UDC 004.052.2
DOI https://doi.org/10.32782/tnv-tech.2023.4.9

SELECTION OF A COMPUTATIONAL PROCESS MODELING TOOL FOR IMPROVING SOFTWARE QUALITY

Paulin O. M. *– Doctor of Technical Sciences,*
Associate Professor at the Department of Software Engineering
National University "Odesa Polytechnic"
ORCID ID: 0000-0002-2210-8317

Nikitchenko M. I. *– Postgraduate Student*
National University "Odesa Polytechnic"
ORCID ID: 0009-0007-9560-7057

A pressing problem in the IT sector is obtaining software products of proper quality. Software modeling tools play an important role in solving this problem. The most widespread tools are the following: technologies for designing software products based on automated models built on finite state machines (FSMs), Petri nets, and the unified modeling language (UML).

The aim of this paper is to improve the quality of software by applying the best of these tools at an early stage of software design.

The paper reviews and analyzes modern means (approaches, methods and tools) for modeling software with the aim of improving its quality, analyzing 30 sources over the past 25 years. It is shown that over the years, the scope of applications of these tools has expanded, but nothing new has emerged in theoretical terms. The above-mentioned tools are compared with each other by the criterion of the greatest expressive power of the language describing the modeling process. It is shown that Petri nets (PNs) have the greatest expressive power. At the same time, we propose to use PNNs at the beginning of the design of the software, i.e., at the stage of algorithmization (development of the computational process). This avoids many errors that can be eliminated at the stage of program debugging only by trial and error, which greatly lengthens the debugging process.

To demonstrate the capabilities of the tools under consideration, each of them builds its own model of the same insertion sorting process.

Key words: *software product, state machine model, FA technology, SWITCH technology, Petri net, Unified Modeling Language, review, analysis, selection criterion, expressive power, process modeling language, algorithmization, computational process, trial and error method, program debugging, insertion sort.*

Паулін О. М., Нікітченко М. І. Вибір засобу моделювання обчислювальних процесів для підвищення якості програмного забезпечення

Актуальною проблемою у сфері ІТ є отримання програмних продуктів (ПП) належної якості. Важливу роль вирішенні цієї проблеми грають інструменти моделювання ПП. Найбільше поширення отримали такі інструменти: технології проектування ПП на основі автоматних моделей, побудованих на кінцевих автоматах (КА), мережі Петрі та уніфікована мова моделювання (UML).

Метою даної є підвищення якості ПП за рахунок застосування найкращого зі згаданих інструментів на ранньому етапі проектування ПП.

У роботі проводиться огляд та аналіз сучасних засобів (підходів, методів та інструментів), моделювання ПП з метою підвищення його якості, при цьому аналізуються 30 джерел за останні 25 років. Показується, що за ці роки сфера додатків цих інструментів розширилася, проте в теоретичному плані нового нічого не з'явилося. Згадані вище інструменти порівнюються між собою за критерієм найбільшої виразної потужності мови опису процесу моделювання. Показується, що найбільшою виразною потужністю мають мережі Петрі (МП). У той самий час нами пропонується використовувати МП на початку проектування ПП, тобто. на етапі алгоритмізації (розробки обчислювального процесу – ОП). Це дозволяє уникнути безлічі помилок, які усуваються на етапі налагодження програм тільки методом проб і помилок, що сильно подовжує процес налагодження.

Для демонстрації можливостей розглянутих інструментів для кожного з них будується своя модель одного і того ж ОП сортування вставками.

Ключові слова: *програмний продукт, автоматна модель, КА-технологія, SWITCH-технологія, мережа Петрі, уніфікована мова моделювання, огляд, аналіз, критерій вибору, виразна потужність, мова опису процесу моделювання, алгоритмізація, обчислювальний процес, метод проб та помилок, налагодження програми, сортування вставками.*

Introduction. Development and implementation of quality software products (SP) becomes a key success factor for enterprises and organizations. However, the increasing complexity of software products and intensification of their development lead to the growing number of errors in programs and significantly increase the time of their debugging.

The quality of software and software in general is defined and regulated by ISO 12207, ISO 9000, CMM and other standards [1], which provide systematization and unification of quality criteria. These standards help developers and customers to evaluate and control the quality of software at different stages of its development and operation. The development of quality PP and software in general is very topical. This paper reviews and analyzes modern sources to select the best tool for software quality improvement. The most common tools are considered: finite automaton (FA) based models (Harel state diagrams [2] and SWITCH-technology [3]), Petri nets [4], Unified Modeling Language (UML) [5]. Each of these tools and their clones has its own advantages and disadvantages.

The aim of the work is to improve the quality of the software in general by choosing the best of the above mentioned tools and moving to the earliest stage of the software development – the stage of algorithmization.

In order to achieve this goal, the following objectives are addressed:

- review of sources on the above instruments;
- analysis of these tools and selection of the best of them according to the criterion of maximum expressive power of the language of description of modeling processes;
- Demonstration of the tools' operation using the example of the computational process of sorting by inserts and comparison of their capabilities.

Main part. Let us consider the above-mentioned directions of software quality improvement, namely: the use of automata models, Petri nets, UML.

1. Automata model. The automata model is represented by two approaches: Harel state diagrams [2] and the SWITCH technique [3]. The common feature of these approaches is the use of the finite automaton (FA) model, but there are significant differences: the first approach uses transition diagrams, while the second uses state diagrams. In addition, the authors have different stages of program construction: the first approach uses the stages from algorithmization to program writing and maintenance, while the second approach uses the stages of program verification and debugging. Let us further note that the use of the switch construct allows structuring and modifying programs and ensuring their isomorphism (pictorial equivalence) with the specification (transition graph).

The theory of finite automata as well as their varieties is presented in [6]. A finite automaton is a *mathematical model* of a discrete control process (transformation of discrete information); the most common finite automata are Mealy and Moore automata. The *state* of an automaton is a set of values of the automaton's memory elements; the current set represents some prehistory of the automaton's behavior as a result of successive exposure to input symbols. Automata have the ability to retain the previous state, so they are called *automata with memory*. An automaton without memory is called a *trivial automaton* or a *combinational circuit*.

Literature analysis of automata models. A review of research conducted over the last 25 years is presented, showcasing a diverse range of approaches to the utilization of automata models for addressing complex problems across various domains. The analysis consists of examining the main concepts, methods and applications of automata models in various fields.

The paper [7] describes a tool for creating and testing deterministic FAs with a graphical interface. This tool allows users to create automata and test input strings in real time.

The article [8] is devoted to the construction of a neural network with a minimum number of neurons, which is necessary for modeling any FA with m states; the lower and upper bounds of this minimum number are determined and found to be linearly dependent on m under certain constraints.

The paper [9] considers FAs that are equivalent to right-linear context-free grammars and represent the lowest level in Chomsky's hierarchy. Standard automata problems (emptiness, universality, equivalence, etc.) are discussed. The paper deals with the issues of descriptive and computational complexity of FAs, presenting an overview of the main ideas and the general picture in this area.

The paper [10] considers the problem of automatic correction of spelling errors in text during machine translation and information retrieval based on FA. The correction method is language-independent and requires only a dictionary and text data to build a language model.

The paper [11] discusses the use of FA in parallel computing, especially in the context of multicore processors. A new kind of automaton called a Simultant Finite Automaton (SFA) is presented, which is designed with efficient parallel processing in mind. A regular expression matching tool based on the SFA was created, which in typical cases achieved significant speedup (by a factor of 10 or more) on a computer with two six-core processors.

Summary on the conducted analysis of automata models. Automata models are a powerful tool for modeling and analyzing programs and their systems in order to write high-quality SPs. Different approaches to the implementation of FAs provide different advantages for designing software systems, allowing you to choose an approach depending on the type of task. Modern graphical interfaces facilitate work with FA and speed up the analysis of SPs.

Note the proposal to use neural networks for modeling the FA itself. The properties and complexity of FA are still discussed. An interesting application is the task of correcting spelling errors in texts using FA. A theoretical novelty is proposed, a new kind of automaton called a "simultaneous finite automaton" (SFA), which is designed with efficient parallel data processing in mind.

Example. Let's consider the representation of the insertion sorting program [6] by a finite automaton. The initial array contains n elements numbered from 0. Let us distinguish the operators of the program by dividing them into two groups: unconditional X and conditional Y. The lists of operators are given in Table 1; the content of each operator is also given here.

According to the program [12] and in accordance with the notations (Table 1), the graph-scheme of the automaton (fig. 1a), or, more precisely, its operational component, is constructed. The operators of input of the initial array and output of the sorted array are subroutines and are not considered in detail.

For complete construction of the automaton it is necessary to construct its control component. For this purpose, we can use Baranov's method [6] of constructing a control

Table 1

Operators and their contents

Оператор	Содержание оператора
Y_1	Input of array A (numbering of elements – from zero)
Y_2	Enter the number n of elements of array A
Y_3	i := 1 – initial value of the external loop parameter
Y_4	key := arr[i] – auxiliary variable
Y_5	j:=i-1 – count, inner loop
Y_6	arr[j+1] := arr[j] – element shift
Y_7	j := j-1 – initial value of the inner loop parameter
Y_8	arr[j+1] := key – insert element to the appropriate place
Y_9	i:=i+1 – count, outer loop
Y_{10}	Output sorted array A'
X_1	Checking the condition i ≤ n-1 – end of the outer loop
X_2	Checking the condition j≥0 & arr[j] > key – end of inner loop

automaton, which contains the following stages: construction of the graph-scheme of the algorithm (GSA); markup of the GSA with state symbols; construction of the automaton graph. In turn, the graph of the automaton contains several sub-stages, including: construction of the transition table; minimization of the automaton, etc.

In this case GSA is interpreted as Moore's automaton, i.e. its vertices are marked with state symbols. Fig. 1b shows the graph of Moore's automaton constructed by the GSA of sorting by inserts.

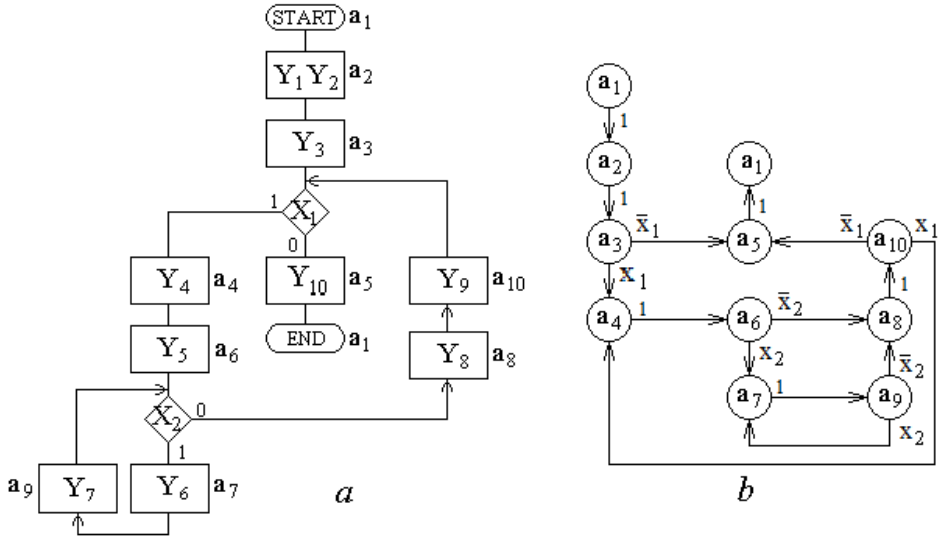

Fig. 1. Algorithm for sorting by inserts:
a – marked graph diagram; b – transition diagram

Here we cut short the process of synthesizing a control automaton, since it is a separate complex task. However, for evaluating the expressiveness of the language of graph schemes, it is quite enough.

2. Petri net. A Petri net (PN) [4] is a bipartite graph that includes two types of elements: positions p and transitions t; respectively, there are 2 types of links between them: $p \rightarrow t$ and $t \rightarrow p$.

Positions denote conditions that allow an action to be performed or an event to occur when a chip is present in a position. *Transitions denote* actions/events that change the state of the system, with the chip moving to the position associated with the transition, which means the formation of a new condition. For a transition to be executed, all positions that have connections to this transition must be active, i.e. have a token each. A *scenario* is the execution of certain actions (in the general case – the occurrence of events) in the presence of necessary conditions.

Note the important rule: transitions and positions alternate.

In general, each element (position and transition) of an PN can have several input and output links, but there are simplified PNs in which restrictions are imposed on the number of inputs/outputs.

Livability is a property of a Petri net which means that the system does not get stuck in an endless loop of performing the same transitions, but is operational.

Petri nets are used to represent and analyze dynamic systems and to model various processes, such as business processes, manufacturing processes, or the processes of interaction between the components of a SP. They help to understand how a system works, identify problems and improve its performance.

Literature analysis of PN. In the last 20–25 years, serious attention of specialists has been paid to both conventional [14–17, 19–21] and colored PNs (CPNs) [13; 18]. The publications consider various areas of PN applications: modeling of dynamic systems on the basis of "classical" PN and its derived models [14]; verification of safety-critical systems [20] and software systems, e.g., firmware for a robot [16]; analysis and modification of logic for multi-agent systems [21]; design/management of construction projects [17]; identification of partially observable systems of discrete events [19]. The paper [18] attempts to illustrate many aspects of software development, to point out some aspects of Petri nets that have been used or can be used to solve software development problems, and to identify new software development problems that can be solved with the help of PN modeling results.

Some approaches to the use of PN are also considered. The paper [22] provides a comparative analysis of the methods, focusing on the creation of executable models of software architectures and identifying research perspectives in this area. The article [23] describes the application of PNs to verify the integrity of rule-based systems using their structural properties. In [24] a combined approach combining WF-networks and PNs for modeling the dynamics of software systems is investigated.

Summary of the analysis conducted on Petri nets. An overview of PNs and their diverse applications, which include systems modeling, project management, software development, and other areas, is provided. Approaches to using PNs, such as creating executable software models and checking the integrity of systems, as well as tools that use PNs to manage hybrid systems, are reviewed.

So, PNs are a powerful and flexible tool for analyzing and modeling dynamical systems, and their applications continue to expand.

Example. Let's consider modeling of the algorithm of sorting by simple insertions with the help of SP. The PN is shown in fig. 2. Table 2 describes positions and Table 3 describes transitions.

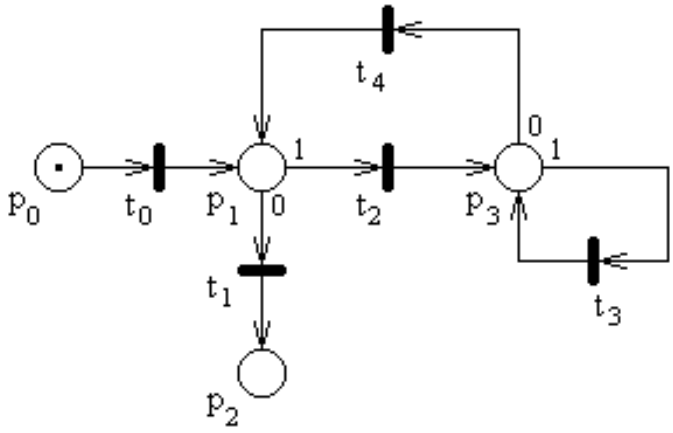

Fig. 2. Petri net for the algorithm of sorting by simple insertions

Note that the positions are combined (Table 3), which is determined by the need to alternate between positions and transitions.

Table 2

Positions and their meanings

Position	Significance
p_0	Beginning
p_1	i≤n
p_2	The end
p_3	j≥0 & arr[j] > key

Table 3

Transitions and their meanings

Transition	Significance
t_0	input A, n; i:=2
t_1	output A’
t_2	key:=ai; j:=i–1
t_3	arr[j+1] := arr[j]; j:=j–1
t_4	arr[j+1] := key; i:=i+1

To model this SP, scenarios have been developed that cover all possible chip movement paths; in the initial state, the chip is in position p_0. Near each position, except the start and end positions, are placed values (0 and 1) that define specific paths of the chip's movement. The scenarios are as follows: 1) p_3, t_3, p_3, t_3…; 2) p_1, t_2, p_3, t_4, p_1, t_2, p_3, t_4, …; 3) p_0, t_0, p_1, t_2, p_3, t_4, p_1, t_1, p_2.

3. Unified Modeling Language. UML [5] is a standard notational language for visualization, design, documentation and specification of software systems and other complex systems. UML provides versatile tools for describing various aspects of a system, including its structure, functionality, behavior, and interaction with the environment.

UML was developed to provide a common and understandable way of communication between developers, analysts, designers, and others involved in the development of software and information systems. It provides graphical symbols and rules for creating diagrams that allow for modeling a variety of aspects of a system.

The main types of UML diagrams include class diagrams, sequence diagrams, state diagrams, and activity diagrams. Each of these diagrams aims to visualize specific aspects of a system and helps developers to better understand its structure and functioning.

Below we will review and analyze articles and other sources of information that contain one or another feature of UML.

Review and analysis of literature sources on UML. Initially, we should pay attention to the book [5], which provides the basics of UML in a two-color format with examples of applications in different domains. It includes an overview of UML and a gradual introduction to the language, providing examples of its application in different domains. The content has been updated to match UML 2.0, including new features, interfaces, and changes to diagrams.

The papers presented below discuss applications of UML diagrams in software engineering with a focus on diagram consistency. UML consistency management is discussed in [25] with parameter-based classification of methods and comparison of methods to identify current trends and issues. The literature review in [26] on the use of UML diagrams shows their applicability for design and modeling, with class diagrams proving to be the most common. The paper [27] presents a systematic corpus of 119 consistency rules useful in UML-based software development. In [28], a semantic conflict detection method for UML class diagrams is proposed.

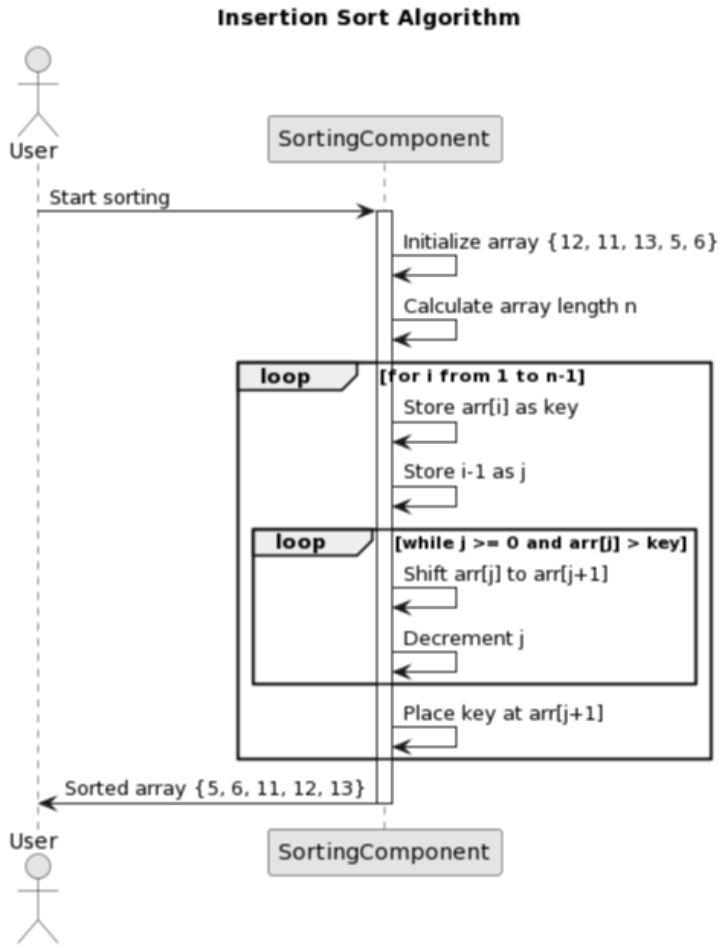

Fig. 3. Algorithm of sorting by simple inserts using UML

Next, approaches to the use of UML are analyzed. In [29], the authors describe consistency rules between an action diagram and a class diagram, translating them into logical predicates and applying them to a case study example. In [30], the authors present 11 consistency rules for model checking between the most commonly used UML diagram types. In [31], the authors propose an automatic approach for analyzing UML-based designs using logical reasoning, allowing hundreds of online model repository designs to be validated.

Summary of the UML analysis performed. A review of the literature on the application of UML in software engineering has been conducted. The focus is on the use of logical methods for consistency and efficient analysis of UML models, and consistency rules and techniques for managing them are discussed. The articles provide a clear introduction to object-oriented analysis and design using UML. These resources are valuable for both novices and experienced software engineers, helping them to improve their modeling and software development skills in the context of UML.

Example. Let's consider a program of sorting by simple inserts [12] and describe it using UML. Let's choose the most appropriate diagram – the sequence diagram, which will describe each step of the program execution (Fig. 3).

Summary and conclusions. The most widespread tools of software modeling to improve its quality are: automata model in two variants proposed by Harel and Shalyto, Petri nets, UML. All the tools, except SWITCH technology, are used in software modeling and only in SWITCH technology modeling is carried out at the stage of algorithmization. And only the tool using switch program design is brought to the level of technology, but its application area is embedded tools intended for automation of industrial objects.

As the review has shown, the latest research in the field of automata models is devoted only to 5 articles, in the field of Petri nets – 13 articles, in the field of UML – 7 articles. It follows that the interest to automata models is decreasing. Petri nets are of the greatest interest for researchers. We consider UML as a tool to be poorly developed, as evidenced by the presence of 14 types of diagrams and 119 rules for their coordination.

Our approach coincides with the approach in SWITCH-technology – we believe that verification and debugging of a program should be carried out at the earliest stage of its development, i.e. at the stage of algorithmization. Further we believe that Petri nets possess the greatest expressive power of the language of description of modeling processes.

So, we consider the proposed approach to be the best. This conclusion is confirmed by modeling the sorting process by inserts for all four tools.

BIBLIOGRAPHY:

1. "ISO/IEC/IEEE International Standard – Systems and software engineering -- Software life cycle processes," in ISO/IEC/IEEE 12207:2017(E) First edition 2017-11, vol., no., pp.1-157, 15 Nov. 2017, doi: 10.1109/IEEESTD.2017.8100771.

2. Harel D. Modeling reactive systems with statecharts: The statemate approach. New York : McGraw-Hill, 1998. 258 p.

3. Shalyto A. Software automaton design: Algorithmization and programming of problems of logical control. Journal of Computer and Systems Sciences International. 2000.

4. Radford P. Petri Net Theory and the Modeling of Systems. The Computer Journal. 1982. Vol. 25, № 1. P. 129. URL: https://doi.org/10.1093/comjnl/25.1.129.

5. Booch G. The unified modeling language user guide. Second ed. Upper Saddle River, NJ : Addison-Wesley, 2005. 475 p.

6. Baranov S. Automata. Logic Synthesis for Control Automata. Boston, MA, 1994. P. 1–66. URL: https://doi.org/10.1007/978-1-4615-2692-6_1.

7. Simulation and Testing of Deterministic Finite Automata Machine / K. B. Vayadande et al. International Journal of Computer Sciences and Engineering. 2022. Vol. 10, no. 1. P. 13–17. URL: https://doi.org/10.26438/ijcse/v10i1.1317.

8. Alon N., Dewdney A. K., Ott T. J. Efficient simulation of finite automata by neural nets. Journal of the ACM (JACM). 1991. Vol. 38, № 2. P. 495–514. URL: https://doi.org/10.1145/103516.103523.

9. Holzer M., Kutrib M. Descriptional and computational complexity of finite automata–A survey. Information and Computation. 2011. Vol. 209, № 3. P. 456–470. URL: https://doi.org/10.1016/j.ic.2010.11.013.

10. Izakovič L. Using Finite-state Automata for Text Lexicons Building. Glottotheory. 2008. Vol. 1, № 1. URL: https://doi.org/10.1515/glot-2008-0003.

11. Sinya R., Matsuzaki K., Sassa M. Simultaneous Finite Automata: An Efficient Data-Parallel Model for Regular Expression Matching. 2013 42nd International Conference on Parallel Processing (ICPP), Lyon, France, 1–4 October 2013. 2013. URL: https://doi.org/10.1109/icpp.2013.31.

12. Introduction to Algorithms / V. J. Rayward-Smith et al. The Journal of the Operational Research Society. 1991. Vol. 42, no. 9. P. 816. URL: https://doi.org/10.2307/2583667.

13. Kristensen L. M., Simonsen K. I. F. Applications of Coloured Petri Nets for Functional Validation of Protocol Designs. Transactions on Petri Nets and Other Models of Concurrency VII. Berlin, Heidelberg, 2013. P. 56–115. URL: https://doi.org/10.1007/978-3-642-38143-0_3.

14. Valk R. Petri Nets as Token Objects. Application and Theory of Petri Nets 1998. Berlin, Heidelberg, 1998. P. 1–24. URL: https://doi.org/10.1007/3-540-69108-1_1.

15. David R., Alla H. Petri nets for modeling of dynamic systems. Automatica. 1994. Vol. 30, № 2. P. 175–202. URL: https://doi.org/10.1016/0005-1098(94)90024-8.

16. Nandanwar M. Formal Verification with Petri Nets. TU Kaiserslautern – Computer Science – Embedded Systems Group: Welcome! URL: https://es.cs.rptu.de/publications/datarsg/Nand22.pdf (date of access: 15.08.2023).

17. Lin C.-P., Dai H.-L. Applying Petri Nets on Project Management. Universal Journal of Mechanical Engineering. 2014. Vol. 2, № 8. P. 249–255. URL: https://doi.org/10.13189/ujme.2014.020801.

18. Kristensen L. M., Jørgensen J. B., Jensen K. Application of Coloured Petri Nets in System Development. SpringerLink. URL: https://link.springer.com/chapter/10.1007/978-3-540-27755-2_18.

19. Riera B. Identification of discrete event systems using ordinary Petri nets. Academia.edu – Share research. URL: https://www.academia.edu/ 64731436 /Identification_of_discrete_event_ systems_using_ordinary_Petri_nets.

20. Integrated formal verification of safety-critical software / N. Ge et al. International Journal on Software Tools for Technology Transfer. 2017. Vol. 20, no. 4. P. 423–440. URL: https://doi.org/10.1007/s10009-017-0475-0.

21. Petri net and rewriting logic based formal analysis of multi-agent based safety-critical systems / A. Boucherit et al. Multiagent and Grid Systems. 2020. Vol. 16, no. 1. P. 47–66. URL: https://doi.org/10.3233/mgs-200320.

22. Aliee F. S. A Comparison of Petri Net Based Approaches Used for Specifying the Executable Model of Software Architecture. ResearchGate. URL: https://www.researchgate.net/publication/221615992_A_Comparison_of_Petri_Net_Based_Approaches_Used_for_Specifying_the_Executable_Model_of_Software_Architecture.

23. Agarwal R., Tanniru M. A Petri-Net based approach for verifying the integrity of production systems. International Journal of Man-Machine Studies. 1992. Vol. 36, № 3. URL: https://doi.org/10.1016/0020-7373(92)90043-k.

24. Suprunenko O. O. COMBINED APPROACH TO SIMULATION MODELING OF THE DYNAMICS OF SOFTWARE SYSTEMS BASED ON INTERPRETATIONS OF PETRI NETS. KPI Science News. 2019. № 5-6. P. 43–53. URL: https://doi.org/10.20535/kpi-sn.2019.5-6.174596.

25. Sulaiman N., Syed Ahmad S. S., Ahmad S. Logical Approach: Consistency Rules between Activity Diagram and Class Diagram. International Journal on Advanced Science, Engineering and Information Technology. 2019. Vol. 9, № 2. P. 552. URL: https://doi.org/10.18517/ijaseit.9.1.7581.

26. Abdulsahib M. A methods of ensuring consistency between UML Diagrams. ResearchGate.URL:https://www.researchgate.net/publication/326901133_A_methods_of_ensuring_consistency_between_UML_Diagrams.

27. Khan A. H., Porres I. Consistency of UML class, object and statechart diagrams using ontology reasoners. Journal of Visual Languages & Computing. 2015. Vol. 26. P. 42–65. URL: https://doi.org/10.1016/j.jvlc.2014.11.006.

28. UML models consistency management: Guidelines for software quality manager / R. S. Bashir et al. International Journal of Information Management. 2016. Vol. 36, no. 6. P. 883–899. URL: https://doi.org/10.1016/j.ijinfomgt.2016.05.024.

29. UML Diagrams in Software Engineering Research: A Systematic Literature Review / H. Koç et al. Proceedings. 2021. Vol. 74, no. 1. P. 13. URL: https://doi.org/10.3390/proceedings2021074013.

30. A systematic identification of consistency rules for UML diagrams / D. Torre et al. Journal of Systems and Software. 2018. Vol. 144. P. 121–142. URL: https://doi.org/10.1016/j.jss.2018.06.029.

31. Costa V., Monteiro R. Detecting Semantic Equivalence in UML Class Diagrams. ResearchGate. URL: https://www.researchgate.net/publication/ 289465971_Detecting_semantic_equivalence_in_UML_class_diagrams.