Applied Computer Systems

ISSN 2255-8691 (online)
ISSN 2255-8683 (print)
May 2019, vol. 24, no. 1, pp. 75–81
https://doi.org/10.2478/acss-2019-0010
https://content.sciendo.com

Definition of the Criteria for Layout of the UML Use Case Diagrams

Olga Filipova[1*], Oksana Nikiforova[2]
[1, 2]*Riga Technical University, Riga, Latvia*

Abstract **– Nowadays, the topicality and applicability of model-driven development in the object-oriented development approach has increased, so it is important that created models and diagrams display not only the content, but also visually reflect information. Transparent diagram placement that influences work productivity is important for displaying information. Manual layout of diagrams is a time-consuming activity, which can also be ineffective, so in this paper the application of UML use case automatic layout is reviewed. The paper also examines the requirements of use case diagrams and placement criteria, which will serve as a basis for the creation of an automatic use case diagram layout algorithm.**

Keywords **– Automatic layout algorithms, layout guidelines, UML, use case diagram.**

I. Introduction

Currently, the model-driven development approach is relevant and widely used. This approach is based on models that are presented in graphical form as diagrams. Large system modelling enables software developers to understand its structure, behaviour and its core elements. For these purposes, spatial placement of Unified Modelling Language (UML) diagram elements is used to determine how well the software system will be understood. The more efficient the placement of the elements, the easier it is to understand the essence of the diagram and the more efficient UML usage is. In order to achieve it, the diagram modeller must place the elements of the diagram in such a way that its readability will increase.

To improve productivity, diagrams can be generated automatically, which in turn can release the diagram modeller from unnecessary work and offer a better result than manual element placement during diagram design, transformation, or during diagram export/import. In order to create an algorithm for automated diagram element placement, it is necessary to define the principles, according to which the elements of the diagram should be placed.

Not only content, but also the placement of the elements is important while creating diagrams, but there are no such graphical deployment solutions for the diagrams that would provide the most essential requirements that are needed for humans. All existing solutions and algorithms have their own weaknesses, so the present research will examine diagram element placement requirements, their problems.

In 2017, a group of researchers performed an extended systematic literature review due to which the trend for the future development of UML notation was determined. In [1], 33 empirical papers published between 2009 and 2016 related to the field of UML model were extracted from electronic databases and considered for review (see Table I).

TABLE I
RESULTS PER TYPE OF DIAGRAM [1]

Available diagram	Number of studies	Percentage
Class diagrams	19	47.5
Sequence diagrams	6	15.0
Use case diagrams	5	12.5
UML diagrams	3	7.5
Activity diagrams	4	10.0
State chart diagrams	2	5.0
Component diagrams	1	2.5

The study shows that in the future the following diagrams will be mostly developed: class, sequence and use case diagrams.

Use cases are the basic concept of several object-oriented development methods [2]. They are being used throughout the analysis and design phases during software development life cycle. Use cases represent what the customer wants system to do, in other words, customer system requirements. At the high level of abstraction, use cases show which purposes the developed system is meant for.

To support use case diagrams in future software development, in this paper we present guidelines for the layout of UML use case diagrams, the automatic layout analysis of the most popular UML diagram tool and the use case diagram guidelines, which will be used in further work for automatic layout algorithm creation.

The present paper is structured as follows: In the next section, UML diagrams are classified for the task of their layout. Section 3 presents the summary of use case diagram layout guidelines. Section 4 provides the information on use case diagram quality measurement. Section 5 summarises the information about the related work. In conclusion of the paper, the authors discuss the present research and state the research directions for the future.

II. Classification of UML Diagrams for the Task of Their Layout

One of the tasks of software development project is to present different aspects of the system before developing the software solution for the required system. To solve this task, system modelling became one of the most important activities during software development. System modelling gives software developers an ability to understand system behaviour, structure and its separate elements. System modelling is a way of thinking about problems using models, which are based on

[*]Corresponding author's e-mail: olga.filipova_2@edu.rtu.lv

real-world ideas. Models are useful for understanding problems, communicating with everyone involved within the project (customers, domain experts, analysts, designers etc.), as well as for modelling enterprises, preparing documentation and designing programs and databases. Modelling promotes a better understanding of requirements, more clear designs and more maintainable systems [3].

Usually a system model is presented as a set of diagrams, where specific notation is defined for each diagram and regulates diagram syntax and semantics. As far as system models are abstractions that portray the essentials of a complex problem or structure by filtering out nonessential details, models are making the problem easy to understand. Thus, the systematic approach to element placement within the diagram, which is specified as a task of diagram layout, plays an important role in completing the task of system modelling, which harmonizes with the area of graph theory.

Nowadays one of the leading notations used in system development is UML (Unified Modelling Language), which is declared as a standard for presentation of software system model and provides a notation, which grows from analysis through design into implementation in object-oriented programming languages.

As a notation of system modelling for different aspects of the system, UML introduces different types of diagrams, which can describe system from different points of view. UML 2.x version distinguishes 14 diagram types (abstract examples of all 14 types are shown in Fig. 1).

We can assume that all diagrams are represented in a graph form to some extent – diagram consists of nodes, which are connected with edges in some manner. However, different diagram types can have different structure: a diagram can have different types of nodes or edges, diagram should be constructed in some special manner.

The simplest presentation of elements from the perspective of graph layout has a deployment diagram. It has two types of elements, one of them is a node, which describes physical place of system deployment, and the other one is an edge between nodes. The same is within the object diagram, where regarding UML notation, diagram has two types of elements – objects and links between them. Layout algorithms used for regular graphs can be applied to this group of UML diagrams. However, diagrams having different types of edges or nodes must be analysed separately from diagrams with one type of edges and one type of nodes because extra type of elements can require extra conditions be taken into consideration in diagram layout [3].

Although the use case diagram has simple structure, where actors communicate with use cases by relationships, the diagram can have additional conditions on actors' placement according to the set of use cases, which in turn specifies the boundary of the system. State machine, also known as state chart, has the same structure as the use case diagram. The UML communication diagram has similar structure, but a bit complicated – objects are connected by edges having complicated structure. The link is presented as a connector, which is anchored with the message object that passes to another object. Therefore, in addition to the rules of element placement according to graph structure, the distance between objects, the placement of the name of message and, if possible, the orthogonal layout of the diagram itself should be taken into consideration to layout the diagram. Composite structure diagram has the same layout requirements: Diagram has two types of nodes, which relate to one type of edges and names placed on them. These four diagram types can be grouped into one requiring specific regulations for graph nodes.

The UML class diagram has one type of nodes (system classes) and several types of edges (relationships), where several aspects of diagram layout should be taken into consideration, e.g., orthogonality, parents up, tidy up, etc. Component diagrams have one type of nodes and several types of edges like class diagram, but in this case different types of edges have the same semantics and are used for improving readability of the diagram. The same condition applies to a profile diagram, which is a rarely used structure diagram in any specification and describes a lightweight extension mechanism to the UML by defining custom stereotypes, tagged values, and constraints. These three diagram types can be joined into diagrams, which require specific regulations for graph edges.

Logically, a package diagram consists of one type of nodes that represent packages and several types of edges that show how packages relate to each other. In most cases, package diagrams are part of other diagram types – this adds more node types to diagram. Activity diagram also has several types of nodes: activity, entry point and exit point; and several arc types. However, the edge structure is complicated: edges can split into several flows and then join into one, also edges can be split into alternative flows. These two types of diagrams can be joined into diagrams, which require specific regulations for graph edges and nodes.

Sequence diagram has even more special structure: all the objects are allocated horizontally at the top of diagram, except objects created during system operating, each of objects has its activity time, and the sequence of messages is shown by links demonstrating its control flows. Interaction overview diagram has the same structure as an activity diagram, but instead of activities, nodes of interaction overview diagram can have separate sequence diagrams, also edges have a simpler structure – no flow separation and joining. Timing diagram cannot be considered as a graph because it has no nodes. These three diagram types can be joined into diagrams with specific conditions for general structure and should be analysed separately in the context of diagram layout.

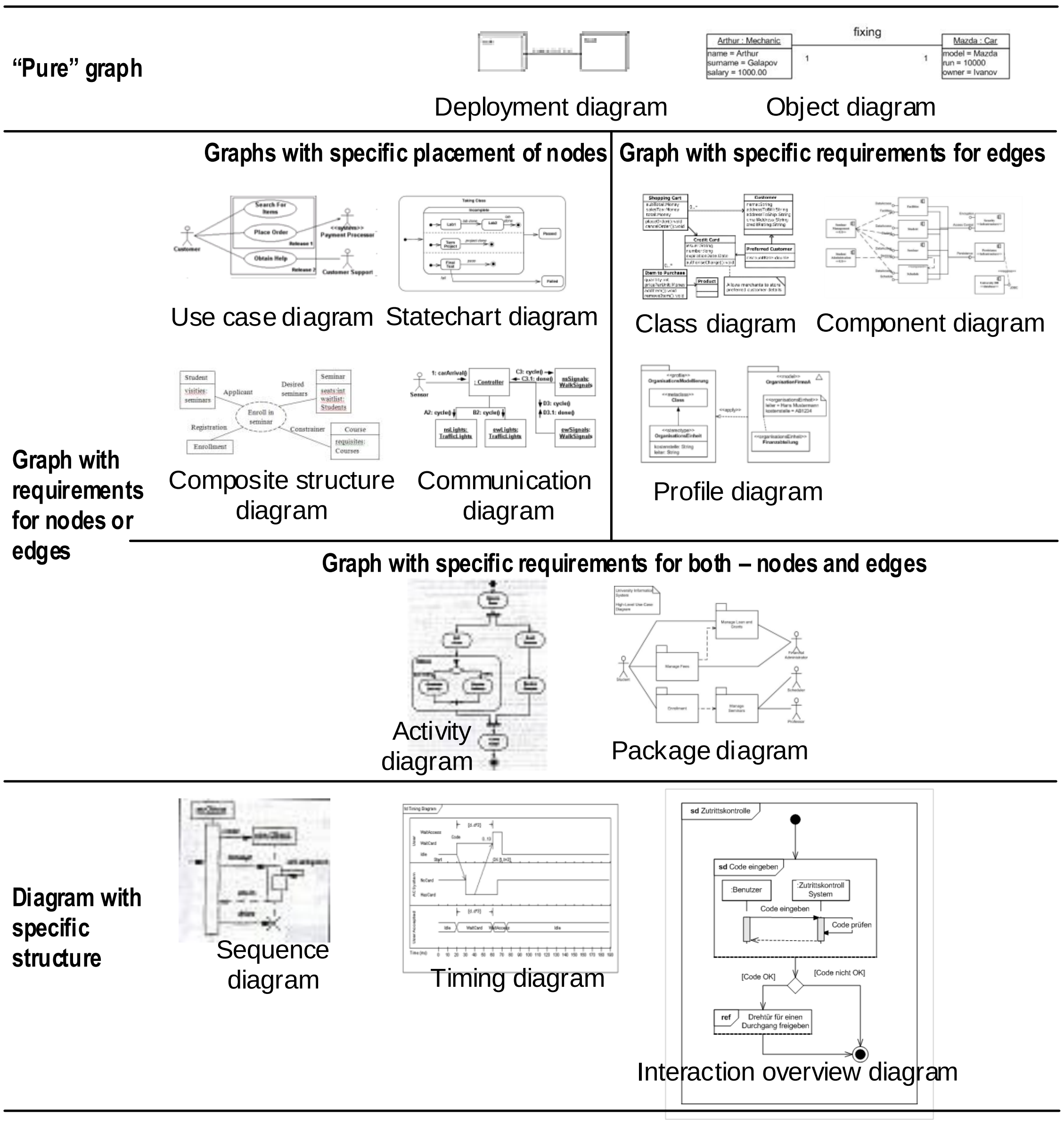

Fig. 1. Classification of UML diagrams in the context of their layout (modified from [3]).

Taking into consideration the specific requirements to structure, placement of elements and construction principles described above, UML diagrams are grouped as it is shown in Fig. 1. Tilley and Huang in [4] discuss that UML diagram efficiency depends on UML syntax and semantics, layout of UML diagram elements and domain knowledge. Therefore, the diagram layout is an essential factor for diagram reading and comprehension, which should be studied based on diagrams aesthetics trying to explain effective diagram construction principles and developing algorithms for automatic diagram layout.

Not all 14 different types of UML diagrams are used on a regular basis when documenting systems and/or architectures. The Pareto principle seems to apply in terms of UML diagram usage as well – 20 % of the diagrams are being used 80 % of the time by developers. The common ones in software development are class diagrams, sequence diagrams and use case diagrams [1]. All these three diagrams belong to different groups in the context of their layout according to classification in Fig. 1, so far, they should be analysed separately. The authors started their research on layout of UML diagrams with class diagram in 2011 offering the algorithm based on application of genetic algorithm [5]. The algorithm gave good results on matching requirements for class diagram layout, but worked very slow – about 10 minutes to layout 40 classes. The research for class diagram continued and the algorithm based on principle of modularity was offered in [6]. It was quite fast, but not all the criteria were supported. In parallel, criteria for layout of UML sequence diagram and layout abilities offered in UML modelling tools have been studied since 2014. As a result, the algorithm for sequence diagram layout has been developed to support 14 criteria out of 16 stated. The main results of this research were published by authors in [6]–[8]. So far, the focus of this paper is placed on the layout of UML use case diagrams.

III. Guidelines for Use Case Diagram Layout

Reorganising diagram elements may take a lot of time that would be better spent on content improvement and addition of missing information. Therefore, in this case, it is useful to leave diagram element placement operation to the automatic element placement algorithm. When the use case diagram consists of a few elements, it can be improved manually because it will not take much time. In turn, use case diagrams often contain much more elements, so the application of the algorithm becomes more topical.

Since the creation of use case diagrams, a few authors have adopted several parts of this idea, especially Cockburn [9] offering guidelines for use case diagrams. Use cases are adopted as part of the UML standard and their diagrams are most widely used in parts of this language. Several other books and articles were published for use cases, describing a variety of systems – not just software, but also business systems (such as embedded systems). Nowadays, the Internet of Things and Industrial Internet choose use cases [10]. Prior to the development of the diagram placement algorithm, the requirements to be met by this algorithm were defined. Until now only two books [9] and [11] are known to describe UML use case diagrams, because placement and diagram perception questions are rarely discussed in UML.

After performing a literature survey, it was determined that guidelines for use case diagrams could be separated into two levels: macro and micro levels (see Table II).

TABLE II
Guideline Types of Use Case Diagram

Level	Guideline type	Number of guidelines
Micro level	Actor layout guidelines	5
	Relationship layout guidelines	4
	Note layout guidelines	2
Macro level	Readability guidelines	8
	Simplicity guidelines	2

Micro level includes 11 guidelines, while macro level contains 10 guidelines that together make up 21 guidelines.

IV. Determination of Use Case Diagram Layout Quality

The quality of use case diagrams is very difficult to determine, but there are some aspects that are simple, such as a number of intersecting lines and line curves are obvious drawbacks and should be avoided.

When it comes to the quality of diagram, its elements and size are determined in the beginning.

According to [12], a diagram element is any line, shape, or textual label that appears in a diagram and:

- can be positioned within the diagram by itself; or
- can be shown or hidden by itself; or
- contains other diagram elements.

In turn, the size of a diagram is the number of its diagram elements. While counting use case diagram elements, the following aspects are also considered:

- names can be neither hidden nor moved so they do not count as separate elements. Stereotypes, on the other hand, can be hidden so they count as labels,
- adornments with fixed position relative to the adorned element are not counted, e.g., arrow heads. Adornments that can be moved include multiplicities, association names.

According to [12], a diagram flaw is an instance of:

1) bends of lines that are considered flaws;
2) intersections that are considered flaws if they are visible and not a syntactic element of the language;
3) touching elements that are considered a flaw, unless they have close syntactic or semantic association;
4) sets of merged lines or aligned lines that are close together and are considered a flaw, unless they have the same type and share exactly one of their endpoints;
5) two flaws that are fused into one flaw, if they are very close together and caused by the same intersecting elements.

Finally, the topological quality of a diagram is defined as the number of laws it contains.

It should be considered that several deficiencies are considered as one, if they are close to each other and caused by the same elements.

When it comes to the quality of diagram, counting only the drawbacks of diagrams would be wrong, because with an increase in the number of diagram elements, the number of drawbacks is also increasing, which makes it obvious that a number of drawbacks in diagrams with a small number of elements will also not be large. Therefore, the following formula is used to determine the level of deficiency:

$$G = \frac{m}{n}, \tag{1}$$

where n – a number of use case diagram elements, m – a number of use case diagram flaws.

Further in the paper, automatic algorithms are reviewed and the quality of element layout is calculated by using the previously mentioned formula (1). The acquired results are marked as follows: a high value means that the quality of use case diagram is poor, and vice versa, a small value means the diagram quality is good.

The authors have reviewed the five most popular UML modelling tools such as MetaEdit+, Astah, Enterprise Architect, Visual Paradigm and MagicDraw and their automatic layout algorithms. They are selected as the only ones from the most popular tools, which offer automatic layout abilities for UML diagrams. The very brief review shows that usually tools suggest applying regular algorithms used for graph layout (tree, circular, organic, compact, hierarchical, orthogonal etc.), which can be applicable to use case diagram; therefore, the authors have selected them for a deeper analysis.

While performing automatic algorithm analysis one and the same use case diagram has been created in all reviewed tools. Diagram has the following elements: 8 actors, 8 use cases, 13 relationships, one system border, 11 text titles which in total makes 41 diagram elements.

Initial diagram flaw rate was different in all reviewed tools due to a specific diagram element display. Some tools, such as

MetaEdit + and Astah, offer only one automatic layout algorithm that in both cases reduces the quality of the diagram. Other tools include several layout algorithms, such as hierarchical, orthogonal, automatic, etc., but these algorithms do not fully support the guidelines defined in the work. The highest flaw rate was determined by the use case diagram after the automatic layout algorithm usage in Astah. This algorithm doubles the flaw rate value.

According to [12], a flaw rate of a diagram should not exceed 0.5 and number of flaws should not exceed 15–20, otherwise a diagram flaw rate makes it less readable and some important information might get lost. By analysing the layout algorithms, it may be seen that only few of the algorithms can reduce the flaw rate to be less than 0.5.

Based on the results of layout algorithm analysis, given in Table III, the authors suggest development of the UML use case diagram layout algorithm for their further research.

TABLE III
USE CASE DIAGRAM FLAW RATE

Case tool	Layout	Flaw rate
MetaEdit+	Without layout	0.6829
	Automatic	0.7561
Astah	Without layout	0.6585
	Automatic	1.3415
Enterprise Architect	Without layout	0.6585
	Circular/Ellipse	0.8537
	Neaten	0.6098
	Spring	1.0240
	Digraph	1.0730
	Converge	0.9024
	Diverge	0.6585
	Auto route	0.2439
Visual Paradigm for UML	Without layout	0.4634
	Orthogonal edge route	0.2683
	Automatic	0.5366
	Hierarchic	0.6341
	Orthogonal	0.7561
	Compact circular	0.8048
	Organic	0.6829
	Compact	0.4634
MagicDraw	Without layout	0.7805
	Quick	0.9756
	Hierarchic	0.9756
	Orthogonal	0.3415
	Organic	0.9756
	Circular	1.0240
	Grid	1.2439

V. RELATED WORK

Already in 1985, several works were devoted to the entity relationship diagrams. Batini, Furlani and Nardelly in their research [13] reviewed several aesthetic and applied topology-form-metric approaches.

The authors of the study [14] defined two algorithms that corresponded to certain functions of visual organisation. The first algorithm gradually enlarged the image by selecting and applying placement guidelines to the nodes for as long as no raw node remained and the second was a parallel genetic algorithm.

New approaches and techniques for graph placement were offered in [15] and [16].

Some studies were also carried out for class diagram placement. Battista and his colleagues [17] studied the algorithms of graph formation and their aesthetics. Class diagram work [18] offered aesthetic criteria reflecting complex features of a UML class diagram, a placement algorithm supporting all these features, as well as a graph-building framework capable of producing images according to these criteria. Authors of [19] proposed an algorithm based on a topology-form-metric approach for class diagram automatic layout. In contrast, in [20], authors formulated the main criteria for effective class diagram placement from the perception theory point of view.

Other UML diagrams were also studied. For example, [21] offered an approach that facilitated communication among project participants by creating sequence diagrams in technical documentation.

The authors [22] proposed several criteria for sequence diagrams based on the traditional aesthetics of graphs. The latest research [23], which was dedicated to sequence diagrams, offered a deployment algorithm that could calculate the life line sequence according to different optimization criteria. This work also looks at the problem of sequence diagram size by introducing vertical compression and managing its text labels to compress them horizontally.

Activity diagrams were not left behind as well. In [24], an automated visualization tool for UML activity diagrams was proposed.

There are also several works devoted to an understanding of diagrams. Harald Störrle demonstrated how the diagram size affected the diagram flaw rate in [12], the more elements diagram contained, the greater number of flaws there were. In another work [25], the author also demonstrated that the quality of the placement of diagram elements affected its understanding.

As can be seen from the foregoing, researchers have recently devoted their attention to the automatic placement of UML sequences, activities and class diagram elements, but some diagram layout mechanisms are not considered at all, such as use case diagrams. The study [26] offered an automatic use case diagram layout algorithm in 2008. In this work, Sugiyama algorithm [27] was used as a basis for use case diagram layout algorithm development. Unfortunately, this algorithm does not follow all previously mentioned guidelines as well as the result of the layout does not support the latest version of XMI.

VI. CONCLUSION AND FURTHER RESEARCH

Within the framework of the present research, the authors have discussed the necessity of use case diagrams for the high-level understanding of system functionality and the automatic layout of UML use case diagrams. Based on the existing studies, the authors have collected a set of guidelines which have been divided into two levels: micro and macro levels. In total, the authors defined more than 20 guidelines, which should be followed while creating use case diagrams to make them easy to understand.

To ensure that an automatic algorithm that meets all defined guidelines does not exist, the authors have analysed automatic layout algorithms. As a result, none of the available algorithms have met all previously defined guidelines.

The main conclusions of the research are the following:

- the layout of the diagram is a complicated task due to a large number of guidelines that should be taken into consideration when placing elements in the diagram;
- modeller cannot use convenient algorithms for graph presentation to layout the UML use case diagram due to its specific structure; therefore, some unique methods should be applied;
- layout quality affects the understanding of UML use case diagrams;
- the quality of the layout algorithm strongly depends on the size of a use case diagram.

As a further research step, the automatic use case diagram layout algorithm, which will support all the above-mentioned guidelines, will be created.

References

[1] M. Guo, C. Zhang, and F. Wang, "What is the Further Evidence about UML? - A Systematic Literature Review" *in 2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, 2017, pp. 106–113. https://doi.org/10.1109/APSECW.2017.28

[2] M. Seidl, M. Scholz, C. Huemer, and G. Kappel, *Introduction. In: UML @ Classroom. Undergraduate Topics in Computer Science*. Springer, Cham, 2015, pp. 206. https://doi.org/10.1007/978-3-319-12742-2_1

[3] A. Galapovs, and O. Nikiforova, "UML Diagram Layouting: the State of the Art", *Computer Science. Applied Computer Systems*, vol. 44, no. 1, 2012, pp. 101–108. https://doi.org/10.2478/v10143-011-0027-0

[4] S. Tilley, and S. Huang, "A qualitative assessment of the efficacy of UML diagrams as a form of graphical documentation in aiding program understanding", *SIGDOC: Proceedings of the 21st Annual International Conference on Documentation*, ACM Press, 2003, pp. 184–191. https://doi.org/10.1145/944905.944908

[5] A. Galapovs, and O. Nikiforova, "Several Issues on the Definition of Algorithm for the Layout of the UML Class Diagram", *Proceedings of MDA&MDSD 2011, 3rd International Workshop on Model Driven Architecture and Modeling Driven Software Development In conjunction with the 6th International Conference on Evaluation of Novel Approaches to Software Engineering*. Lisbon: SciTePress, pp. 68–78, 2011.

[6] O. Nikiforova, D. Ahilcenoka, D. Ungurs, K. Gusarovs, and L. Kozacenko, "Several Issues on the Layout of the UML Sequence and Class Diagram", *Proceedings of the 9th International Conference on Software Engineering Advances, ICSEA*, October 12–16, 2014, pp. 40–47. Available from http://www.thinkmind.org/

[7] O. Nikiforova, S. Putintsev, and D. Ahilcenoka "Analysis of Sequence Diagram Layout in Advanced UML Modelling Tools", *Applied Computer Systems*, vol. 19, no. 1, 2016, pp. 37–43. https://doi.org/10.1515/acss-2016-0005

[8] O. Nikiforova, and K. Gusarovs, "Comparison of BrainTool to Other UML Modeling and Model Transformation Tools", *AIP Conference Proceedings, International Conference on Numerical Analysis and Applied Mathematics ICNAAM 2016*, vol. 1863, no. 1, 2017. https://doi.org/10.1063/1.4992503

[9] A. Cockburn, *Writing Effective Use Cases*. Boston: Addison-Wesley, 2001, pp. 304.

[10] I. Jacobson, I. Spence, and B. Kerr, "Use-case 2.0", Communications of the ACM, vol. 59, no. 5, 2016, pp. 61–69. https://doi.org/10.1145/2890778

[11] S. W. Ambler, *The Elements of UML 2.0 Style*. New York: Cambridge University Press, 2005, pp. 201. https://doi.org/10.1017/CBO9780511817533

[12] H. Störrle, "Diagram Size vs. Layout Flaws: Understanding Quality Factors of UML Diagrams", *in ESEM '16 Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Article No. 31, 2016, pp. 10. https://doi.org/10.1145/2961111.2962609

[13] C. Batini, L. Furlani, and E. Nardelly, "What is a Good Diagram? A Pragmatic Approach. In Entity-Relationship Approach: The Use of ER Concept in Knowledge Representation", *Proceedings of the Fourth International Conference on Entity-Relationship Approach*, USA, IEEE Computer Society and North-Holland, 1985, pp. 312–319.

[14] C. Kosak, J. Marks, and S. Shieber, "Automating the Layout of Network Diagrams with Specified Visual Organization", *IEEE Trans. Systems, Man and Cybernetics*, vol. 24, no. 3, 1994, pp. 440–454. https://doi.org/10.1109/21.278993

[15] K. Freivalds, and P. Kikusts, "Optimum Layout Adjustment Supporting Ordering Constraints in Graph-Like Diagram Drawing". *Proc. of the Latvian Academy of Sciences*, 2001, pp. 43–51.

[16] K. Freivalds, U. Dogrusoz, and P. Kikusts, "Disconnected Graph Layout and the Polyomino Packing Approach," *Proc. of Graph Drawing 2001, Lecture Notes in Computer Science*, vol. 2265, 2002, pp. 378–391. https://doi.org/10.1007/3-540-45848-4_30

[17] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Pearson, 1999.

[18] H. Eichelberger, "Aesthetics of class diagrams", *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT, 2002, pp. 23–31.

[19] M. Eiglsperger, M. Kaufmann, and M. Siebenhaller, "A topology-shape-metrics approach for the automatic layout of UML class diagrams", *Soft Vis'03: Proceedings of the 2003 ACM Symposium on Software Visualization*, 2003, pp. 189–198. https://doi.org/10.1145/774833.774860

[20] D. Sun, and K. Wong, "On evaluating the layout of UML class diagrams for program comprehension" *13th International Workshop on Program Comprehension* (IWPC'05), 2005, pp. 1–10.

[21] G. Bist, N. MacKinnon, and S. Murphy, "Sequence diagram presentation in technical documentation", *SIGDOC'04: Proceedings of the 22nd Annual International Conference on Design of Communication*, New York, NY, USA, 2004, pp. 128–133. https://doi.org/10.1145/1026533.1026566

[22] T. Poranen, E. Makinen, and J. Nummenmaa, "How to draw a sequence diagram", *Proceedings of the Eighth Symposium on Programming Languages and Software Tools*, SPLST'03, University of Kuopio, Department of Computer Science, 2003, pp. 91–102.

[23] C. D. Schulze, G. Hoops, and R. von Hanxleden, "Automatic Layout and Label Management for Compact UML Sequence Diagrams", *2018 IEEE Symposium on Visual Languages and Human-Centric Computing* (VL/HCC), Lisbon, 2018, pp. 187–191. https://doi.org/10.1109/VLHCC.2018.8506571

[24] A. Malesevic, D. Brdjanin, and S. Maric, "Tool for automatic layout of business process model represented by UML activity diagram", *Eurocon*, 2013, pp. 537–542.

[25] H. Störrle, "On the Impact of Layout Quality to Understanding UML Diagrams: Size Matters", *Proceedings of 17th International Conference on Model Driven Engineering Languages and Systems*, MODELS 2014, pp. 518–534. https://doi.org/10.1007/978-3-319-11653-2_32

[26] H. Eichelberger, "Automatic layout of UML use case diagrams", *SoftVis '08, Proceedings of the 4th ACM symposium on Software visualization*, 2008, pp. 105–114. https://doi.org/10.1145/1409720.1409738

[27] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for Visual Understanding of Hierarchical System Structures", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 2, 1981, pp. 109–125. https://doi.org/10.1109/TSMC.1981.4308636

Olga Filipova received the Bachelor degree in Computer Systems from the Transport and Telecommunication Institute, Riga (Latvia) in 2011. At present, she is a second-year Master student at the Department of Applied Computer Science, Riga Technical University.
E-mail: olga.filipova_2@edu.rtu.lv

Oksana Nikiforova received the Doctoral degree in Information Technologies (system analysis, modelling and design) from Riga Technical University, Latvia, in 2001. She is a Professor at the Department of Applied Computer Science, Riga Technical University, where she has been working since 1997. Her current research interests include object-oriented system analysis, design and modelling, especially the issues in model driven software development.
E-mail: oksana.nikiforova@rtu.lv
ORCID iD: https://orcid.org/0000 0001-7983-3088