UDK 004.415.2

Oleksiy B. Kungurtsev[1], Candidate of Technical Sciences, Professor of the System Software Department, E-mail: abkun@te.net.ua, ORCID: 0000 – 0002 – 3207 – 7315
Nataliia O. Novikova[2], Senior Teacher of the Department "Technical Cybernetics and Information Technology named prof. R. V. Merkt", E-mail: nataliya.novikova.31@gmail.com, ORCID: 0000 – 0002 – 6257 – 9703
[1] Odessa National Polytechnic University, Avenue Shevchenko, 1, Odessa, Ukraine, 65044
[2] Odessa National Maritime University, Mechnikov str. 34, Odessa, Ukraine, 65029

IDENTIFICATION OF CLASS MODELS IMPERFECTION

Abstract. *The analysis of methods for testing models of program classes is carried out. It is shown that in connection with the increase in the volume of work at the stage of compiling models, the relevance of model verification is increasing. It has been established that to test class models obtained as a result of an automated description of use cases, it is necessary to improve the existing class model and expand the set of checks in comparison with existing solutions. The class model was further developed. The model has three sections: the class head, class methods, and class attributes. The model improving is to introduce the concept of the purpose of creation and use for the class as a whole, its methods and attributes. Each operation associated with the construction of a class model is provided with a link to the corresponding use case and its item, which allows, if necessary, the transition from requirements to model description elements (direct trace) and from description elements to requirements (reverse trace). A type system for model elements has been introduced, which allows, without specifying types at the level of a programming language, to fully represent the declaration of functions and class attributes. Based on a number of design patterns and refactoring cases, three categories of situations are identified when the class model should be improved: criticisms on the class as a whole, criticisms on the functions of the class, criticisms on the attributes of the class. For each category, a set of criticisms on the model is established and solutions for their identification are proposed. The proposed models and algorithms are implemented in a software solution and have been tested in terms of the completeness of identifying criticisms on the model and reducing the time for the process of identifying criticisms compared to traditional technologies for defects detecting in the class models.*

Keywords*: use cases; class model; scenario, conceptual classes, design patterns*

Introduction

The representation of functional requirements as Use Cases (UC) is commonly used in the development of information systems (IS). The first fundamental work on determining the application scope, structure and description rules of UCs appeared at the beginning of the current century [1-2]. However, to date, interest in the use of UC has not diminished [3-4]. In [5], a classification of UCs scenario items was proposed, which laid the foundation for the automation of their description. Continuation of research in this direction [6] made it possible to supplement the classification and combine the description of UC with the construction of conceptual classes of a software product. In [7], an automated technology for the conceptual class modelling was proposed. In [8], methods for correcting the model of conceptual classes in connection with changes in the formulation of various items of scenarios for Use Cases are proposed and tracing of each item of the Use Case scenario in conceptual classes and their methods and attributes. Given the significant increase in the volume of work on a class modelling within the total amount of work on a software project, the task of automating the analysis and improving the quality of class models has become urgent.

Analysis of known solutions

One of the most well-known methods of analyzing class models is the compilation of CRC (class-responsibility-collaboration) cards [9]. Such cards allow to check the distribution of responsibilities between classes at the level of the aggregative functions of the software product being designed, therefore they are mainly used in the process of brainstorming. The task of improving the quality of models is directly related to the identification of metrics that determine quality. In [10], an attempt was made to determine the basic metrics that determine the quality of object-oriented software.

In relation to class models, it is of interest to implement the following metrics:

– Weighted Methods per Class (WMC); determines the number and complexity of methods implemented in the class;

– Lack of Cohesion of Methods (LCOM); defines the relationship between methods by parameters and attributes;

– Coupling between Object Classes (CBO); determines the number of other classes that this class is associated with.

In [11], based on a statistical study of multiple projects, conclusions were drawn about the frequen-

cy of occurrence of defects of various types in soft ware models, which allows prioritizing the functions of class models. In [12], it is proposed to take into account in complexity metrics frequency and density of software module failures.

In [13], it was proposed to introduce 3 levels of metrics to determine the quality of object-oriented software:

– Package level matrics;

– Class level matrics;

– Method level matrics.

From the point of view of model analysis, series of class and method level metrics are of interest: the length of the attribute and method name, the number of lines of code in the class, the number of methods and attributes in the class, the number of parameters in the method.

In [14], a quality model for new design patterns was proposed. The proposed description properties and corresponding metrics (utility, completeness, consistency, and compre-hensibility) are looking promising.

The study [15] analyzed the most popular of the existing approaches to verification of the most commonly used UML diagrams – class diagrams. It is shown that the method based on the creation of the driver, the pattern-based method, the method of constructing an identity graph and the method of representing classes in the form of sets allow solving only specific problems of analyzing the quality of models. It is shown how to comprehensively use these methods. In furtherance of the proposed method, a new language of block simulation has been created [16], which allows combining inductive and deductive approaches to creating simulation models in one model. The proposed methods and language do not allow testing all the components of class models built on the basis of UC, but they can be useful in assessing the completeness of the model considered in this paper.

An analysis of known solutions shows that existing technologies for analyzing class models do not cover all aspects of model testing or are not sufficiently automated. This leads to the retention of a significant share of "manual labour" in the analysis. Thus, the problem is to reduce the time it takes to analyze class models. To solve the problem, it is proposed to develop a method for identifying defects in models constructed as a result of the application of technology for the automated creation of class models in accordance with the description of the UC.

Study objectives

To solve the problem described, the following research objectives are outlined:

– improvement of the class model to expand its testing capabilities;

– identification of class mismatch with the basic design patterns;

– identification of the discrepancy between the methods of the class model and the requirements for the methods of the program class;

– identification of discrepancies of the attributes of the class model to the requirements for attributes of the program class.

Conceptual Class Model (CCM)

The model below differs from the model proposed in [7] in the following significant changes:

– the structural elements (head, methods, attributes) are distinguished in the model;

– the concept of the purpose of using the class is introduced, which allows to simplify the search for the desired class and apply design patterns to it;

– the concept of the purpose of using the class method is introduced;

– the concept of the purpose of creating a class attribute is introduced;

– such attribute characteristics as the minimum and maximum values that can be obtained in the general data packet when testing the modules are removed;

– the universal data types for class attributes, arguments and return values of methods are introduced.

All classes included in the CCM are represented by the set:

$$Mc = \{c\}, \quad (1)$$

Each class (prototype) is represented by a tuple

$$c =< cHead, mMeth, mAttr >, \quad (2)$$

where: $cHead$ is the class head;

– $mMeth$ is the set of class functions (methods);

– $mAttr$ is the set of class attributes.

The head of class

The head of class is represented by a tuple

$$cHead =< cName, tC, location, uName, nP, mPurp >,$$

where: $cName$ is the name of class;

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

– $tC =$"$class$"|"$prototype$" is the type of class;

– $location =$"u"|"s" is the lifetime of the class objects (either during the execution of the use case – u, or during the operation of the system – s);

– $uName, nP$ are the name of the use case (UC) and the number of the point where the class (prototype) was created;

– $mPurp = \{< uName, purpose >\}$ is a list of class usage goals.

Class functions

The set of class functions *mMeth* contains functions (methods) of the form:

$$func =< fName, mPurp, mArgs, returnVal, mRsArgs, mNewValAttr, mCalcAttr, mRfFunc >, \quad (3)$$

where: $fName$ is the name of function,

– $mPurp = \{< uName, purpose >\}$ is a list of purposes of using the function;

– $mArgs = \{ < id, \arg Type, \arg Purp > \}$ is a list of function arguments (each argument is represented by an identifier id, type $argType$ and purpose of use $argPurp$);

– $returnVal =< retType, purpose >$ is the value returned by the function (represented by the type of the returned value and the purpose of use);

– $mRsArgs$ is a set of method arguments used as a result of the calculation;

– $mNewValAttr$ is a set of attributes that take on new values as a result of function execution;

– $mCalcAttr$ is a set of class attributes used in calculations of this function;

– $mRfFunc$ is a set of references to external functions (methods of other classes) used in this method. Each element of the set $mRfFunc$ is represented by a tuple:

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

where: $cName_j$ is the class to which the external function belongs (in the general case, several external functions may belong to the same class); $fName$ is the name of the external function.

Class attributes

The set of class attributes contains attributes of the form:

$$mAttr = \{< attrName, purpose, attrType, mAttrRf >\}, \quad (4)$$

where: $attrName$ is the name of attribute;

– $purpose$– is the purpose of attribute use;

– $attrType$ is the type of attribute;

– $mAttrRf = \{< fName, uName, nP >\}$ is a set of references to functions (methods) that use the attribute.

Data types

Class attributes, method arguments, and method return values must be typified. However, the class model does not provide a specific language for its implementation. As a result of the analysis of the general approach to data typify [19] it was proposed to use the following data types in the model:

• *List* – a list (can represent a linear list, an array, a set, etc.);

• *Struct* – a structure (in the general case, contains fields of various types), must contain the numbering of the fields;

• *Text* – any text;

• *Numb* – any number format;

• *Bool* – Boolean value;

• *Void* – the function does not return a value;

• *PClass* – a reference to a class object corresponds to the class name $cName$;

The *Void* type isn't used for $argType$. The type *PClass* isn't used for $retType$.

Cases when a class model should be improved

Based on the analysis of the information presented in the class model, the previously considered metrics, well-known design patterns [17] and refactoring cases [18], the following options (criticisms) are proposed for the analysis of the class model with respect to its structural elements.

Criticisms of class in general:

– the class has many different purposes of use (mismatch with the High Cohesion pattern); the class is difficult to understand and maintain;

– the class has many references to other classes (mismatch with the Low Coupling pattern); the class is difficult to understand and maintain;

– the class has few functions differing of get ... () and set ... (); perhaps there is a need to add functionality to the class, or combine it with another class;

– a very large class; perhaps there is a need to split the class into several classes;

– it makes sense to present a class as derived from a base class; if two or more classes have a number of identical attributes and methods, then it makes sense to create some parent class, and present the classes in question as being derived from that one.

Criticisms of class methods:

– the method has many arguments; this makes it difficult to understand the method; perhaps there is a need to split the method into several ones;

– the method returns more than one value; if the method returns more than one value through attrib utes, or through a name and attributes simultaneously, then it makes the program difficult to understand and could be a source of errors;

– several methods of the same class have the same name; this makes it difficult to understand the program and could be a source of errors; perhaps the method should be renamed;

– the method uses many references to methods of other classes; it might make sense to transfer the method to another class;

Criticisms of class attributes:

– attribute values are set by methods of other classes (via set ... () methods of this class); it is allowed to use of the set ... () method when initializing the object; reuse often indicates that attribute value manipulations come from other classes;

– the attribute is duplicated in other classes; there are problems with maintaining the current value of the attribute; perhaps there is a need to place this attribute in only one class;

– the attribute is used by other classes through the get ... () method, i.e. without processing in its own class;

– the attribute is often used in other classes and rarely in its class; perhaps there is a need to transfer the attribute to another class.

Identification of cases requiring improvement of the class model as a whole

Inconsistency with the High Cohesion pattern. The set of purposes of class use should be analyzed. Analysis automation involves creating a set of purposes for a class by removing duplicated ones:

$$mPurp' = \{purpose\}. \quad (5)$$

To make a decision on the coincidence of purposes $purpose_i$ and $purpose_j$, it was proposed to apply phrase comparison using the multi-word terms identification method (MWT) [20]. We introduce a threshold value *TPurMax* (for example, 2), the excess of which should attract the attention of the researcher to the class

$$|mPurp'| > TPurMax.$$

Mismatch with the Low Coupling pattern. The connection of some class with other ones is determined by the presence of *PClass* type arguments among the arguments of the methods of this class, as well as references *mRfFunc* to the methods of other classes from the body of the methods of this class.

The set of references to other classes in method arguments is defined as:

$$mArgPClass = \bigcup_{j=1}^{j=n} \{arg_i \mid argType_i = PClass\};\ i = 1, m\ ,$$

where: *m* is the number of arguments in *i*-th method;

– *n* is the number of class methods.

The set of references to other classes in the body of methods is defined as:

$$mFuncRf = \bigcup_{j=1}^{j=n} \{mRfFunc_i.cName\};\ i = 1, p\ ,$$

where *p* is the number of references to external functions in the *i*-th method.

The number of connections of this class with other ones is calculated by the formula

$$mRf = mArgPClass \cup mFuncRf\ . \quad (6)$$

If $|mRf| > TCoupMax$, where *TCoupMax* is the permissible number of connections, then the researcher should pay attention to the class.

There are few functions differing of get ... () and set ... (). The analysis consists in determining the absolute and relative number of functions other than get ... () and set ... ().

The absolute number of functions

$$NAFunc = |\{func_i.Name \mid func_i.Name[1-3] \neq "set" \wedge func_i.Name[1-3] \neq "get"\}| \quad (7)$$

(5)

Here, the expression in square brackets defines the indices of the characters to be compared.

The relative number of functions

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

$$NRFunc = \frac{NAFunc}{|mMeth|}. \quad (8)$$

If $NRFunc > TFuncMax$, where $TFuncMax$ – is the admissible value of the ratio of functions, then the researcher should pay attention to the class.

The very large class. Typically, the size of a class is determined by the number of lines of its code. When analyzing models, the researcher can be provided with information on the number of methods and class attributes. Number of methods:

$$NMeth = |mMeth|. \quad (9)$$

Number of attributes:

$$NAttr = |mAttr|. \quad (10)$$

In this case, the metrics $NMeth$ and $NAttr$ contain enough information to estimate the size of the class.

It makes sense to present a class as being derived from a base class. To identify the "kinship" between classes c_i and c_j it is proposed to compare the lists of purposes of classes using.

We introduce the operation of determining the set of shared purposes of classes c_i and c_j:

$$mPurpose_{i,j} = mC_i.purpose \cap_{\approx} mC_j.purpose,$$

based on MWT. If $purpose_{i,j} \neq \varnothing$, then the assessment of the possible "kinship" of the classes is left to the discretion of the expert.

If the decision is positive, a list of attributes is defined that can be shared within classes c_i and c_j.

$$mSameAttr_{i,j} = \{c_i.mAttr_p, c_j.mAttr_k \mid (c_i.mAttr_p.purpose = c_j.mAttr_k.purpose) \wedge (c_i.mAttr_p.attrType = c_j.mAttr_k.attrType)\}, \quad p = 1,P;\ k = 1,K,$$

where P and K are the numbers of attributes in the classes c_i and c_j respectively, and the coincidence of purposes for the attributes is determined by the method of fuzzy string matching (FSM). If $mSameAttr_{i,j} \neq \varnothing$, then the assessment of the possible "kinship" of the classes is left to the discretion of the expert.

If there are coincident attributes, then you should look for methods coincident by purpose of use:

$$mSameMeth_{i,j} = \{c_i.mMeth_p, c_j.mMeth_k \mid (c_i.mMeth_p.purpose = c_j.mMeth_k.purpose)\},\ p = 1,P; k = 1,K.$$

He final decision on introducing a class hierarchy is determined by the expert after a detailed analysis of the methods.

Identification of cases requiring improvement of the class methods

The method has many arguments. It makes sense to determine the value that defines the concept of a "large" number of arguments. Let it be $TArgsMax$, then it is be possible to form and present to the researcher the set of class methods for which the number of arguments is equal to or greater than $TArgsMax$:

$$mFuncLArgs = \{func_i \mid (|func_i.mArgs| \geq TArgsMax)\}. \quad (11)$$

The method returns more than one value. The researcher should be provided with a set of methods that return more than one value with the name of the method or through attributes.

$$mFuncLreturnVal = \{func_i \mid (|func_i.mRsArgs| \geq 2) \vee (|func_i.mRsArgs| \geq 1) \wedge returnVal \neq Void)\}. \quad (12)$$

Several methods of the same class have the same name. The researcher should be provided with a set of methods that have the same name.

$$mFuncSameNames = \{func_i \mid (func_i.fName = func_j.fName)\}, \quad i = 1,n-1;\ j = i+1,n \quad (13)$$

where *n* is the number of methods within the class.

The method uses many references to methods of other classes. It makes sense to determine the value that defines the concept of a "large" number of references. Let it be $TRfMax$, then it is possible to form and present to the researcher the set of class methods for which the number of links is equal to or greater than $TRfMax$:

$$mLRfFunc = \{func_i \mid (|func_i.mRfFunc| \geq TRfMax)\}. \quad (14)$$

Identification of cases requiring improvement of the class attributes

Attribute values are set by methods of other classes (via set ... () methods of this class). It makes sense to determine the value that defines the concept of a "large" number of calls to the set ... () method. Let it is $TSetMax$, then it is possible to form and present to the researcher a set of attributes for which the number of calls to the set ... () method exceeds the value $TSetMax$.

Define the number of references to the set... () method for each attribute

$$NSet_i = |\{mAttrRf_i \mid mAttrRf_i.fName[1-3] = "set"\}|.$$

Define a set of attributes for which the number of links exceeds $TSetMax$

$$mAttr' = \{mAttr_i \mid mAttr_i.NSet_i \geq TSetMax\}. \quad (15)$$

The attribute is duplicated in other classes. To determine duplicate attributes, it is necessary to compare the attributes of different classes by name, purpose of use and type. To compare attributes, it was proposed to use fuzzy string matching methods [21-22]. In addition, it makes no sense to consider classes with a local lifetime ($location = "u"$), since for objects of these classes temporary duplication of attributes can be allowed. To help the analyst, we will form a set of attributes that can be considered as duplicating each other.

$$\begin{aligned} mSameAttr = \{&c_i.mAttr_p, c_j.mAttr_k \mid \\ &(c_i.mAttr_p.purpose = c_j.mAttr_k.purpose) \\ &\wedge (c_i.mAttr_p.attrType = c_j.mAttr_k.attrType) \\ &\wedge (c_i.location = "s") \wedge (c_j.location = "s")\}, \\ &i = \overline{1,q-1}; \quad j = \overline{i+1,q}, \end{aligned} \quad (16)$$

where q is the number of classes.

The attribute is used by other classes through the get ... () method, i.e. without processing in its own class. It makes sense to determine the value that defines the concept of a "large" number of calls to the get ... () method. Let it be $TGetMax$, then it is possible to form and present to the researcher a set of attributes for which the number of calls to the get... () method exceeds the value $TGetMax$.

Define the number of references to the get ... () method for each attribute

$$NGet_i = |\{mAttrRf_i \mid mAttrRf_i.fName[1-3] = "get"\}|.$$

Define a set of attributes for which the number of references exceeds $TGetMax$

$$mAttr' = \{mAttr_i \mid mAttr_i.NGet_i \geq TGetMax\}. \quad (17)$$

The attribute is often used in other classes and rarely in its class. It is necessary to determine for each attribute the number of references to the get ... () method and other methods, as well as the ratio of these values.

Define the number of references to other methods for each attribute

$$NOtherFunc_i = |\{mAttrRf_i\}| - NGet_i$$

$$RCoeff_i = \begin{cases} (NGet_i / NOtherFunc), \\ \quad if \quad NOtherFunc \neq 0 \\ 0, \\ \quad if \quad NOtherFunc = 0 \end{cases}. \quad (18)$$

The value $RCoeff_i = 0$ indicates that the attribute $attrName_i$ is not used in its class.

Testing the results of the study

To analyze the models, the Model Analysis software product was created using three specialized classes: the GeneralAnalysisClass class for identifying cases requiring improvement of the class model as a whole, the MethodAnalysisClass class for identifying cases requiring improvement of the class model methods, and the AttributeAnalysisClass class for identifying cases requiring improved class model attributes.

The usage of the UseCaseEditorV3 software product [7] to create the models under study turned out to be irrational since the model obtained as a result of the automated description of the UC contained a very small and unpredictable amount of inaccuracies. Therefore, the model was represented by the Abstract ConceptualClass, which attribute area contains the head of the class model (head), class model methods (metList) and class model attributes (attrList). To create models, we used its subclass WorkingConceptClass, which contains a dialogue constructor and the showAll() method to demonstrate the contents of the model (Fig. 1). Objects of the WorkingConceptClass class were created with

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

predefined inaccuracies that were to be detected in experiments. The use of the software product ModelAnalysis revealed all cases of deterioration of models. To determine the time advantage when applying ModelAnalysis, an experiment was conducted with 12 students.

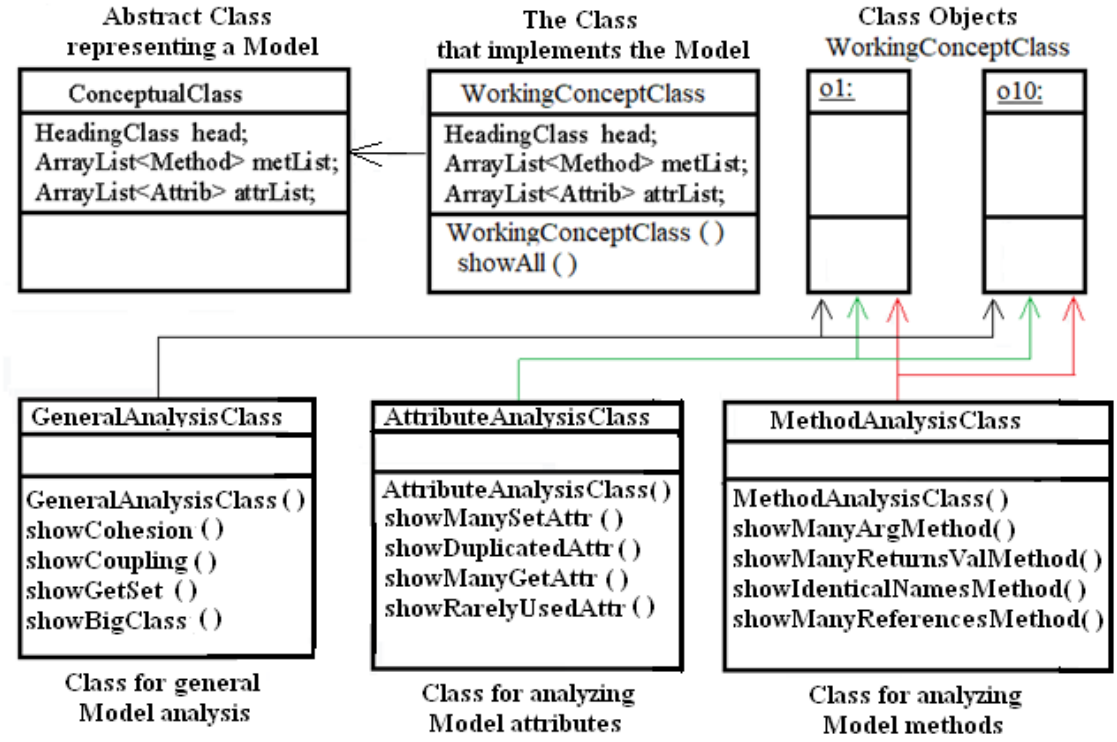

Fig. 1. Structure of classes implementing model analysis

For research, 5 and 10 models of classes related by reciprocal references were proposed. Respondents were asked to identify defects in the model. Defect detection time was fixed and averaged for all respondents. The experimental data are summarized in Table 1.

Table 1. The results of defects detection experiments in models

No.	Type of defects	Average time defect detection in minutes	
		(5 classes)	(10 classes)
1	Class has many different purposes	2	4
2	The class has many references to other classes	10	25
3	The class has some functions differing of get ... () and set ... ()	2	5
4	Very large class	1	2
5	It makes sense to present the class as being derived from the base one	9	22
6	The method has many arguments	1	2
7	The method returns more than one value	3	7
8	Several methods of the same class have the same name	1	2
9	The method uses many references to methods of other classes	6	17
10	Attribute values are set by methods of other classes	5	13
11	The attribute is duplicated in other classes	6	14
12	The attribute is used by other classes through the get ... () method	6	15
13	The attribute is often used in other classes and rarely in its own	5	12

The experiment showed that the detection of defects in the model in manual mode in some cases requires considerable time, while the automation of this process reduces the analysis time to fractions of a second. If we assume that the same time is spent on these stages in debugging, the proposed method allows reducing the debugging time by 50 %.

Conclusion. An improvement of the class model used in the UC automated description technology is proposed by introducing the concept of the purpose of use for a class, method, and attribute, which allows evaluating the quality of the distribution of responsibilities and data between classes. A list of possible data types for the model is developed, which allows checking the consistency of model description elements. Mechanisms have been developed for the automated detection of class discrepancy with basic design patterns; class model methods discrepancy with the requirements for program class methods, class model attributes discrepancy with the requirements for program class attributes. The software module has been created that allows class models testing. The experiments showed the effectiveness of the proposed solutions in terms of identifying defects and reducing the time to analyze class models.

References

1. Cockburn, Alistair. (October 15, 2000). "Writing Effective Use Cases. Crystal Series for Software Development 1st Edition", *Addison-Wesley Professional*, 294 p.

2. Leffingwell, Dean & Widrig, Don. (May 15, 2003). "Managing Software Requirements. A Use Case Approach. 2 edition", *Addison-Wesley Professional*, 544 p.

3. Skorikov, Eugene. (20.09.2019). "The strengthening of UseCase methodic (by the origins of Alistair Cockburn)" [Electronic Resource]. – Access mode: URL: https://habr.com/ru/post/468267/. – Active link – 20.09.2019 (in Russian).

4. Brandi, Denis. (Oct 16, 2019)."Why you need Use Cases/Interactors" [Electronic Resource]. – Access mode: URL: https://proandroiddev.com/why-you-need-use-cases-interactors-142e8a6fe576. – Active link – 16.10.2019.

5. Vozovikov, Yu. N., Kungurtsev, A. B. & Novikova, N. A. (2017). "Informacionnaya tekhnologiya avtomatizirovannogo sostavleniya variantov ispol'zovaniya", [Information technology for automated use cases], *Naukovi praci Donec'kogo nacional'nogo tekhnichnogo universitetu*, Pokrovs'k, Ukraine, No. 1(30), 46-59 p. (in Russian).

6. Kungurtsev, A. B., Novikova, N. A., Reshetnyak, M. U. & Cherepinina, Y. V. (2018). "Utochnenie klassifikatsii i modeley punktov tsenariev variatov ispolzovaniya" [Clarification of classification and models of scenario items of use cases], *Tehnichni nauki ta tehnologiyi*, Chernigiv, Ukraine, No. 1 (11), pp. 79-89 (in Russian).

7. Kungurtsev, Oleksii, Novikova, Nataliia, Reshetnyak, Maria, Cherepinina, Yana, Gromaszek, Konrad & Jarykbassov, Daniyar. (2019). "Method for defining conceptual classes in the description of use cases", *Proc. SPIE 11176*, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019, 1117624 (6 November 2019). DOI: 10.1117/12.2537070.

8. Novikova, Nataliia. (2020). "Changing and tracing of software requirements at level of conceptual classes", *Applied Aspects of Information Technology*, Vol. 3, No. 1, Odessa, Ukraine, *Publ. Science and Technica*. DOI: 10.15276/aait.01.2020.2.

9. Stoletoff, Dmitry. (17 Mar 2016). "CRC-cards – the project's everydayness" [Electronic Resource]. – Access mode: URL: https://imega.club/2016/03/17/crc-cards/ – Active link –17.03.2016.

10. Parthasarathy, S & Neelamegam, Anbazhagan. (June 2006). "Analyzing the Software Quality Metrics for Object Oriented Technology. " *Information Technology Journal* 5(6). DOI: 10.3923/itj.2006.1053.1057 [Electronic Resource]. – Access mode: URL: https://www.researchgate.net/publication/45949604_Analyzing_the_Software_Quality_Metrics_for_Object_Oriented_Technology.

11. Zhiyi, Ma. (June 2017). "An approach to improve the quality of object-oriented models from novice modelers through project practice", *Frontiers of Computer Science*, Vol. 11, Issue 3, pp. 485-498. DOI: 10.1007/s11704-016-5164.

12. Jabbar, Abdul & Sarala, Subramani (June 2011). "Advanced Program Complexity Metrics and Measurement", *International Journal of Computer Applications* , 23(2), pp. 29-33. DOI: 10.5120/2860-3679 [Electronic Resource]. – Access mode: URL:

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

https://www.researchgate.net/publication/269670226_Advanced_Program_Complexity_Metrics_and_Measurement.

13. Saeed, Mustafa Ghanem, Alasaady, Maher Talal, Malallah, Fahad Layth & Faraj, Kamaran HamaAli. (2018). "Three Levels Quality Analysis Tool for Object Oriented Program-ming", *International Journal of Advanced Computer Science and Applications*, Vol. 9, No. 11 [Electronic Resource]. – Access mode: URL: https://thesai.org/Downloads/Volume9No11/Paper_73Three_Levels_Quality_Analysis_Tool.pdf.

14. Pattiyanon, Charnon & Senivongse, Twittie. (2017). "Quality model for assessing object-oriented design patterns under development", *18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* [Electronic Resource]. – Access mode: URL: https://ieeexplore.ieee.org/document/8022749**,** DOI: 10.1109/SNPD.2017.8022749.

15. Litvinov, V. V. & Bogdan, I. V. (2012). "Testirovanie modeley obektno-orientirovan-nogo programmnogo obespecheniya" [Testing object-oriented software models], *MatematichnI mashini I sistemi*, No. 2 (in Russian).

16. Litvinov, V. V., Zadorozhniy, A. A. & Bogdan, I. V. (2017). "Yazyik blochnogo imitatsionnogo modelirovaniya na baze modifitsirovannyh diagramm deyatelnosti uml" [Language of block imitation modeling on the basis of modified diagrams of ump activity], *Matematichsh mashini i sistemi*, No. 4.

17. Freeman, Eric, Freeman, Elisabeth, Robson, Elisabeth, Sierra, Kathy & Bates, Bert. (2004). "Head First Design Patterns", *O'Reilly Media, Inc.*, 638 p.

18. Fowler, Martin. (November 30, 2018). "Refactoring: Improving the Design of Existing Code (2nd Edition)". Addison-Wesley Professional; 2 editions, 448 p.

19. Chauhan, Charad. (2013). "Prog-ramming Languages – Design and Const-ructs", *Laxmi Publications*, 280 p.

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

21. Kalyanathaya, Krishna Prakash. (2019). "A Fuzzy Approach to Approximate String Matching for Text Retrieval in NLP", *Journal of Computational Information Systems*, pp. 26-32 [Electronic Resource]. – Access mode: URL: https://www.researchgate.net/publication/333249900_A_Fuzzy_Approach_to_Approximate_String_Matching_for_Text_Retrieval_in_NLP.

22. Julien Tregoat. (Jan 9, 2018). "An Introduction to Fuzzy String Matching" [Electronic Resource]. – Access mode: URL: https://medium.com/@julientregoat/an-introduction-to-fuzzy-string-matching-178805cca2ab. – Active link – 9.01.2018.

Received 10.05.2020
Received after revision 03.06. 2020
Accepted 12.06. 2020

УДК 004.415.2

[1]**Кунгурцев, Олексій Борисович,** канд. техніч. наук, професор, професор каф. «Системне програмне забезпечення», E-mail: abkun@te.net.ua, ORCID: 0000 – 0002 – 3207 - 7315
[2]**Новікова, Наталія Олексіївна**, ст. викладач каф. «Технічна кібернетика й інформаційні технології ім. проф. Р.В. Меркта», E-mail: nataliya.novikova.31@gmail.com,
ORCID: http:// orcid.org/0000 – 0002 – 6257 – 9703
[1] Одеський національний політехнічний університет, пр. Шевченка, 1, м. Одеса, Україна, 65044
[2] Одеський національний морський університет, вул. Мечникова, 34, м. Одеса, Україна, 65029

ВИЯВЛЕННЯ НЕДОСКОНАЛОСТІ МОДЕЛЕЙ КЛАСІВ

Анотація. *Проведено аналіз способів тестування моделей програмних класів. Показано, що в зв'язку зі збільшенням обсягу робіт на етапі складання моделей, зростає актуальність верифікації моделей. Встановлено, що для перевірки моделей класів, отриманих в результаті автоматизованого опису варіантів використання, необхідно удосконалити існуючу модель класу і розширити набір перевірок порівняно з існуючими рішеннями. Отримала подальший розвиток модель класу. У моделі представлені три розділи: заголовок класу, методи класу і атрибути класу. Удосконалення моделі полягає у введенні поняття мети створення та спрямування класу в цілому, його методів і атрибутів. Кожна операція, пов'язана з побудовою моделі класу, забезпечується посиланням на відповідний варіант використання і його пункт, що дозволяє при необ-*

хідності виконати перехід від вимог до елементів опису моделі (пряме трасування) і від елементів опису до вимог (зворотне трасування). Введена система типів для елементів моделі, що дозволяє без конкретизації типів на рівні мови програмування досить повно представити оголошення функцій і атрибутів класів. На підставі ряду шаблонів проектування і випадків рефакторінга виділені три категорії ситуацій, коли слід покращувати модель класу: зауваження до класу в цілому, зауваження до функцій класу, зауваження до атрибутів класу. Для кожної категорії встановлено набір зауважень до моделі та запропоновано рішення для їх виявлення. Запропоновані моделі та алгоритми реалізовані в програмному рішенні і пройшли апробацію з точки зору повноти виявлення зауважень до моделі і скорочення часу на процес виявлення зауважень порівняно з традиційними технологіями виявлення дефектів в моделях класів.

***Ключові слова**: варіанти використання; модель класу; сценарії, концептуальні класи, шаблони проектування*

УДК 004.415.2

[1]**Кунгурцев, Алексей Борисович,** канд. технич. наук, профессор, профессор каф. «Системное программное обеспечение», E-mail: abkun@te.net.ua, ORCID: 0000 – 0002 – 3207 - 7315
[2]**Новикова, Наталия Алексеевна,** старший преподаватель кафедры «Техническая кибернетика и информационные технологии им. проф. Р.В.Меркта», E-mail: nataliya.novikova.31@gmail.com, ORCID: 0000 – 0002 – 6257 – 9703
[1] Одесский национальный политехнический университет, пр. Шевченко, 1, г. Одеса, Украина, 65044
[2] Одесский национальный морской университет, ул. Мечникова, 34. г. Одесса, Украина, 65029

ВЫЯВЛЕНИЕ НЕСОВЕРШЕНСТВА МОДЕЛЕЙ КЛАССОВ

***Аннотация.** Проведен анализ способов тестирования моделей программных классов. Показано, что в связи с увеличением объема работ на этапе составления моделей, возрастает актуальность верификации моделей. Установлено, что для проверки моделей классов, полученных в результате автоматизированного описания вариантов использования, необходимо усовершенствовать существующую модель класса и расширить набор проверок сравнительно с существующими решениями. Получила дальнейшее развитие модель класса. В модели представлены три раздела: заголовок класса, методы класса и атрибуты класса. Усовершенствование модели заключается в введении понятия цели создания и использования для класса в целом, его методов и атрибутов. Каждая операция, связанная с построением модели класса, снабжается ссылкой на соответствующий вариант использования и его пункт, что позволяет при необходимости выполнить переход от требований к элементам описания модели (прямая трассировка) и от элементов описания к требованиям (обратная трассировка). Введена система типов для элементов модели, позволяющая без конкретизации типов на уровне языка программирования достаточно полно представить объявление функций и атрибутов классов. На основании ряда шаблонов проектирования и случаев рефакторинга выделены три категории ситуаций, когда следует улучшать модель класса: замечания к классу в целом, замечания к функциям класса, замечания к атрибутам класса. Для каждой категории установлен набор замечаний к модели и предложены решения для их выявления. Предложенные модели и алгоритмы реализованы в программном решении и прошли апробацию с точки зрения полноты выявления замечаний к модели и сокращения времени на процесс выявления замечаний сравнительно с традиционными технологиями выявления дефектов в моделях классов.*

***Ключевые слова:** варианты использования; модель класса; сценарий, концептуальные классы, шаблоны проектирования*

Kungurtsev, Oleksiy B.
Research field: Automation of Information Systems Design, Computer word processing techniques

Novikova, Nataliia O.
Research field: Automation of Information Systems Design

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)