are based on an object-oriented (OO) approach. The object-oriented approach involves a series of sequential steps by the developer. Fig. 1 gives a generalized representation of the main stages of a typical OO-technology for creating an IS program module in the form of a set of activities: presentation of requirements in the form of use cases (UC), drawing up a model of conceptual classes, building interaction diagrams; creation of the specification of program classes, coding and testing.

The developers of the software product in the manual mode perform almost all mentioned activities and provide information communication between them. This is confirmed by an analysis of the literature.

© Kungurtsev O., Novikova N., Zinovatna S., Komleva N., 2021

In [1] and [2], a number of rules for describing UC are proposed, the high labor intensity of the process is noted, but there are no proposals for its reduction. References [3] and [4] provide guidelines for the use of UML diagrams, but do not consider the relationship of the said activity with the previous and subsequent design steps. In [5], the issues of testing based on UC are also considered in the form of a separate process. In [6], the application of OO-technology in the development of software products is considered, but exclusively in relation to mobile applications. In [7], the advantages and disadvantages of using UML diagrams are described, but only with respect to the specifics of reengineering existing systems in accordance with the evolution of software types. An assessment of the existing software development technology is given in [8], where as one of its main disadvantages it is indicated that all its processes are based solely on the human factor.

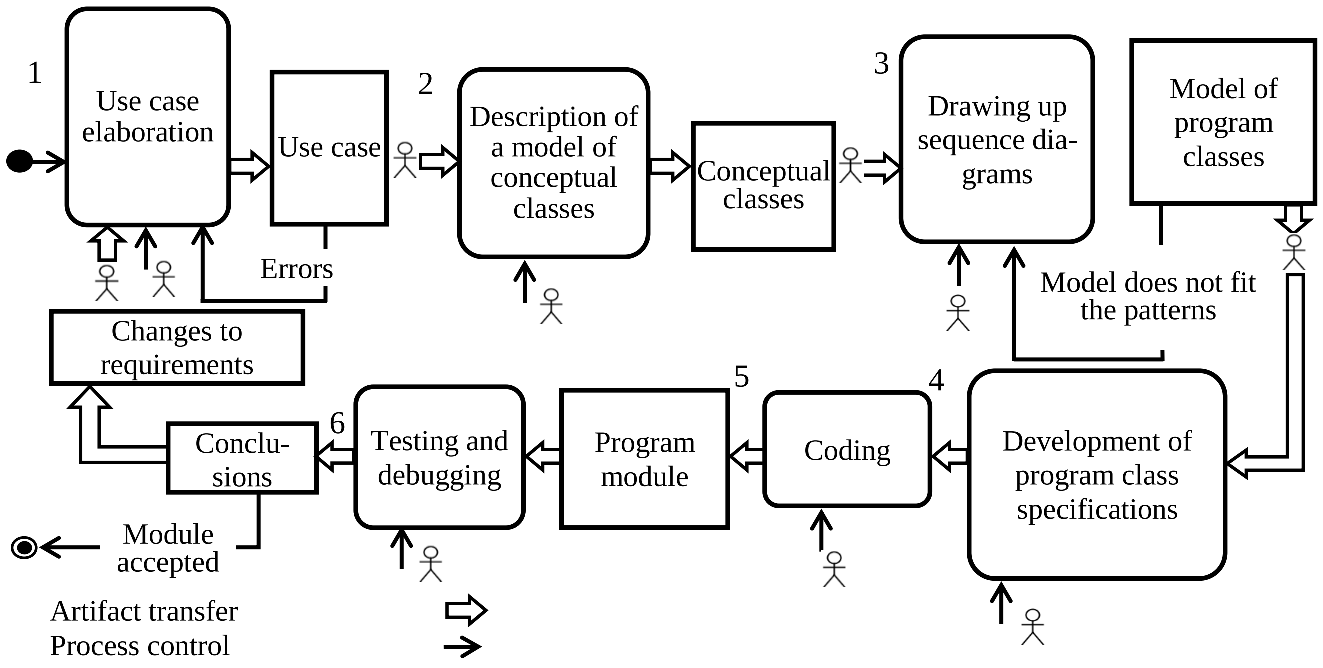


Fig. 1. **Typical object-oriented technology for software module development**
Source: **compiled by the authors**

Thus, increasing the productivity of existing technology by automating individual processes and exchanging information between them is a task of current interest.

The aim of the study is to reduce the time and the number of errors at the stages of forming requirements and creating class models by combining separate fragments of creating a model of program classes in a unified technology.

To achieve the goal, the following tasks should be solved:

– to clarify the rules for describing UC and improve the classification of items in the scenario of precedents;

– to change the method of automated UC generation taking into account changes in the classification;

– to improve the method of forming a model of program classes (MPC) by making changes to the class model;

– to develop new technology for development of a program module;

– to carry out approbation of the adopted decisions.

2. REVIEW OF THE LITERATURE AND PROBLEM STATEMENT

Drawing up UC is quite a laborious process. In [1], the following data on the time spent on compiling the UC are given: a group of 10 people produced 140 short descriptions of the UC per week (2.8 descriptions per person per day). In [2], automation tools are presented for development based on use cases taking into account frequently used artifacts, which is expected to reduce the overheads and complexity of the simulation. However, automation is limited to the use case diagram when working with product lines.

Most IT specialists believe that a UC description should contain three main parts: a preamble, a main successful scenario, and alternative scenarios. However, there is a great divergence of opinions on the content and formats of UC presentation. For instance the popularizes of the Rational Unified Process [9] propose to include 9 positions in the preamble. And in work [3] it is recommended to reduce their number to 4, to exclude the concept of a trigger, but also to introduce the concept of the target of UC. In [10], it is also proposed to have 4 positions in the preamble, but instead of the actor, a trigger is introduced.

There is no prevailing opinion regarding the format of the script items. In work [1] the following format is proposed: one column of the text (not tables); numbered steps; no sentences with “if”; the numbering convention in the extensions section, including combinations of numbers and letters. In work [5] it is proposed to write a scenario in the form of a table with three columns, in work [10] – as a table with four columns.

There is no consensus on the format and location of alternative scenarios. In work [1] it is proposed to move alternative scenarios outside the main scenario, and in work [5] it is proposed to divide an alternative scenario into two streams: an alternative stream and a stream that leads to unsuccessful completion.

Thus, a large number of contradictory rules for compiling UC and lack of classification of their items do not allow automating the process of their formation. It remains extremely labor intensive.

The first stage of IC design usually begins with the separation of conceptual classes. In [11], options are considered for defining conceptual classes using a list of categories of conceptual classes and by highlighting nouns. In [12], it was proposed to use the terms of the subject area for this purpose. In [13], it is proposed to distinguish conceptual classes basing on the analysis of each UC separately.

Regardless of how the concept class is allocated, it is usually represented as follows:

$$c = <cName, mCAttr'>, \quad (1)$$

where: *cName* is class name; *mCAttr'* is a subset of the class attributes needed to understand its use; attribute types are usually not specified.

The conceptual class model is a document linked with the subsequent stages of work on the project only through the developer (Fig. 1).

Usually, the model of a program class is understood as its specification, which can be represented as:

$$classSpecif = <cName, mCAttr, mcFunc>, \quad (2)$$

where: *mCAttr* is a list of class attributes with indication of types; *mcFunc* is a list of class methods with arguments and return type.

To obtain the specification, UML tools are usually used [14] in the form of sequence diagrams [15]. However, the authors do not propose to establish a direct connection between the processes of building interaction diagrams and creating class specifications. Also, when constructing interaction diagrams and class specifications, design patterns are used [16]. In [17], it is also not proposed to check for compliance the results obtained with design patterns, or it is not indicated at all at which design stages the patterns are used. Thus, the communication of the various design stages is realized through the developer. In [18], on the basis of the publications, it is shown that UML diagrams are often used while designing or modeling, however, more often in the description, class diagrams are already used, and sequence and state diagrams had a low frequency of use, not to mention the description of UC. In [19], a study was carried out, which also showed that developers do not use most of the UML diagrams at all.

On the basis of the analysis of literature data, the following conclusions can be drawn:

– there are no generally accepted rules for describing UC, this process is performed manually, is laborious, and associated with a large number of errors;

– the process of constructing class models is divided into several stages, the models are not informative enough, it is difficult to trace the mapping of requirements in classes and make changes to the requirements;

– in the existing technology for developing a software module the stages are connected through the developer, which determines a significant proportion of manual labor, a large dependence of the results on the qualifications and experience of the developer.

3. RESULTS OF DEVELOPMENT OF TECHNOLOGY FOR CREATION OF THE SOFTWARE MODULE

3.1. Clarification of the rules for generating use cases and classification of the scenario items

A use case is drawn up by a systems analyst (SA) after working with an expert in a specific domain. Systems analyst lays down in UC the possibilities of the future software product, therefore, each point of the scenario must be analyzed in terms of the possibility and method of its implementation.

It is proposed in each point of the scenario to indicate all the actions that the system must perform in accordance with the considered business process and all the data that are used in this case.

Example

A. Traditional revision of the UC script item.

n. The client contributes a certain amount. The cashier fixes the amount in the system. The system calculates the rest of money and generates a ticket.

B. Recommended edition.

n. The client contributes a certain amount. The cashier fixes the amount in the system. The system calculates the rest of money, marks the sold seat, and generates a ticket. It contains the departure date, the train number and class, the carriage number and class, the seat number. When transferring for the second train, the number of the train, and its class, the number of the carriage and its class, the number of the seat, and the transfer station are indicated. The transaction is recorded (transaction attributes will be defined later).

It is proposed to perform only one validation of the input data at each point in the scenario.

Example

A. Traditional revision of the UC scenario item.

n. The student reports their specialty and group. The secretary enters the received data into the system. The system confirms it.

B. Recommended edition.
n. The student reports their specialty. The secretary enters the system. The system confirms the presence of the specialty.
n + 1. The student reports their group. The secretary enters the system. The system confirms.

If the scenario item provides for the creation of an object, then you should indicate what it is created for.

Example
A. Traditional revision of the UC scenario item.
n. The client contacts the cashier about buying a ticket. The cashier creates a "new sale" in the system.
B. Recommended edition.
n. The client contacts the cashier about buying a ticket. The cashier creates a "new sale" in the system to store all information about the ticket.

"Visible" and "Invisible" parts of the UC description. It is proposed to make a part of the UC description invisible for the subject area expert, since this part will not relate to the requirements for a software product, but to its design.

It is proposed in all paragraphs where some data are discussed, to indicate in the invisible part the storage location (the invisible part of the description is located after the symbols "//").

Example
n. The client reports the name of the departure and destination stations. The cashier enters them into the system. The system confirms the presence of stations // Station departSt and station arrivSt are stored in the NewSale class.

It is impossible to automate the process of describing the UC without introducing a classification of a scenario points. In [20], based on the experience of working with UCs, a classification of scenario items was proposed. To improve and substantiate this classification, 37 examples of UC descriptions in [21], more than 10 descriptions in [5, 11], descriptions from [22], as well as the results of solving problems within the educational process, formulated in [23], were analyzed. The hierarchical classification method was chosen [24, 25] due to its simplicity, high information saturation and visibility [24]. The qualification characteristics were determined by the procedures for communicating the user with the system. The result is shown in Fig. 2.

At the second level of the hierarchy, there are three groups of items: "Input", "Command" and "Request". Group "Input" contains items "Data input" and "Select from the list". "Data entry" provides for the entry of one or more data into the system. "Select from the list" provides for the input of data corresponding to some items from the list. The list should have been displayed earlier. The "Command" group includes five items. The item "Repeat of actions" defines a repeating group of items in the scenario. The "UC call" item gives a link to another UC, to which control is transferred. Items "Successful completion of the UC" and "Unsuccessful completion of the UC" provide for certain sets of actions in appropriate situations. The item "Create" creates a model of the program class. The "Request" group includes four items. "Request with value input" provides for receiving data depending on the entered value. "Service request" involves the selection of one of the alternative scenarios. "Request for a list of values" further provides for selection from this list. A "Value request" requires a single value to be logged out.

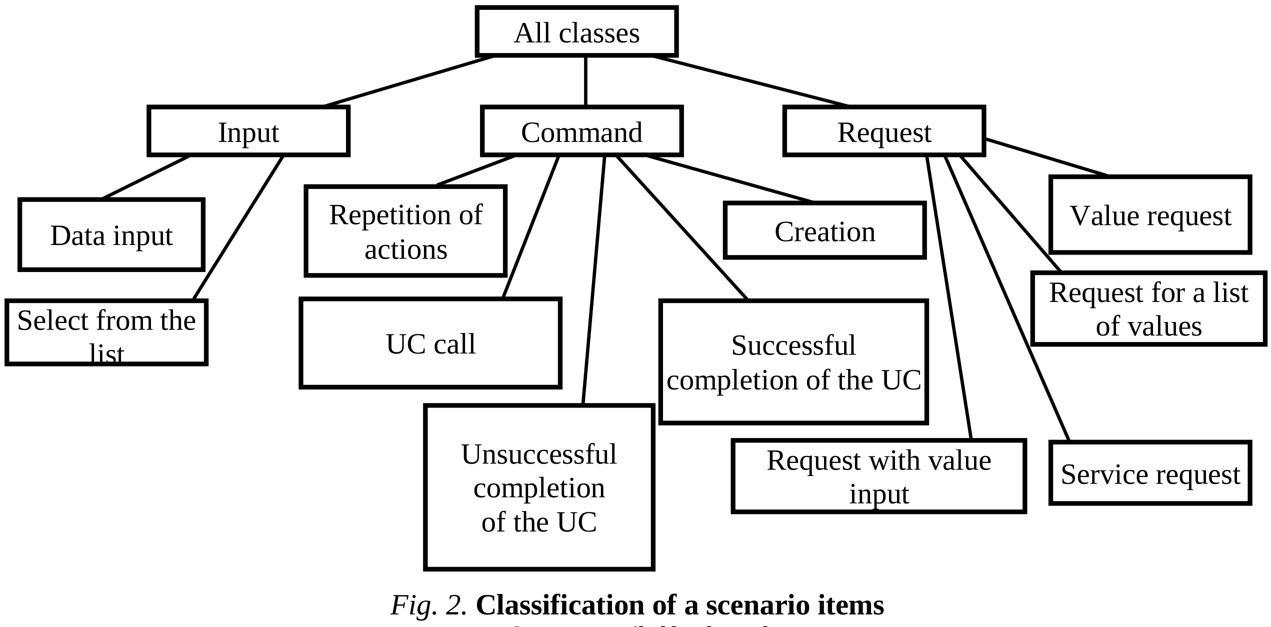


Fig. 2. **Classification of a scenario items**
Source: **compiled by the authors**

3.2. Method of automated generation of use cases

To describe the UC scenarios, a number of rules were formulated, which individually are not new, but together they allow automating the process of compiling UC scenario items:

– UC appears to be a major success scenario and extensions;

– the initiator of the action of each item of the main successful scenario is always the user;

– at each point in the scenario, the imaginary system must perform certain actions.

The method of automated generation of use cases involves the sequential implementation of three stages.

At the first stage method is proposed to represent the UC in the form:

$$u = <uName, pr, sc, mc>, \qquad (3)$$

where: *uName* – UC name; *pr* – preamble; *sc* – UC scenarios; *mc* – set of classes which implement UC.

The preamble includes: *ap* – the main actor; *da* – interests of the participants; *pc* – preconditions for performing UC; *gm* – minimum guarantees; *gs* – guarantees of success.

As a result of the first stage, the UC gets a name, and all the elements of the preamble are determined.

At the second stage of the method, the items of the main scenario are formed. An item model is a sequence of elements, each of which can be:

– automatically generated data (scenario item number – *nP*),

– designation of a certain person acting within the scenario (*Client*, *Actor*);

– pre-prepared piece of the text – *tpi*;

– text fragment formed in the process of composing a item – *tuj*;

– a type value that can be entered into the system, or obtained from it;

The model also uses metacharacters. Items in square brackets – "[...]" are optional. Items enclosed in parentheses "(...)" can be repeated. Items enclosed in curly braces "{...}" belong to the design phase, so they are not included in the software requirements documents. The "+" symbol denotes string concatenation. The "/" symbol means the use of one of two elements separated by this symbol.

As an example, the model of the item "Enter data" is shown.

$$inputData = <nP, [Client, tp1, tu1,] \; Actor, tp2, (tu2, \{data1 \; [tp3]\})>, \qquad (4)$$

where: *nP* – UC point number; *tp*1 = "reports"; *tu*1– user-generated text, such as an address; *tp*2 = "enters the system"; *tu*2 – user-generated text depends on tu1; *data*1 = {d1, d2, ..., dn) – list of input data; *tp*3= "The system confirms the correctness of the data".

As a result of the second stage, a verbal description is formed for each item of the main successful scenario. The description format is determined by the item type according to the classification and the corresponding item model. It also defines the data that at this point in the scenario will enter or be extracted from the software product.

At the third stage alternative scenarios are formed. For each item in the main scenario, which contains a phrase like "the system confirms" or "Client / Actor agree", the user is prompted to create an alternative scenario.

Each alternative scenario begins with the phrase

$$N.l: S,$$

where: *N* is the number of the point of the main scenario in which some condition was not met; *S* is the text determining the condition for the transition to the alternative scenario.

The first item in the extension script is initialized by the system. The rest are built using the previously discussed item models.

The third stage completes the verbal description of the UC. The UC text can be presented to the Customer for agreement and approval. In addition, the necessary information has been prepared for further automation of the process of building a model of program classes (UC name, number, type, and data for each point of the scenario).

3.3. An improved method of forming a model of program classes

According to Fig. 1, after writing the UC, a model of conceptual classes is built, then – interaction diagrams and a model of program classes. The disadvantage of this technology is a large proportion of manual labor and the isolation of separate stages, which determines the high complexity of design as a whole. The developer, forming the UC, necessarily thinks about the ways of its implementation in the future system (Fig. 1, activity 1), but this is not recorded anywhere. They must perform the same work again when drawing up models of conceptual and program classes (Fig. 1, activities 2 and 3).

To eliminate the indicated disadvantages of the existing technology, a method is proposed that makes it possible to automate the process of forming a model of program classes. The essence of the method lies in the fact that upon completion of the description of each point of the scenario, a fragment of the MPC is created. It is intended for the implementation of the functions provided for by this point (Fig. 3). In this case, the formation of the UC and the creation of the class model are performed in parallel. If necessary, the process of forming a class model can be put aside, but all the necessary information for it is saved.

The use of the existing model of the program class (2) provides for a "manual mode" when performing operations of correcting the model, tracing, searching for classes and their functions. Therefore, to implement the proposed method for constructing the MPC, an improved model of the program class has been developed. This model significantly increases the information content of the existing model (2) and has essential additional capabilities in comparison with the model presented in [26].

Advanced model of program clasess

In the model of the program class [27], it is proposed to improve the typing of data. To realize it, all types are divided into simple and structured. The simple ones include:

Text is any text;

Numb is any number;

Bool is boolean value;

Void is absence of the data;

pClass is a reference to a class object.

The structured ones include:

List is a list (can represent a linear list, array, set, etc.);

Struct is a structure (in general, it contains fields of different types), must contain the numbering of the fields.

This approach makes it possible to describe rather complex data sets.

Example 1. Let it be the case that at a certain item k of the scenario it is necessary to present a list X, where each element represents a person P and contains a surname N and a year of birth Y. Then the type X will be described as follows

X: List-> P -> Struct (2) (1) N: Text (2) Y: Numb

It also provides a methodology for defining the goals of a class, methods, and attributes.

We represent the class by a tuple

$$cl= <cHeader, mCAttr, mFunc>. \quad (5)$$

The *cHeader* class header looks like this:

$$cHeader= <cName, tC, uName, nP, mPurp>, \quad (6)$$

where: *cName* is the class name; *uName*, *nP* is the name of the UC and the number of the scenario item in which the class or prototype was created; *tC* is the type of the class (class or prototype); *mPurp* stands for many purposes of using the class; each element of the set has the form: <uName, *nP*, *purpose*>.

A class can only be created in the "Create" UC item. In this case, one goal is introduced into the empty set, *mPurp*.

Example 2. A point of a certain scenario is being considered. "The client turns to the receiver in order to hand over things to dry cleaning. The receiver creates a new order in the system". In accordance with the item, a NewOrdern class will be created with the purpose of "Storing data on a dry cleaning order".

The rest of the elements of the *mPurp* set will be added as they appear in the function class. Let us represent the class function by the tuple

$$func= <fName, fPurp, mArgs, returnVal, mNewValAttr, mRfFunc>, \quad (7)$$

where: *fName* is the function name; *fPurp* is the purpose of using the function.

Whenever a service is required from a class, it must be represented by a function. Therefore, the purpose of using the function is added to the set of purposes of the class.

Example 3. A point of a certain scenario is being considered. "The client informs about the type of service. The receiver enters the type of service into the system. The system confirms". Let the ServiceTypeList class be created earlier. Then *fPurp* will take the value "Check for service availability", and the same value will be added to *mPurp*.

returnVal = <retType, purpose> – the value returned by the method;

The rest of the elements of expression (7) have the following meaning:

mArgs stands for many function arguments; each argument is represented by id, *argType*, and purpose of use *argPurp*;

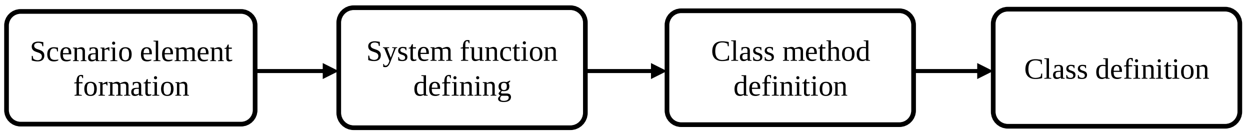


Fig. 3. **Relationship between the scenario item and class models**
Source: **compiled by the authors**

mNewValAttr is a set of attributes, the values of which change as a result of the method execution;

mRfFunc stands for many references to external functions (methods) of other classes that are used in this method. Each element of *mRfFunc* is represented by a tuple:

$$mRfFunc_i = <cName_j, fName>,$$

where: $cName_j$ is the class that the external function belongs to; *fName* is the name of the external function.

Let us represent the class attribute with a tuple

$$Attr = <attrName, attrPurp, attrType, mattrRf>, \quad (8)$$

where: *attrName* is the attribute name; *attrPurp* is the purpose of using the attribute; *attrType* is the attribute type; $c=\{<fName, cName, uName, nP>\}$– method references that use the attribute.

The purpose of the *attrPurp* attribute is actually its name in the language of the class modeller. The attribute description elements *attrName*, *attrPurp*, and *attrType* remain unchanged from the time the attribute is created. The *mattrRf* set is replenished with a new element each time the attribute is used with a new function. The proposed methodology for the formation of goals makes it possible to give a preliminary quantitative assessment of the degree of similarity of classes, methods and attributes.

Stages of the improved method of formation and the adjustment of the model of program classes

The method contains four stages.

At the first stage, the UC is determined for the construction of the MPC. In this case, two options are possible:

– a previously not described UC is selected and the MPC is formed simultaneously with the description;

– the UC is selected, for which the description was compiled, but the MPC was formed.

From the point of view of the total time spent on design, the first option is preferable. However, various circumstances can create a situation when the processes of UC description and MPC creation are separated in time. For example, a customer has limited the time it takes to get interviews from his employees. In any case, it is necessary to control the process of describing the UC.

We define the set of UC

$$mU= \{<uName_i, ma, state>\}, i = 1,k, \quad (9)$$

where: $uName_i$ is UC name; *ma* is a set of people interested in UC; *state* is a UC readiness degree.

There are three suggested values for the state: “Proposed”, “Description completed”, “Completed”. The degree of readiness “Proposed” means, from the point of view of the preparation of the MPC, the simultaneous work with the description. If the UC is readily “Composed Description”, then only the MPC is created.

The procedure for describing UCs, as well as designing the corresponding MPC for large software products, is difficult to establish in advance. This leads to the fact that a certain class may be in demand until the moment of its “creation” in a certain UC. Therefore, the model uses the concept of “class” for the previously “created” class and “prototype”– for a class that has not yet been created, but there is a need to use it.

At the second stage, changes are made to the MPC in accordance with each point of the scenario. Fig. 4 shows the main actions associated with adjustment of the MPC when implementing a scenario item in the model.

Adjustment of the MPC is realized by performing a combination of procedures corresponding to each point of the scenario (Table).

From Table 1 it follows that eight procedures in various combinations implement all types of scenario items.

At the third stage, an automatic construction of the specification of program classes is performed basing on the obtained MPC. The third stage is optional and is performed at the request of the developer.

The created MPC contains all the information necessary for constructing the specifications of the program classes. Therefore, specifications can be created without the participation of an expert.

We represent the specification of the program class in the form

$$classSpecif= <cName, mClassAttr, mFunc>,$$

where: *cName* represents the class name in the model and in the specification;

mClassAttr is a list of class attributes;

mFunc is a list of class functions (methods).

Each attribute can be represented as

$$attri= <attrName: attrType>,$$

where – *attrName*, *attrType* is the name and type of the attribute taken from the description of the attributes in the model (8).

We represent each function in the form

$$fName\ (mArgs): retType,$$

where: *fName* is the function name according to (7); *retType* is a return type; *mArgs* is a list of function arguments.

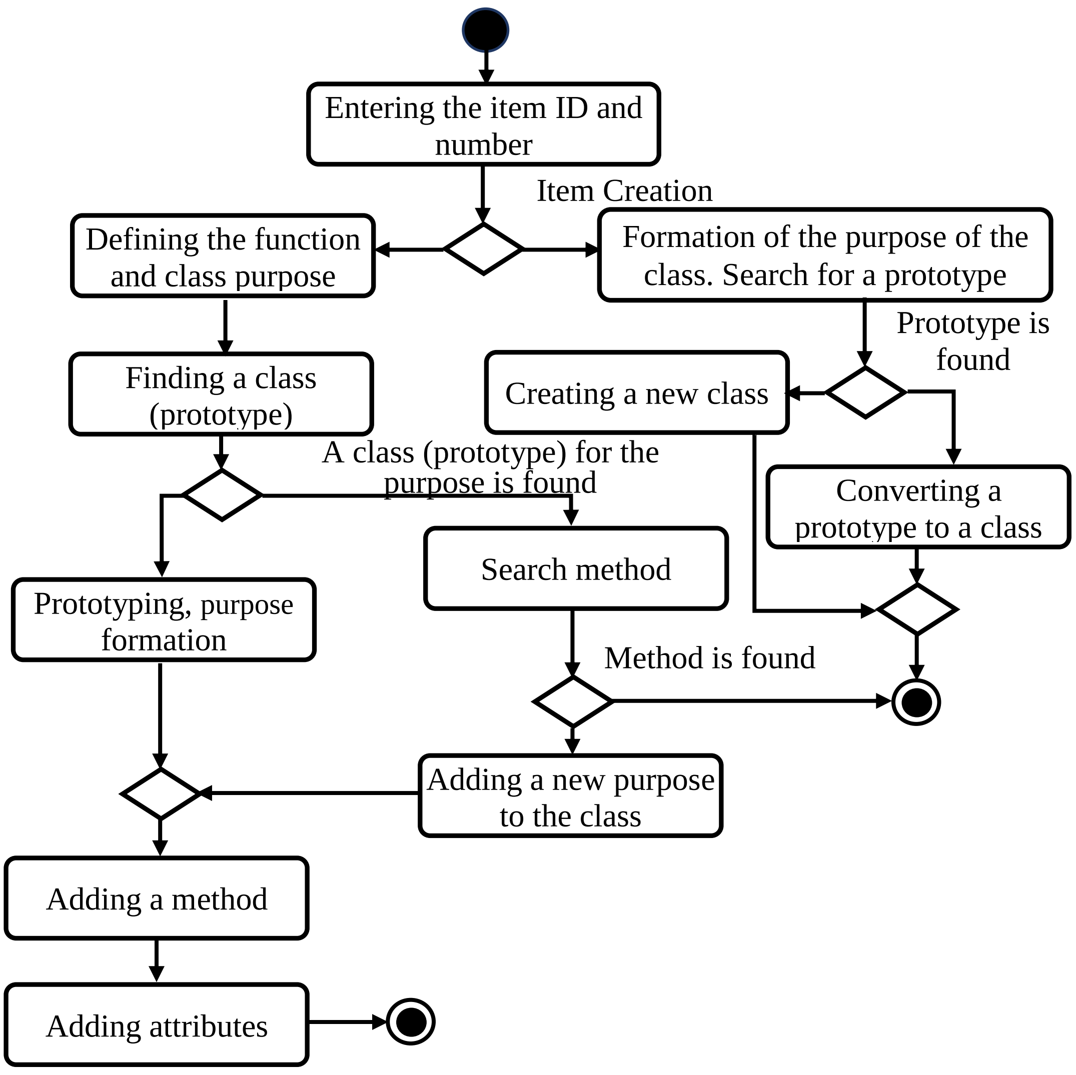

Entering the item ID and number
Item Creation
Defining the function and class purpose
Formation of the purpose of the class. Search for a prototype
Prototype is found
Finding a class (prototype)
Creating a new class
Converting a prototype to a class
A class (prototype) for the purpose is found
Search method
Prototyping, purpose formation
Method is found
Adding a new purpose to the class
Adding a method
Adding attributes

Fig. 4. **Changes to the Programming Class Model when a new scenario item is added**
***Source:* compiled by the authors**

Table .**Combinations of procedures for building a model in accordance with the points of the scenario**

Scenario Item Type	Class procedures
Data input	Method formation; search for classes by purposes; adding a method to a class
Select from the list	Method formation; search for classes by purposes; adding a method to a class
Successful completion of UC	Method formation; formation of attributes; search for classes by purpose
Unsuccessful completion of UC	Method formation; search for classes by purposes; adding a method to a class
Creation	Formation of the title; method formation; formation of attributes; search for classes by purposes; search for classes by purposes; class union
Value request	Method formation; search for classes by purposes; adding a method to a class; forming a class based on a method
Request for a list of values	Method formation; search for classes by purposes; adding a method to a class; forming a class based on a method
Request with the value entry	Method formation; formation of attributes; search for classes by purposes; adding a method to a class; forming a class based on a method
Service request	Method formation; search for classes by purposes; adding a method to a class; forming a class based on a method

***Source:* compiled by the authors**

We represent each argument as:

arg= <*id*: *argType*>,

where: *id* is the argument identifier; *argType* is the type of the argument.

At the fourth stage, the MPC is adjusted when editing requirements (scenario items). This stage is typically performed after code creation and testing, and it is mandatory for any agile technology.

Changing a scenario item will be considered as deleting the item in the old edition and drawing up a new edition of the item. The item deletion process involves deleting or adjusting the classes that "serviced" the deleted item. The correction algorithm is shown in Fig. 5.

When you delete an item, all classes that were used when creating this item are determined. If a class "serviced" only this item, then it is removed. If a method of some class "serviced" only this item, then it is deleted. If an attribute of a class was used only in the method that is being removed, then this attribute is also removed.

4. THE TECHNOLOGY OF THE SOFTWARE MODULE DEVELOPING

The new information OO-technology has been developed, which makes it possible to automate and tie together the processes of UC formation, the development and correction of the class model, the specification of program classes and unit testing (Fig. 6).

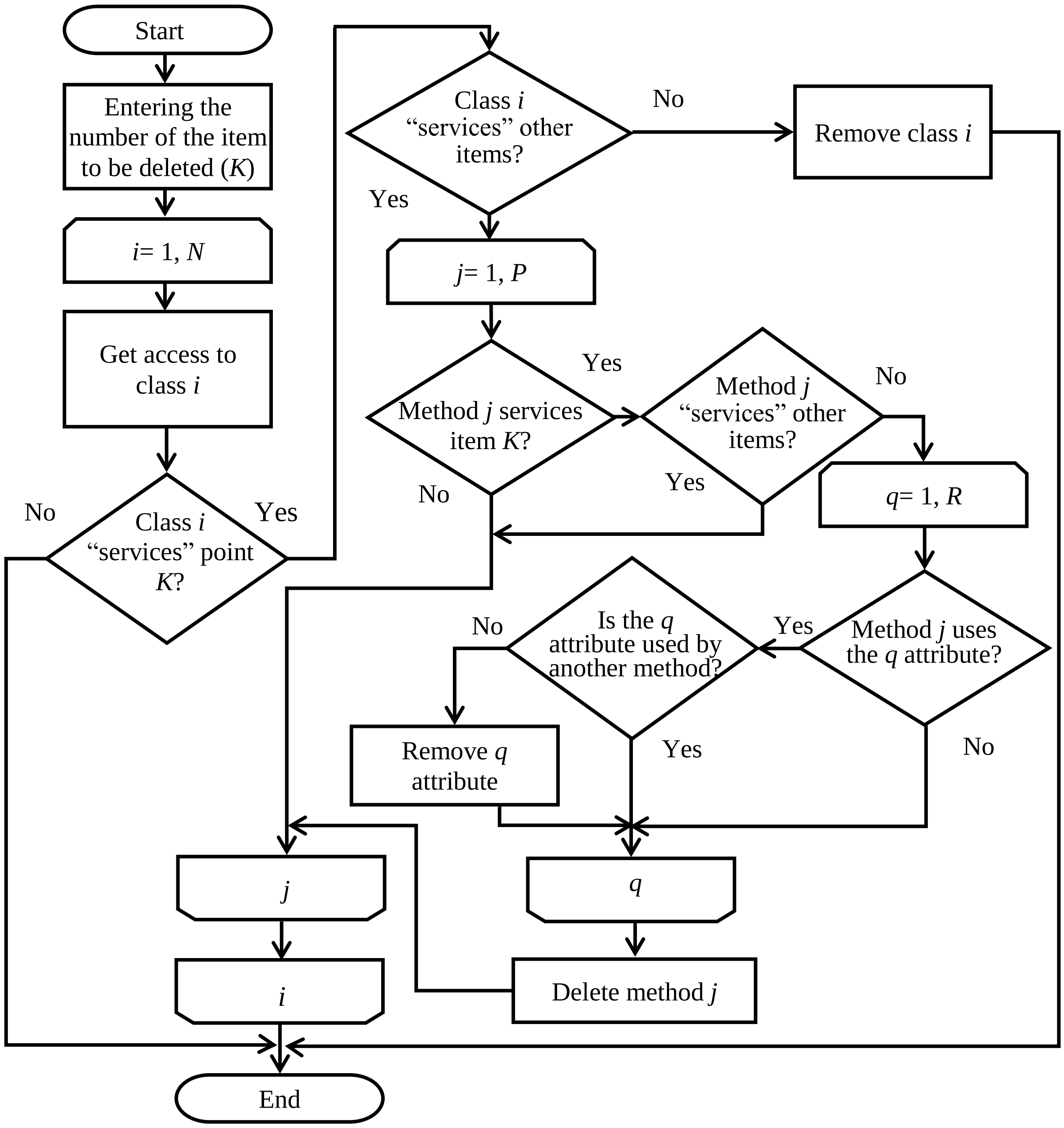

Start
Entering the number of the item to be deleted (K)
i= 1, N
Get access to class i
Class i "services" point K?
No
Yes
Class i "services" other items?
No
Yes
Remove class i
j= 1, P
Method j services item K?
Yes
No
Method j "services" other items?
No
Yes
q= 1, R
Method j uses the q attribute?
Yes
No
Is the q attribute used by another method?
No
Yes
Remove q attribute
j
q
i
Delete method j
End

Fig. 5. **Algorithm for correcting the model of program classes when deleting a scenario item**
Source: **compiled by the authors**

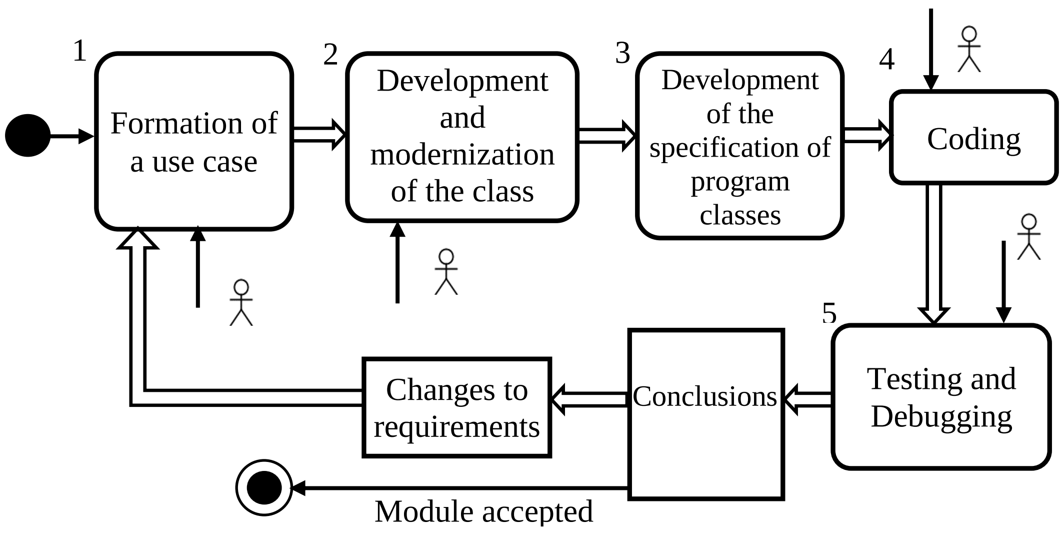


Fig. 6. **New technology for the software module developing**
***Source:* compiled by the authors**

In contrast to known technology (Fig. 1), the proposed technology is built of automated blocks, in which the number of local iterations is significantly reduced, as well as part of the developer's work in manual mode. Operations for identifying IS users and assigning them to the UC have been added to the process of UC formation. The need for the stage of creating a model of conceptual classes has disappeared.

Fig. 7 shows the main activities required to develop a software module and the degree of their automation in the existing and proposed technology.

For each activity, the symbols at the top left of the corresponding block characterize the existing technology, and the symbols at the top right - the proposed one.

From Fig. 7 it follows that the proposed technology makes it possible to automate most of the work.

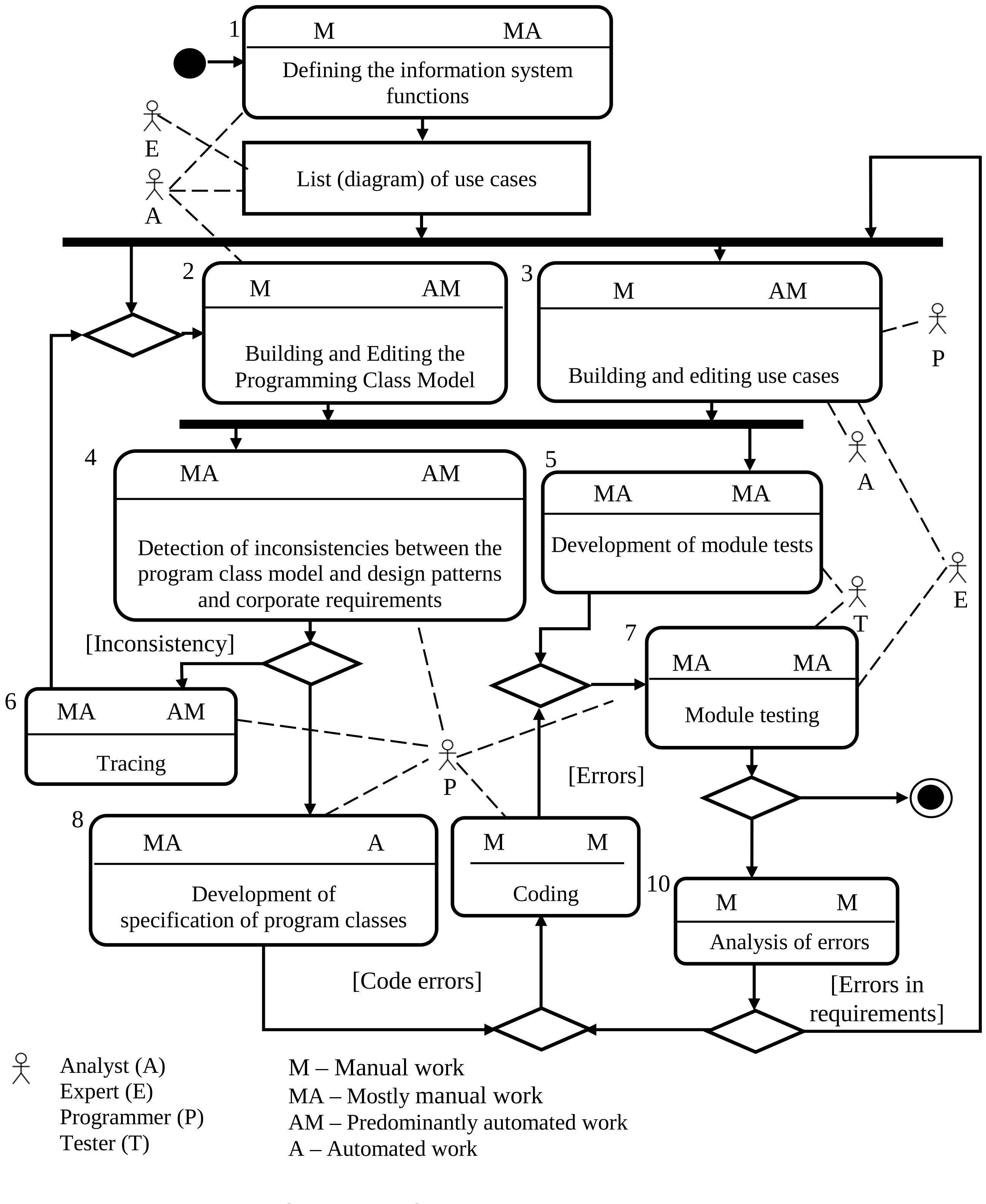
5. APPROBATION OF ACCEPTED SOLUTIONS

When designing software for the implementation and testing of methods and models, it was decided, if possible, to separate the part of the system responsible for describing the UC from the part responsible for creating the class model. Therefore, the precedent description subsystem was singled out (Fig. 8). Before creating a description of a specific UC, it may be necessary to adjust the general information about all UCs, in particular, the list of stakeholders and the list of UCs. For this purpose, the module “General Data Formation Wizard” is intended. The “UC preamble formation wizard” is a template for entering the data provided in expression (3). The “Master of the formation of items of the main scenario” implements the algorithms developed for each of the 10 types of items. The “Wizard for Forming Items for Alternative Scenarios” defines those items of the main scenario, which should have transitions to alternative scenarios. The wizard for the formation of data tables is connected in the case when the process of describing the UC and designing the MPC is performed at different times. In this case, the initial information about the data associated with the corresponding UC points is recorded in the “Data Table”.

The subsystem for generating class models – ModelEditor is shown in Fig. 9. A system analyst or programmer, using the Class Formation Wizard or the Item Removal Wizard, can select from the UC List the use case that interests them. To form a class, combinations of procedures are used that correspond to the type of a specific scenario item. The completed class model is placed in the List of class models. If it is necessary to delete a scenario item, the Item Delete Wizard adjusts the MPC in accordance with the algorithm shown in Fig. 5. In addition, the text of the scenario is edited (List of UCs). Since the proposed methods for automating the description of UC and constructing a model of program classes do not cover the entire cycle of creating a software module, the effectiveness of the decisions made was evaluated for the corresponding stages of the technology.

As criteria for evaluating the results obtained, two characteristics that are most often used in practice were used: the time to complete a certain amount of work and the number of errors made in this case. The second characteristic can be expressed through the first, but is useful for analyzing the results.

Ten students who successfully completed the study of the disciplines “Analysis of software requirements” and “Design of software” took part in experiments on the analysis of the automated UC formation method effectiveness. Five subject areas that were not considered in the educational process were proposed. During the research, students could use the Internet. The traditional method of compiling the UC corresponded to the "manual" mode, the proposed method was automated. Time was recorded from the moment the task was issued to the end of the work. As a result of the experiment, a decrease in the number of errors in the automated mode as compared to the "manual" mode was obtained by an average of 2.6 times (Fig. 10a), and the reduction in time– on average by 57 % (Fig. 10b).



1
M
MA
Defining the information system functions
E
List (diagram) of use cases
A
2
M
AM
Building and Editing the Programming Class Model
3
M
AM
Building and editing use cases
P
A
E
4
MA
AM
Detection of inconsistencies between the program class model and design patterns and corporate requirements
5
MA
MA
Development of module tests
T
[Inconsistency]
6
MA
AM
Tracing
7
MA
MA
Module testing
P
[Errors]
8
MA
A
Development of specification of program classes
M
M
Coding
10
M
M
Analysis of errors
[Code errors]
[Errors in requirements]
Analyst (A)
Expert (E)
Programmer (P)
Tester (T)
M – Manual work
MA – Mostly manual work
AM – Predominantly automated work
A – Automated work

Fig. 7. **Assessment of the degree of automation in new and traditional technology**
Source: **compiled by the authors**

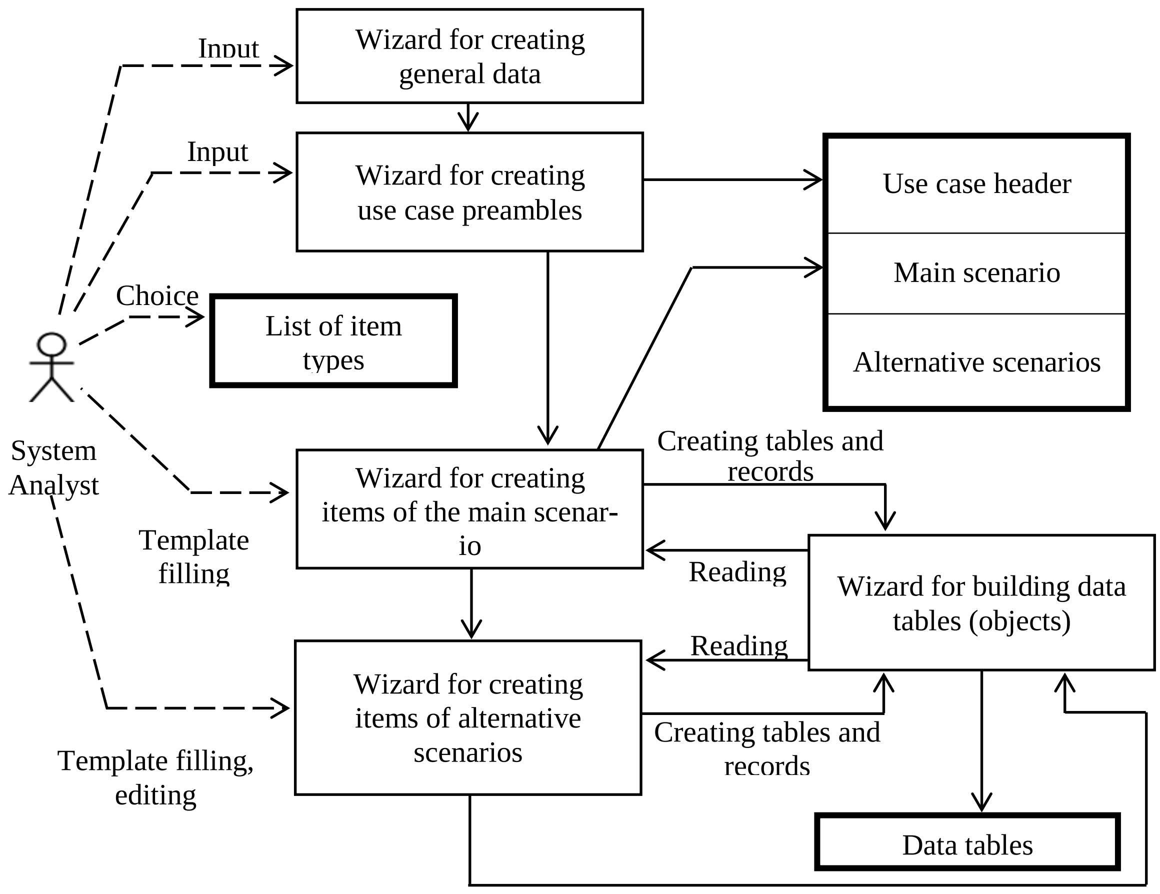

Input
Wizard for creating general data
Input
Wizard for creating use case preambles
Use case header
Main scenario
Alternative scenarios
Choice
List of item types
System Analyst
Wizard for creating items of the main scenar-io
Creating tables and records
Template filling
Reading
Wizard for building data tables (objects)
Reading
Wizard for creating items of alternative scenarios
Creating tables and records
Template filling, editing
Data tables

Fig. 8. **The structure of the use case description subsystem UseCaseEditor**
***Source:* compiled by the authors**

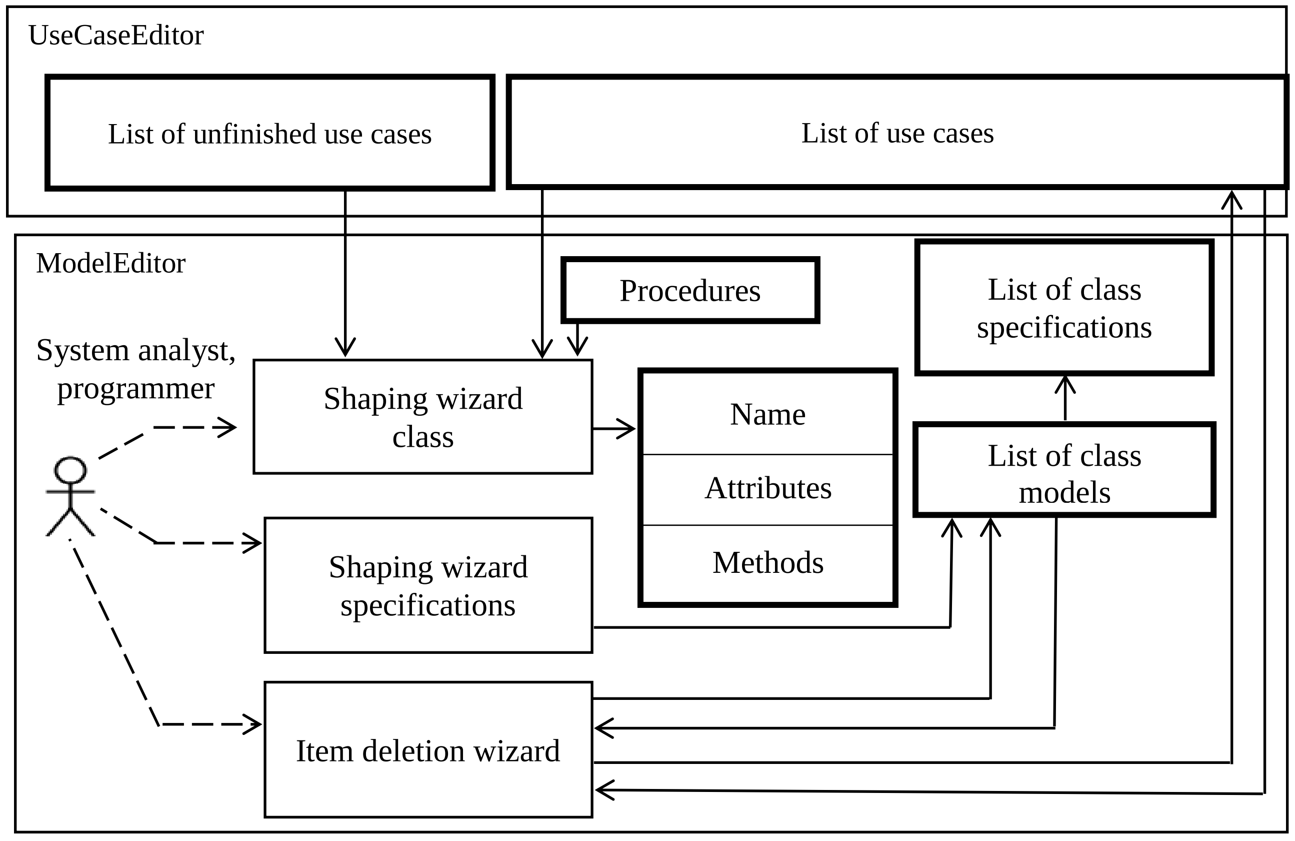
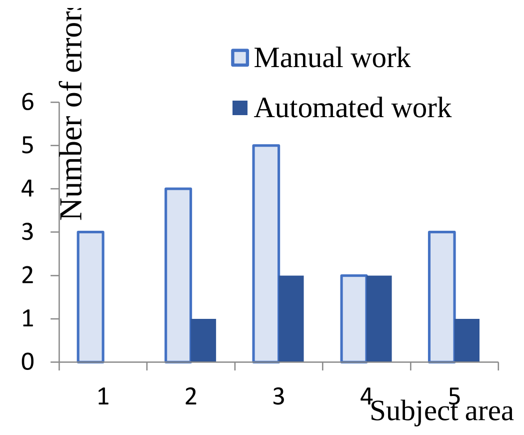
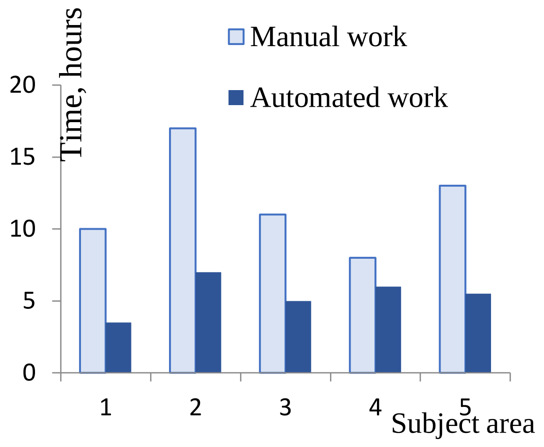


Fig. 9. **The structure of the subsystem for describing use cases ModelEditor**
***Source:* compiled by the authors**

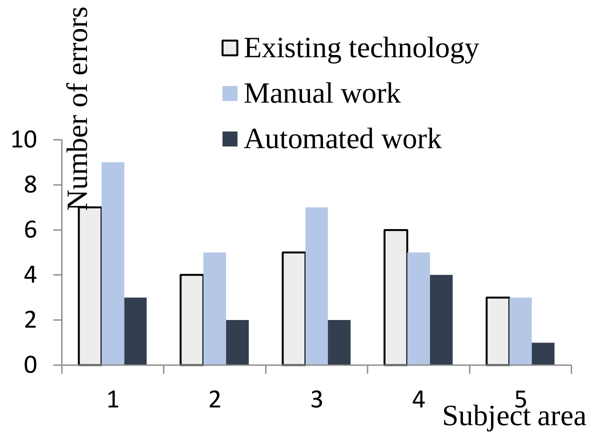


a

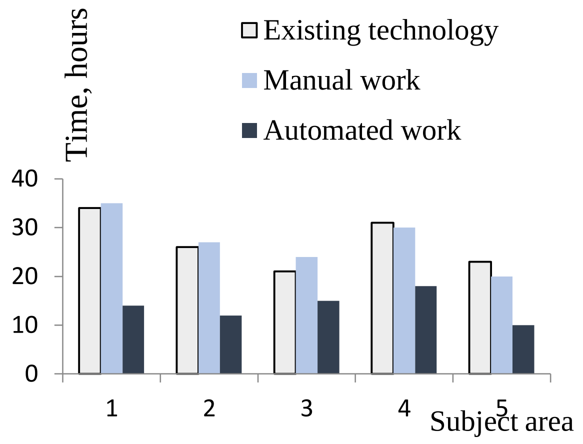


b

Fig. 10. **Comparative histogram for method of automated generation of use cases:**
a **– by the number of errors;**
b **– by the time of description**
Source: **compiled by the authors**



a



b

Fig. 11. **Comparative histogram for the model building method program classes:**
a **– by the number of errors;**
b **– by the time of description**
Source: **compiled by the authors**

To analyze the effectiveness of the method for constructing the MPC, a comparison was made between the three modes of creating the MPC. In manual mode – the proposed model was built without the use of software tools. In the automated mode - the same model, but using a software product. In the “classical technology” mode, a class model corresponding to this technology was created. The manual mode turned out to be the most laborious due to the use of a more complex model than in the classical technology. The automated mode showed an advantage over the existing technology in reducing errors by an average of 54 % (Fig. 11a) and time– on average by 41 % (Fig. 11b).

The proposed technology was also tested in the process of modernizing the existing IS. Within the framework of a real project, it was not possible to make a detailed comparison of technologies. According to the experts, the reduction in the time for the description of the UC, the construction of the MPC and testing was 19 %.

6. DISCUSSION OF THE OBTAINED RESULTS OF APPROBATION OF THE PROPOSED TECHNOLOGY OF CREATION OF SOFTWARE MODULES

Reducing the time and errors in the automated mode of constructing the MPC is obtained through the use of the proposed model of the program class representation and the method of its construction. This made it possible to reduce the time spent on searching for “suitable” classes, to formalize the description of methods and attributes.

In the proposed model of program classes and technologies, such relationships between classes as inheritance and composition have not yet been considered. Also, the issue of presenting some nonfunctional requirements has not been resolved, for example, the allowable delay in the response of the system. These disadvantages are the subject of further research.

As presented, the technology can be used in all projects where an object-oriented and the presentation of functional requirements in the form of use cases approach are adopted.

7. CONCLUSION

1. For the first time, the classification of items of UC scenarios has been performed on the basis of an analysis of the set of existing descriptions of UC from different subject areas. Ten types of items have been defined. The rules for UC description have been clarified, which made it possible to further formalize and automate the UC description process.

2. The method of automated UC description has been improved, which provides for the use of models for each type of item, which makes it possible to significantly speed up the process of describing UC and reduce the number of errors.

3. The method of forming a model of program classes has been improved by typing data at the level of the class model. The concepts of responsibilities for a class, its methods, attributes have been introduced. This made it possible to combine the process of building a specific class model with the formation of a scenario item and to automate the corresponding stage of work.

4. An object-oriented technology has been developed, which, unlike existing technologies, has made it possible to link and automate the main processes of development of a software module. The presence of additional information in the model of program classes made it possible to simplify the coding process, organize forward and backward tracing in the process of module testing and debugging. The totality of the decisions made increases the quality and shortens the development time in general.

5. Approbation of the research results has been completed. The efficiency of the decisions made is shown both at the stage of UC description and at the stage of compiling and using the model of program classes. This is confirmed by the reduction in the time for the specified types of work by approximately 50 % and the number of errors – by more than 2.5 times.

REFERENCES

1. Coburn, A. “Modern methods for describing functional requirements for systems” [in Russian]. Moscow: Russian Federation. 2016. 264 p. ISBN: 978-5-85582-326-4.

2. Hajri, I., Goknil, A., Briand, L. C. & Stephany, T. “Configuring use case models in product families”. *Software and Systems Modelling*. 2018; 17 (3): 939–971. DOI: https://doi.org/10.1007/s10270-016-0539-8

3. “UML fundamentals – use-case diagrams” [in Russian]. – Available from: https://proprof.com/archives/2594. – [Accessed Dec 2020].

4. Iqbal, S., Al-Azzoni, I., Allen, G. &Khan, HU “Extending UML use case diagrams to represent non-interactive functional requirements”. *E-informatica Software Engineering Journal*. 2020; 14 (1): 97–115. DOI: https://doi.org/10.37190/e-Inf200104.

5. “Use case and use case testing complete tutorial”. – Available from: https://www.softwaretestinghelp.com/use-case-testing. – [Accessed Dec 2020].

6. Kravchenko, I. A. & Speranskiy, V. A. “Cross-platform practices for mobile application development of automated trade accounting”. *Applied Aspects of Information Technology. Publ. Nauka i Tekhnika*. Odessa: Ukraine. 2018; Vol. 1 No.1: 48–58. DOI: https://doi.org/10.15276/aait.01.2018.3.

7. Velykodniy, S. S. “Analysis and synthesis of the results of complex experimental research on reengineering of open cad systems”. *Applied Aspects of Information Technology. Publ. Nauka i Tekhnika*. Odessa: Ukraine. 2019; Vol. 2 No.3: 186–205. DOI: https://doi.org/10.15276/aait.03.2019.2.

8. Mahmudova, S. “Methods of organizing the technological process of software development”. *Review of Information Engineering and Applications*. 2018; 5 (1): 1–11. DOI: https://doi.org/10.18488/journal.79.2018.51.1.11.

9. Powelll-Morse, A. “What is rational unified process and how do you use it?”– Available from: https://airbrake.io/blog/sdlc/rational-unified-process. – [Accessed Dec 2020].

10. Abramova, A. “Use cases. What are they and why are they needed?” [in Russian]. – Available from: https://systems.education/use-case. – [Accessed Dec 2020].

11. Larman, C. “Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development”. 2004. 736 p. ISBN-13: 978-0131489066.

12. Kungurtsev, O., Zinovatnaya, S., Potochniak, Ia. & Kutasevych, M. “Development of information technology of term extraction from documents in natural language”. *Eastern-European Journal of Enterprise Technologies*. 2018; Vol. 6 No. 2(96): 44–51. DOI: https://doi.org/10.15587/1729-4061.2018.147978.

13. Jacobson, I., Spence, I. & Bittner, K. “USE-CASE 2.0 The guide to succeeding with use cases”. 2011. 55 p. – Available from: https://www.ivarjacobson.com/sites/default/files/field_iji_file/article/use-case_2_0_jan11.pdf. – [Accessed: Nov. 2020].

14. Booch, G., Rumbaugh, J. & Jacobson, I. “Unified modeling language user guide”. 2005. 496 p. ISBN-13: 978-0321267979.

15. “Sequence Diagram”. – Available from: https://plantuml.com/ru/sequence-diagram. – [Accessed Nov 2020].

16. Freeman, E., Robson, E., Sierra, K. & Beit, B. Head First. “Design patterns” [in Russian]. 2018. 656 p. ISBN 978-5-496-03210-0.

17. Dubina, O. “Review of design patterns” [in Russian]. – Available from: http://citforum.ru/SE/project/pattern/. – [Accessed Nov 2020].

18. Koç, H., Erdoğan, A. M., Barjakly, Y. & Peker, S. “UML diagrams in software engineering research: a systematic literature review”. *7th International Management Information Systems Conference, 9-11 December 2020*. Proceedings 2021; 74 (1): 13. DOI: https://doi.org/10.3390/proceedings2021074013.

19. Ventayen, T. J. M. & Ventayen, R. J. M. “Systems modeling usage in project management among junior and senior business system developers”. *International Journal of Applied Science*. 2018; Vol. 1, No. 1; 1–7. DOI: https://doi.org/10.30560/ijas.v1n1p1.

20. Kungurtsev, A., Novikova, N., Reshetnyak, M. & Cherepinina, Ya. “Clarification of the classification and models of items of scenarios of use cases” [in Russian]. *Technical Science and Technology*. Chernigiv: Ukraine. 2018; No. 1 (11): 79–88.

21. “Enterprise architect. Create | verify | share. Official Version: 15.2 Build 1559”. – Available from: https://sparxsystems.com/products/ea. – [Accessed Nov 2020].

22. “40 use case templates & examples”. – Available from: https://templatelab.com/use-case-templates. – [Accessed: Dec. 2020].

23. Kungurtsev, O. B. & Zinovatna, S. L. “Work with the requirements to software” [in Ukrainian]. Odessa: Ukraine. 2019. 232 p. ISBN 978-966-927-457-1.

24. Kuzmin, A. A. “Hierarchical classification of document collections” [in Russian]. Thesis. 2017. 120 p. – Available from: http://www.frccsc.ru/sites/default/files/docs/ds/002-073-05/diss/08-kuzmin/008-kuzmin_main-txt.pdf?809. – [Accessed Dec 2020].

25. Adamovich, I. M., Volkov, O. I. & Markova, N. A. “A method for classifying information on the basis on hierarchical tags and its implementation on the example of a family archive fund” [in Russian]. *Systems and means of informatics.* 2012; 22. (2): 146–156. – Available from: http://www.mathnet.ru/links/91d22b2066a220d61047771352b266df/ssi284.pdf. – [Accessed Dec 2020].

26. Kungurtsev, O., Novikova, N., Reshetnyak M., Cherepinina Ya., Gromaszek, K. & Jarykbassov, D. “Method for defining conceptual classes in the description of use cases”. *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments*. 2019. Vol. 11176. DOI: https://doi.org/10.1117/12.2537070.

27. Novikova, N. O. “Changing and tracing of software requirements at level of conceptual classes”. *Applied Aspects of Information Technology. Publ. Nauka i Tekhnika*. Odessa: Ukraine. 2020; Vol. 3 No.1: 393–404. DOI: https://doi.org/10.15276/aait.01.2020.2.

Conflicts of Interest: the authors declare no conflict of interest

Received 12.12.2020
Received after revision 27.02.2021
Accepted 14.03.2021

DOI: https://doi.org/10.15276/aait.04.2021.4
УДК 004.4’24

Автоматизована об’єктно-орієнтована технологія створення програмного модуля

Олексій Борисович Кунгурцев[1)]
ORCID: https://orcid.org/0000-0002-3207-7315; akungurtsev19@gmail.com. Scopus Author ID: 57188743440
Наталія Олексіївна Новікова[2)]
ORCID: https://orcid.org/0000-0002-6257-9703; nataliya.novikova.31@gmail.com. Scopus Author ID: 57212034123
Світлана Леонідівна Зіноватна[1)]
ORCID: https://orcid.org/0000-0002-9190-6486; zinovatnaya.svetlana@opu.ua. Scopus Author ID: 57219779480

Наталія Олегівна Комлева[1)]
ORCID: http://orcid.org/0000-0001-9627-8530; komleva@opu.ua. Scopus Author ID: 57191858904
[1)] Одеський національний політехнічний університет, пр. Шевченка, 1. Одеса, 65044, Україна
[2)] Одеський національний морський університет, вул. Мечникова, 34. Одеса, 65029, Україна

АНОТАЦІЯ

Показано, що більшість технологій створення інформаційних систем засновані на об'єктно-орієнтованому підході й передбачає представлення функціональних вимог у вигляді варіантів використання. Однак не існує загальної думки про формат варіантів використання й правила опису пунктів сценаріїв. У роботі вдосконалена класифікація пунктів сценаріїв варіантів використання на основі аналізу множини існуючих описів з різних предметних областей. Уведено нові й уточнені існуючі правила опису варіантів використання, що дозволило надалі формалізувати й автоматизувати процес опису варіантів використання. Запропоновано також автоматизувати процес формування моделі програмних класів за рахунок внесення додаткової інформації, яка пов'язує клас із варіантом використання. Таким чином, модель програмних класів містить значно більше інформації для кодування, чим існуючі моделі в UML-діаграмах. Розроблено метод побудови моделі програмних класів. Методи автоматизованого опису варіантів використання й побудови моделі програмних класів зв'язані в єдиний процес. Рівень інформаційної насиченості моделі класів дозволяє також автоматизувати процес налагодження, пов'язаний зі зміною вимог. Оскільки ухвалені рішення стосуються більшості етапів процесу створення програмного модуля, у сукупності вони представляють нову технологію. Запропоновані модель, методи й технологія були реалізовані в програмних продуктах ModelEditor і UseCaseEditor. Апробація методу автоматизації опису варіантів використання показала зменшення кількості помилок у порівнянні із традиційним способом опису більш, ніж в два рази, і скорочення часу – більш, ніж в півтора рази. Апробація методу побудови моделі програмних класів показала його перевагу в порівнянні з існуючою технологією: зменшення кількості помилок і скорочення часу – практично в півтора рази. Запропонована технологія може бути використана при розробці будь-яких інформаційних систем.

Ключові слова: варіант використання; модель програмних класів; інформаційна технологія; об'єктно-орієнтована технологія

ABOUT THE AUTHORS

Oleksii B. Kungurtsev – Candidate of Engineering Sciences, Professor, Department of System Software. Odessa National Polytechnic University, 1, Shevchenko Ave. Odessa, 65044, Ukraine
ORCID: https://orcid.org/0000-0002-3207-7315; akungurtsev19@gmail.com. Scopus Author ID: 57188743440
***Research field*:** Methods and means of increasing the productivity of information systems; communication means with automated systems in natural language

Олексій Борисович Кунгурцев – кандидат технічних наук, професор кафедри Системного програмного забезпечення. Одеський національний політехнічний університет, пр. Шевченка, 1. Одеса, 65044,Україна

Nataliia O. Novikova – Candidate of Engineering Sciences, Senior Lector, Department of the Technical Cybernetics and Information Technology named prof. R. V. Merkt. Odessa National Maritime University, 34, Mechnikov Str. Odessa, 65029, Ukraine
ORCID: http://orcid.org/0000-0002-6257-9703; nataliya.novikova.31@gmail.com. Scopus Author ID: 57212034123
***Research field*:** Methods for object-oriented design of software products; learning automation systems

Наталія Олексіївна Новікова – кандидат технічних наук, старший викладач кафедри Технічної кібернетики й інформаційних технологій ім. проф. Р. В. Меркта. Одеський національний морський університет, вул. Мечникова, 34. Одеса, 65029, Україна

Svitlana L. Zinovatna – Candidate of Engineering Sciences, Associate Prof., Department of System Software. Odessa National Polytechnic University, 1, Shevchenko Ave. Odessa, 65044, Ukraine
ORCID: https://orcid.org/0000-0002-9190-6486; zinovatnaya.svetlana@opu.ua. Scopus Author ID: 57219779480
***Research field*:** Data analysis; information system productivity

Світлана Леонідівна Зіноватна – кандидат технічних наук, доцент кафедри Системного програмного забезпечення. Одеський національний політехнічний університет, пр. Шевченка, 1. Одеса, 65044,Україна

Nataliia O. Komleva – Candidate of Engineering Sciences, Associate Prof., Department of System Software. Odessa National Polytechnic University, 1, Shevchenko Ave. Odessa, 65044, Ukraine
ORCID: http://orcid.org/0000-0001-9627-8530; komleva@opu.ua. Scopus Author ID: 57191858904
***Research field*:** Data analysis; software engineering; knowledge management

Наталія Олегівна Комлева – кандидат технічних наук, доцент кафедри Системного програмного забезпечення. Одеський національний політехнічний університет, пр. Шевченка, 1. Одеса, 65044,Україна