DOI: 10.15514/ISPRAS-2021-33(4)-1

Validation Automation of UML Diagrams Created by Students

T.S. Gasheva, ORCID: 0000-0002-8095-4538 <gasheva_99@mail.ru>
D.I. Vlasov, ORCID: 0000-0002-1968-5148 <dima.vlasov@icloud.com>
A.V. Otinov, ORCID: 0000-0002-4226-5694 <otinovandry@gmail.com>
N.N. Datsun, ORCID: 0000-0001-8560-7036 <nndatsun@inbox.ru>

Perm State University,
15, Bukireva st., Perm, 614990, Russia

Abstract. Unified Modeling Language (UML) is widely used standard for models visualization in software industry. Hence, a preparation of IT professionals involves the learning modeling process. Studies of student perception of UML modeling indicate that this process is perceived as quite complex. This paper presents software for validation activity, class and use-case diagrams by XMI representation. To achieve this goal, we researched existing methods and systems. Besides, we analyzed mistake catalogues and Perm State University's student models to propose a mistake classification and checklist that presents a list of validation to be done. This paper focuses on validation each type of diagram separately, without maintaining consistency between different UML models. However, all these validation modules are combined in one system, which allows to check any of the described types of diagrams.

Keywords: validation; UCD; AD; CD

For citation: Gasheva T.S., Vlasov D.I., Otinov A.V., Datsun N.N. Validation Automation of UML Diagrams Created by Students. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 7-18. DOI: 10.15514/ISPRAS-2021-33(4)-1

Автоматизация проверки UML диаграмм, созданных студентами

Т.С. Гашева, ORCID: 0000-0002-8095-4538 <gasheva_99@mail.ru>
Д.И. Власов, ORCID: 0000-0002-1968-5148 <dima.vlasov@icloud.com>
А.В. Отинов, ORCID: 0000-0002-4226-5694 <otinovandry@gmail.com>
Н.Н. Дацун, ORCID: 0000-0001-8560-7036 <nndatsun@inbox.ru>

Пермский государственный национальный исследовательский университет,
614068, Россия, г. Пермь, ул. Букирева, д. 15

Аннотация. Унифицированный язык моделирования (UML) является широко используемым стандартом для визуализации моделей в отрасли программного обеспечения. Следовательно, подготовка ИТ-специалистов включает в себя обучение моделированию. Исследования восприятия студентами UML-моделирования показывают, что обучение проходит довольно непросто. В данной статье представлено программное обеспечение для проверки диаграмм активности, классов и прецедентов по их представлению в формате XMI. Для достижения этой цели мы исследовали существующие методы и системы. Кроме того, мы проанализировали каталоги ошибок и модели студентов Пермского государственного национального исследовательского университета, чтобы предложить классификацию ошибок и контрольный список, который представляет список необходимых проверок. В данной статье основное внимание уделяется проверке каждого типа диаграмм независимо друг от друга, без сохранения согласованности между различными моделями

UML. Однако все эти модули проверки объединены в одну систему, которая позволяет проверить любой из исследованных типов диаграмм.

Ключевые слова: проверка; UCD; AD; CD

Для цитирования: Гашева Т.С., Власов Д.И., Отинов А.В., Дацун Н.Н. Автоматизация проверки UML диаграмм, созданных студентами. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 7-18 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(4)–1

1. Introduction

Modern approaches to validation student models are UML is a standard that provides system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes [1]. This standard is widely used in the software industry.

On the one hand, it is used in Object-oriented analysis and design (OOAD) in the development of complex systems [2]. Formal methods or model-based specification [3] are used to verify such models [4], [5].

On the other hand, a preparation of IT professionals involves the learning modeling process [6] and Model Driven Architecture (MDA) [7]. Studies of student perception of UML modeling indicate that this process is perceived as quite complex. This opinion is shared by both computer scientists [8], [9] and computer science majors [8].

Besides, the process of manual validation of models is a time-consuming process, especially during the review of dozens of student models by the teacher. Therefore, the creation of such system is actual.

This paper presents a description of a system for automatically checking use case (UCD), class (CD) and activity diagrams (AD) based on an XMI [10] representation. This format can be exported from most Case-tools creating UML diagrams and contains description of the diagram elements.

Not all types of UML diagrams were chosen for research. To automate the validation, two analysis phase models were chosen - UCD and AD, and one design phase model - CD. This choice is based on the experience of checking these models and on others papers [8], [11].

We researched existing tools for validation UML diagrams. We put forward the criteria of applicability to the solution of our problem for the found tools but they do not fulfill our criteria.

Modern approaches to validation student models are based on the use of catalogues or lists of common mistakes [12], [13], [14], [15], [16]. Using the results of these catalogues and own review models of Perm State University (PSU) students we classify the mistakes in two dimensions. The first dimension represents the severity of the mistake. We distinguish three types of severity: Warning, Serious, Fatal. In the second dimension, we consider three main categories of mistakes: Model, Consistency, Layout. Model mistakes contains subcategory.

In this study, we consider in detail only the model mistakes. Using these mistakes, checklists for UCD, CD and AD are composed. These checklists are used to validate diagrams using Case-based reading method (CBR) [17]. Besides, for checking AD we use a graph with semantics similar to the Petri net, which was mentioned in the article [18], and colored tokens, similar to tokens in colored Petri nets [19].

To this end, we present the system implementation and demonstrate case studies. The quality of mistakes detections by the system is high, which confirms the achievement of the research goal.

2. Related Works

2.1 Analysis of Existing Mistakes Classifications

Chren et al. [12] evaluated over 2700 UML diagrams and examined students' mistakes with use case diagrams, activity diagrams, class diagrams, state machine diagrams, sequence diagrams, communication diagrams, and entity-relationship diagrams. They produced their catalogue from papers and their own research and identified 146 mistakes. They split all mistakes in two dimensions – the first one represents the severity of mistakes, the second refers to the nature of the mistakes. The overview of the classification scheme is in fig. 1 [12].

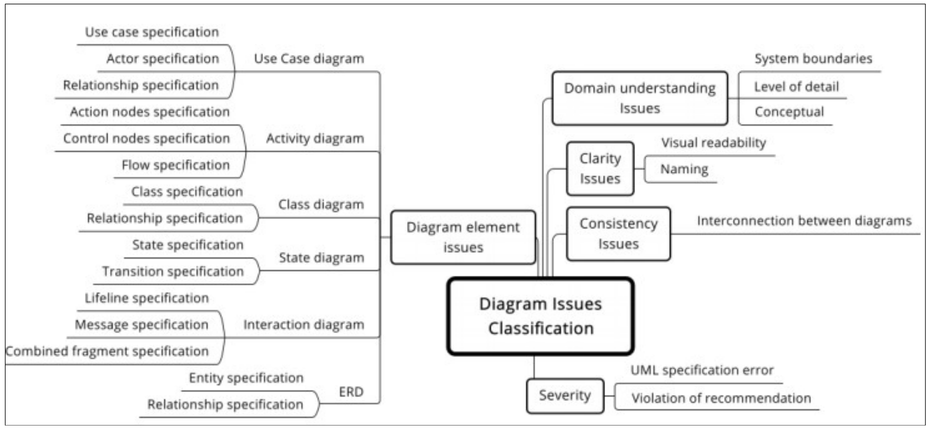


Fig. 1. Orthogonal classification scheme for diagram issues

They figured out that the most common mistakes for UCD, AD, and CD are Diagram element issues. The detection rate for top four mistakes in UCD is 31-42.9%, in AD - 26.2-42.9%, in CD - 42.9-59.5%.

Delgado et al. [16] considered use case, class, state-machine, sequence diagrams. They separated mistakes on six categories: Layout mistakes, Traceability, Notation, Semantic, Documentation, Naming, Conventions. They concluded that in UCD the most common mistakes are Notation mistakes and the least common are Traceability mistakes. In CD the most common defects are Documentation and the least common Traceability.

Reuter [8] classified mistakes in a category system that is commonly known from errors in programming languages and the Standard Classification for Software Anomalies [20]: Lexical, Syntactic, Semantic, Logic, Missing, and Unnecessary. Unfortunately, the numbers of mistakes were not presented, so that no conclusions could be drawn about the frequency of mistakes.

Bolloju [15] separated 380 mistakes in the 14 projects in three groups: Syntactic, Semantic, Pragmatic. They found out that the most frequent types of mistakes for UCD and CD are Syntactic (57% for UCD and 64% for CD) and Semantic (64% and 50%, respectively).

In our work, we use classification structure similar to Chren's. We also use two dimensions, one of which is severity and another links with the origin of the mistake. However, we use different severity subcategories. It relates with our future work – the development of system of qualifying the degree of deviation of the student model.

2.2 Analysis of Existing Software for Validation of UML Diagrams

Special criteria were identified for the existing UML diagram validation software:

- meet all the standards of the UML language;
- open source and free to use;
- support for graphical display of diagrams;
- software is easy to install and use.

The first two criteria are critical when analyzing existing software because of the target group of users – teachers and students.

For UC validation, two software products were discovered: FOAM [21] tool and Rational Rose [22]. FOAM it's open-source project, but disadvantage of this tool is that in order to analyze an existing UCD, you should independently translate all the contents of the diagram into a special text format, in which you must manually specify all the established elements and the relationships between them. Also, this tool doesn't have a graphical interface. Rational Rose it's a commercial product, that supports modeling UCDs and their continues validation. But the list of checking mistakes is quite small.

For AD validation, there are analyzed tools such as UML-VT [23] and Woflan [24]. These funds also do not match the main criteria identified earlier.

In existing publications [25], there are only brief descriptions of algorithms for validating CD, and it is impossible to study and analyze them in detail, since they are in the private domain. The set of libraries used in private validation systems: Eclipse (2000 LoC) and Java classes (11500 LoC) [26] [26], Dresden OCL toolkit [26], [27] extensible libraries (for processing and loading constraints) and MDR (for importing/exporting UML models from XMI [10]).

2.3 Analysis of UML Diagram Creation Software Providing Metadata

The choice of the software that will be used for the construction of diagrams is an important step in this work, since the chosen software tool will determine the possibility and success of further analysis, design and implementation of the prototype. That is why special requirements were defined for the selection process of competing modeling systems. The result of the research in this issue was the choice of the GenMyModel [28]. It combines a simple user interface, does not require installation and has the function to export the diagram in the required formats. The advantage of this tool is that when building diagrams, it does not allow you to perform some activities that can lead to mistakes. Due to this, the list of conditions to be checked can be reduced.

Based on this, it was decided that the input data (XMI and PNG files) will be generated using the GenMyModel tool.

3. Classification of Students Mistakes

Based on existing catalogues and own student's models analysis, we composed the classification.

1) Model: The Model category covers mistakes that violate the UML specification or recommendation.
   a) Lexical.
   b) Syntactic.
   c) Semantic.
2) Consistency: The Consistency category contains mistakes that are related to maintaining the dependencies between the diagrams.
3) Layout: The mistakes in the Layout category are caused by incorrect arrangement of elements on the diagram (overlapping elements, crossing of relationship lines); these mistakes are extremely difficult to detect using the XMI representation.
4) Severity.
   a) Warning.
   b) Serious.
   c) Fatal.

The Severity category depends on how accurately the mistake can be detected by system (warning – if the mistake cannot be accurately identified by the system), on the possibility of further validation (fatal mistakes in AD lead to the termination of further validation) and on the recommendation (warnings for minor mistakes, fatal for major mistakes).

Model mistakes are also divided into lexical, syntactic, and semantic subcategories. On lexical step, the diagram elements are considered separately, individually (a mistake in the name of the element, the type of the element does not belong to the diagram). On syntactic step, we consider mistakes related to the constructing diagram rules, the rules for connecting diagram elements. Semantic captures the quality of a model. This category covers mistakes relating to invalid representation of domain, violation of the boundaries of the system, incorrect display of meaning.

In this work, we consider in detail only the model mistakes. GenMyModel does not allow to make some syntax mistakes in UCD, AD and CD, so we did not include them in the checklists. Table 1 presents the examples of Model mistakes.

Table 1. Examples of model mistakes

Mistake	Subcategory	Severity
Invalid actor name: should be represented by a noun, starting with a capital letter	Lexical	Serious
More than one initial node	Syntactic	Fatal
When specifying the roles multiplicity, some numbers are negative integers	Semantic	Serious

4. Validation Methods

4.1 Approach for Use-Case Diagram Validation

A research was carried out among the existing UCD validation methods. For the use case diagrams, the following were identified: Object-Oriented Reading Techniques (OORT) [29] and CBR [30] – reading based on a list of requirements.

Since we have a list of mistakes, the CBR methodology was chosen. CBR is a very common method. List of mistakes should be checked during the validation. CBR provides more aid and advice to the inspectors than ad-hoc reading and is therefore a very common technique.

Initially, the XML document is read into the program internal data representation. Each chart element has a unique identifier, coordinates on the image, and a name or description. On the reading stage, some types of mistakes can be verified (mostly lexical). Items not included in the UCD list are excluded.

The next stage involves conducting sequential checks of each type from the list of the most common mistakes of students.

4.2 Approach for Activity Diagram Validation

To solve the AD validation problem, the analysis of existing methods was carried out, such as the construction of a Petri net [31], the use of temporal logic [32], as well as graph with semantics similar to Petri Net [18].

For this work, we use a subset of UML elements. We consider the initial node, final node, decision node, merge node, activity node, fork node, join node, swimlane, comment node, flow.

Now we describe the basic idea of the validation process. For each element in AD, there is a class in our system. Each class has field for token and field for list of links on the next objects. Besides, it can contain additional information about AD element. For each AD element, the object is created

and placed in a graph. The graph's vertices represent AD's nodes and the graph's edges represent AD's flows.

The validation is divided into two steps. At the first step, lexical, semantic and part of syntactic mistakes are checked. Then we check unpaired using fork and join by modeling token flow through the graph.

After the model passes the first steps of validation, in order to continue validation, we impose some restrictions. The restrictions are as follows.

1) AD must have exactly one initial node, one or more final nodes and at least one activity node,
2) initial node has no incoming edge and the final node has no outgoing flow,
3) activity, merge, join and init nodes must have exactly one outgoing flow,
4) fork and decision nodes can have any number of outgoing flows,
5) activity, fork, decision, final nodes should have only one incoming flow,
6) each join and merge node can have any number of incoming flows,
7) flows cannot start and finish in the same node.

If the restriction was violated, we finish validation without graph checking.

The tokens flow through the graph along the edge directions from initial to final node. The validation is completed when all token flows are checked or a mistake is found. In this case mistakes are the situations when several tokens appear in one node at once or when a token remains in the graph upon reaching the final node or when deadlock occurs (the situation when there is no token can be moved).

According to [18], the graph uses token-flow semantics. Each element has own set of rules, which ensure the token flows through the graph.

The state of the graph at each step can be encoded with a sequence of zeros and ones, where zero means that the element is inactive (does not have a token), and one means that it is active. A stack of current masks and a set of checked masks are created. At each step, the top mask is taken from the current masks and processed. The processing's result is new masks, that are checked for use early using a set of used masks and, if they have not been previously used, are added to the list of current masks.

At each step, all existing tokens are moved to one of the next nodes. In the case of a decision node, a token can activate a random element. Therefore, it generates several possible next states that are pushed into the current mask stack.

However, the problem of unpaired use of a joint and a fork remains. When there are several fork nodes corresponding to one join nodes, the join can be activated wrongly. Fig. 2 presents this issue.

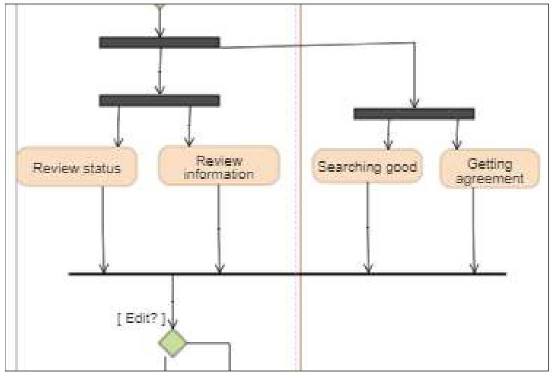


Fig. 2. Several fork nodes corresponding to one join nodes

To figure out this kind of mistake, it was proposed to use tokens of different colors. It is some additional data that is stored on the token's stack [19]. The fork node generates a unique color every time a token pass through it. The output tokens have the same color; it means that the fork's color is placed on the token's stack of colors. For join node activation, it is necessary that the colors at the top of the stacks have the same color. If the condition is right the join node becomes activated, and the output token remains all colors except the top color. In other case, a fatal mistake occurs and validation is completed.

4.3 Approach for Class Diagram Validation

There are very few fully automated methods for validating CD. Most of the existing solutions require a translation process into specific data formats that must be performed by a human. This approach is not suitable, since we need to quickly validate the diagrams, and the translation process takes a sufficient amount of time.

The validation process is based on and similar to the program compilation analysis stage, and it can be divided into three stages: the first stage is lexical analysis, the second is syntactic analysis and the third stage is semantic analysis. At each stage, the corresponding rules will be checked. During the first stage, metadata is converted into a set of tokens, the use of invalid tokens, incorrect names, designations and properties of tokens is detected. During the second stage, the correctness of creating constructions of the UML language from a valid set of tokens. And at the final stage of validation, the semantics of the constructed class diagram is considered, namely the correctness of the semantic meanings of words, phrases and elements.

Validation process begins with reading all data about the model from the XMI file. All properties of the CD tokens can be retrieved from these data. Already on the basis of these properties, it will be easy to detect some inconsistencies and mistakes.

The main point in this method is to designate a set of rules for constructing UML CD such as to identify all mistakes in the validated diagram. The set of rules was compiled using the UML specification [1]. The list of all rules that will be checked during validation was described in detail earlier. Also, special attention will be paid to common mistakes when constructing CD.

5. Implementation

For developing the system, we used C#. The process of validation system creating was divided into two stages:

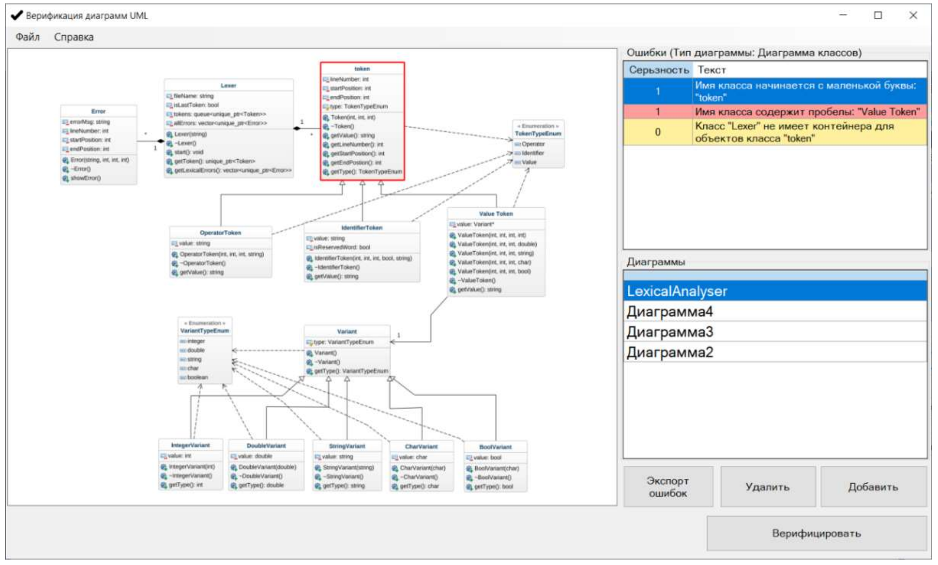
1) implementation of the UCD, AD, and CD validation modules in prototype mode as a console application and presentation the result in the form of text message,
2) integration of UCD, AD, and CD validation modules into the system with a user graphical interface.

The system has the following features:

1) the ability to upload one or several files into the system;
2) the ability to validate one or several models;
3) the ability to automatically find the pair "metadata - image" while files are uploading into the system;
4) the ability to add and remove diagrams from the current list of models;
5) the ability to work only with metadata diagrams (without images);
6) dynamic changing the graphical presentation of diagrams while switching is occurring between them;
7) dynamic mistakes designation on the graphical presentation of the diagram while switching is occurring between mistakes;
8) highlighting each mistake in a different color depending on its severity;

9) the ability to check all diagrams with "not checked" status at once.

Fig. 3 shows the results of CD validation.



Ошибки (Тип диаграммы: Диаграмма классов)

Серьзность	Текст
1	Имя класса начинается с маленькой буквы: "token"
1	Имя класса содержит пробелы: "Value Token"
0	Класс "Lexer" не имеет контейнера для объектов класса "token"

LexicalAnalyser
Диаграмма4
Диаграмма3
Диаграмма2

Fig. 3. Results of CD validation

6. Case Study

We analyzed the quality of projects developed in PSU by full-time undergraduate majors "Software" and "Computer security» (course «Modeling of Information Systems») and part-time specialty "Information technologies" (course "Design and implementation of information systems»). These are projects of information systems (IS) for business applications, computer security, and information technology. Each project includes models of three stages of IS life cycle: analysis (UCD, AD, sequence diagrams), design (collaboration diagrams and CD), implementation (component diagrams and deployment diagrams). The projects were carried out by teams of 2-3 students.

The checklists used in CBR to validation diagrams include 6, 10, and 5 lexical mistakes; 14, 15, and 8 syntactic, 7, 1, and 2 semantic mistake for CD, AD, and CD.

Fig. 4 presents examples of mistakes in Table 1 in: a) for UCD, b) for AD, c) for CD.

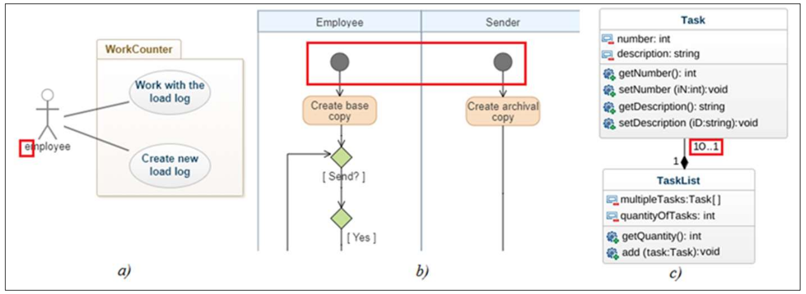


Fig. 4. Examples of mistakes found by system

7. Results

An analysis was carried out of 191 the work of IT students who create models of information systems based on an object-oriented approach to modeling. UCD module was tested on 70 student models, AD – 41, CD – 80. UCD module can detect 25 mistakes, AD – 26, CD – 26.

Fig. 5 presents percentage of mistakes found by system. System did not find semantic mistakes in students' CD models.

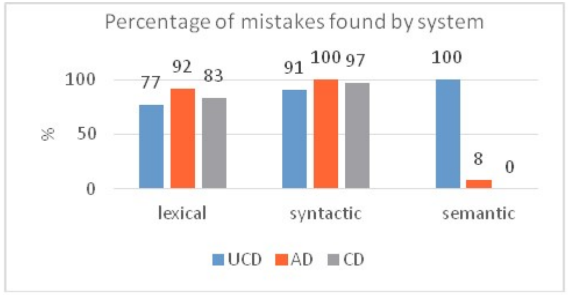


Fig. 5. Percentage of mistakes found by system

8. Discussion and Future Work

The current version of the UML diagram validation system solves the following tasks:

1) based on the exported XMI file, mistakes are searched in UCD, AD, and CD,
2) visualization of found mistakes and their display on exported diagram images.

It can be recommended for use by two categories of users:

1) students: to check models before submitting for teacher's grading.
2) teachers: to validate models of students.

For the future, we will work on the validation functions for teachers in order to qualify the degree of deviation of the student model from the task specification and to form the recommended score for the model.

9. Conclusion

We proposed a mistakes' classification with two dimensions and observed model mistakes in detail. We justified the choice of the tool for creating UML diagrams. Modules for validation of UCD, AD, CD were developed and realized. These modules were integrated into a system allowing package processing of model files. In the end, we tested the system on 191 student models.

References / Список литературы

[1]. Unified Modeling Language 2.5.1. (2017). Object Management Group, Available at: https://www.omg.org/spec/UML/About-UML/, accessed 03.09.2021.
[2]. Boberić Krstićev D., Tesendic D. Experience in Teaching OOAD to Various Students. Informatics in Education, vol. 12, 2013, pp. 43-58.
[3]. Koznov D.V. Methodology and tools for domain-specific modeling. Doctor Degree thesis. Saint-Petersburg, 2016, 430 p. (in Russian) / Кознов Д.В. Методология и инструментарий предметно-ориентированного моделирования. Диссертация доктора технических наук. СПб., 2016 г., 430 стр.
[4]. Baresi L., Morzenti A. et al. A logic-based approach for the verification of UML timed models. ACM Transactions on Software Engineering and Methodology, vol. 26, issue 2, 2017, article no. A7.
[5]. Daw Z., Mangino J., Cleaveland R. UML-VT: A Formal Verification Environment for UML Activity Diagrams. In Proc. of the MoDELS 2015 Demo and Poster Session co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015), 2015, pp. 48-51.

[6]. Bourque P., Dupuis R. et al. Guide to the software engineering body of knowledge. IEEE Software, vol. 16, no. 6, 1999, pp. 35-44.
[7]. Object Management Group. MDA Guide revision 2.0. 2014, Available at: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01, accessed 03.09.2021.
[8]. Reuter R., Stark T. et al. Insights in Students' Problems during UML Modeling. In Proc. of the IEEE Global Engineering Education Conference (EDUCON), 2020, pp. 592-600.
[9]. Matyokurehwa K., Makoni K.T. Students' Perceptions in Software Modelling Using UML in Undergraduate Software Engineering Projects. International Journal of Information and Communication Technology Education, vol. 15, no. 4, 2019, article no. 2.
[10]. Object Management Group. XML Metadata Interchange (XMI) Specification 2.5.1. 2015. Available at: https://www.omg.org/spec/XMI/2.5.1/PDF, accessed 03.09.2021.
[11]. Lima V., Talhi C. et al. Formal Verification and Validation of UML 2.0 Sequence Diagrams using Source and Destination of Messages. Electronic Notes in Theoretical Computer Science, vol. 254, 2009, pp. 143-160.
[12]. Chren S., Buhnova B. et al. Mistakes in UML Diagrams: Analysis of Student Projects in a Software Engineering Course. In Proc. of the IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), 2019, pp. 100-109.
[13]. Fernández-Sáez A.M., Caivano D. et al. On the use of UML documentation in software maintenance: Results from a survey in industry. In Proc. of the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2015, pp. 292-301.
[14]. Chytalová K. Catalog of errors in UML diagrams PB007 - software engineering I. Lasaris Lab, Faculty of Informatics, Masaryk University, 2018, Available at: https://drive.google.com/file/d/1J3_Ueb4E2YdAZjksryC4-F123Xqmyhkm/view, accessed 03.09.2021 (in Czech).
[15]. Bolloju N., Leung F.S.K. Assisting novice analysts in developing quality conceptual models with UML. Communications of the ACM, vol. 49, no. 7, 2006, pp. 108-112.
[16]. Delgado A., Dias A., Brito e Abreu F. Verification and Validation of UML Diagrams using Checklists. Available at: https://moodle.fct.unl.pt/pluginfile.php/22771/mod_folder/content/0/ArtigoGrupoB.pdf?forcedownload=1, accessed 03.09.2021.
[17]. Sabaliauskaite G., Matsukawa F. et al. An experimental comparison of checklist-based reading and perspective-based reading for UML design document inspection. In Proc. of the International Symposium on Empirical Software Engineering, 2002, pp. 148-157.
[18]. Rafe V., Rahmani A.T. Formal Analysis of Workflows Using UML 2.0 Activities and Graph Transformation Systems. Lecture Notes in Computer Science, vol. 5160, 2008, pp. 305-318.
[19]. Jensen K. A brief introduction to coloured Petri Nets. Lecture Notes in Computer Science, vol. 1217, Berlin, 1997, pp. 203-208.
[20]. IEEE 1044-2009 - IEEE Standard Classification for Software Anomalies. 2010, 23 p.
[21]. Vinarek J., Imko V., Hnetynka P. Verification of Use-Cases with FOAM Tool in Context of Cloud Providers. In Proc. of the 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2015), 2015, pp. 151-158.
[22]. Rational Rose. Available at: https://www.ibm.com/software/developer/rosexde/, accessed 03.09.2021.
[23]. UML-VT. Available at: http://www.cs.umd.edu/~rance/projects/uml-vt/, accessed 03.09.2021.
[24]. Woflan. Available at: https://www.win.tue.nl/woflan/doku.php/, accessed 03.09.2021.
[25]. Mokhati F., Gagnon P., Badri M. Verifying UML Diagrams with Model Checking: A Rewriting Logic Based Approach. In Proc. of the Seventh International Conference on Quality Software (QSIC 2007), 2007, pp. 356-362.
[26]. Cabot J., Claris'o R., Riera D. Verification of UML/OCL Class Diagrams using Constraint Programming. In Proc. of the IEEE International Conference on Software Testing Verification and Validation Workshop, 2008, pp. 73-80.
[27]. Dresden OCL. (2016), Available at: https://github.com/dresden-ocl/dresdenocl, accessed 03.09.2021.
[28]. GenMyModel. Available at: https://www.genmymodel.com/, accessed 03.09.2021.
[29]. Conradi R., Mohagheghi P. et al. Object-Oriented Reading Techniques for Inspection of UML Models – An Industrial Experiment. Lecture Notes in Computer Science, vol. 2743, 2003, pp. 483-500.
[30]. Naveed A., Ikram N. A novel checklist: Comparison of CBR and PBR to inspect use case specification. Communications in Computer and Information Science, vol. 558, 2015, pp. 109-125.

[31]. Baresi L., M. Pezzè. On Formalizing UML with High-Level Petri Nets. Lecture Notes in Computer Science, vol. 2001, 2001, pp. 276-304.
[32]. Araujo J., A. Moreira. Integrating UML Activity Diagrams with Temporal Logic Expressions. In Proc. of the 10th International Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'05), 2005, pp. 91-98.

Information about authors / Информация об авторах

Dmitry Igorevich VLASOV – student. His research interests include modeling languages, object-oriented programming, modeling tools, programming language toolkits.

Дмитрий Игоревич ВЛАСОВ – студент бакалавриата. Научные интересы включают языки моделирования, объектно-ориентированное программирование, средства моделирования, инструментарии языков программирования.

Tatiana Sergeevna GASHEVA – student. Her research interests include modeling, UML, object-oriented programming.

Татьяна Сергеевна ГАШЕВА – студент бакалавриата. Научные интересы включают моделирование, язык UML, объектно-ориентированное программирование.

Andrei Valerievich OTINOV – student. Research interests: object-oriented programming, modeling, information systems.

Андрей Валерьевич ОТИНОВ – студент. Научные интересы: объектно-ориентированное программирование, моделирование, информационные системы.

Nataliya Nikolaevna DATSUN – Candidate of Physical and Mathematical Sciences, associate professor of the Computer Science Department. Research interests: modeling languages, modeling tools, object oriented modeling.

Наталья Николаевна ДАЦУН – кандидат физико-математических наук, доцент, доцент кафедры математического обеспечения вычислительных систем. Сфера научных интересов: языки моделирования, средства моделирования, объектно-ориентированное моделирование.