UDC 004.415.2.045 (076.5) https://doi.org/10.15407/pp2020.02-03.164

DOMAIN ENGINEERING APPROACH OF SOFTWARE REQUIREMENT ANALYSIS

O.V. Chebanyuk, O.V. Palahin, K.K. Markov

Requirement analysis is one of the important processes in software development lifecycle management. In Agile approach requirements software models are the basic of generating other software development artifacts. Improving requirements approaches and techniques allows avoiding mistakes in other software development artifacts. Domain engineering fundamentals is the basic for "template oriented" approaches of software development artifacts designing. Reusing domain models and knowledge allows adding details in vertical "model to model" transformation operations, refine generated software development artifacts, organize systematic software reuse and perform many other activities. Paper proposes an approach of requirement analysis based on UML Use Case diagrams transformations into communication ones and the next refinements of them by means of information from domain models. The advantages of the proposed approach is the next: proposed transformation method involves "many to many" transformation in order to save the semantic of initial model. Domain knowledge are used to complete communication diagram by means of adding details after transformation to them. In order to perform Use case to communication transformation graph representation of software models is chosen.

Key words: Domain Engineering, Domain Analysis, Requirement Analysis, Software Model Transformation, UML diagram.

Аналіз вимог є важливим процесом життєвого циклу розробки програмного забезпечення. У гнучких методологіях розробки програмного забезпечення моделі вимог є такими артефактами розробки програмного забезпечення, що містять вихідну інформацію для здійснення подальших завдань розробки. Удосконалення методик аналізу вимог дозволяє уникнути ситуації, коли помилки артефактів, що проектуються при аналізі вимог, переносяться на інші артефакти розробки програмного забезпечення. Доменна інженерія забезпечує фундаментальні основи для впровадження «шаблонно-орієнтованих» методик проектування артефактів розробки програмного забезпечення. Повторне використання доменних моделей та знань дозволяє доповнити інформацію про структуру моделі, що має більш детальну нотацію після виконання вертикальної трансформації «з моделі у модель», уточнити спроектований артефакт розробки програмного забезпечення, організувати систематичне повторне використання програмних модулів та виконати багато інших завдань. У роботі представлено методику аналізу вимог до програмного забезпечення, що базується на трансформації діаграм прецедентів у діаграми комунікацій з їх подальшим уточненням за допомогою інформації, що міститься у доменних моделях. Перевагою представленої методики по зрівнянню з існуючими є те, що для трансформації використовуються всі складові вихідної моделі з метою перенести її семантику на результуючу модель. Після трансформації виконується уточнення діаграм комунікацій із використанням накопичених знань про домен. Вихідною інформацією для трансформації моделей програмного забезпечення їх аналітичне представлення у графовій формі.

Ключові слова: доменна інженерія, доменний аналіз, аналіз вимог, трансформація моделей програмного забезпечення, UML діаграма.

Анализ требований является важным процессом жизненного цикла разработки программного обеспечения. В гибких методологиях разработки программного обеспечения модели требований являются артефактами разработки программного обеспечения, которые содержат исходную информацию для осуществления дальнейших задач разработки. Совершенствование методик анализа требований позволяет избежать ситуации, когда ошибки артефактов, которые проектируются при анализе требований, переносятся на другие артефакты разработки программного обеспечения. Доменная инженерия обеспечивает фундаментальные основы для внедрения «шаблонно-ориентированных» методик проектирования артефактов разработки программного обеспечения. Повторное использование доменных моделей и знаний позволяет дополнить информацию о структуре модели, имеющей более подробную нотацию после выполнения вертикальной трансформации «из модели в модель», уточнить спроектированный артефакт разработки программного обеспечения, организовать систематическое повторное использование программных модулей и выполнить много других задач. В работе представлена методика анализа требований к программному обеспечению, основанная на трансформации диаграмм прецедентов в диаграммы коммуникаций с их последующим уточнением с помощью информации, содержащейся в доменных моделях. Преимуществом представленной методики по сравнению с существующими является то, что для трансформации используются все составляющие исходной модели с целью перенести ее семантику на результирующую модель. После трансформации выполняется уточнение диаграмм коммуникаций с использованием накопленных знаний про домен. Исходной информацией для трансформации моделей программного обеспечения является их аналитическое представление в графовой форме.

Ключевые слова: доменная инженерия, доменный анализ, анализ требований, трансформация моделей программного обеспечения, UML диаграмма.

Introduction

In practice, domain engineering finds practical implementation in Software Product Line approach. There are software engineering standards with recommendations to organize lifecycle processes in AGILE approach (ISO 12207, ISO 15288, ISO 19770-1, ISO 29119-2, ISO 20000-4). General recommendations of software development lifecycle process organization are complicated by specific operations aimed to organize an effective reuse of different software development artifacts. As software models are central development artifacts in AGILE approach operations of their reuse will allow to avoid designing and other mistakes. In order to organize effective software artifacts reuse scheme it is necessary to answer on two research questions (RQ):

(RQ1) What should be reused? Other words: how to select proper domain knowledge for reuse?

(RQ2) How to merge domain model with software development artifacts?

Effective solving of these questions propose performing the next activities:

- forming request of searching in domain area through domain artifacts in repository;
- organizing search procedure and defining matching criterion;
- merging domain knowledge with software development artifacts.

Related papers and practical research

Involving Domain engineering into software artifacts reuse started in the end of the previous century. Reuse researches performed in two directions. Research laboratories of big companies accumulated practical achievements in this area. Scientific research directed to development of an analytical approaches.

As a result of research laboratories practices analysis shows that the next factors slowed the process of software artifacts reuse:

Successful search of software development artifacts in Motorola practices was limited because it was some difference rules using meta-information while preparing information about software artifacts and its further reuse during search.

IBM focused on architectural solutions reuse. As a procedure of architectural solutions adoption for future projects is quite complicated, architectural solutions may contain errors or rigid design characteristics.

Hewlett-Packard developer teams make some free procedure of adoption software development life cycle process including extra processes for preparing high – quality software development artifacts ready for the further reuse.

Table 1 Summarizing results of research laboratories IBM, Motorola and Hewlett-Packard companies.

Table 1. Level of coverage requirements of software artifacts reuse in application engineering

Application engineering requirements for effective reuse of software development artifacts	Motorola	IBM	Hewlett Packard
Formal apparatus of software artifacts reuse	+	+	–
Formal apparatus of software artifacts semantic similarity	–	+	–
Providing maturity level of software development lifecycle processes	–	–	+

Analysis of scientific papers devoted to domain engineering development pointed that there is a list of factors that slow the development of software artifacts reuse in domain engineering:

1. Absence of the common concept and complex approach of software artifacts reuse information that is based on gathering information while domain analysis and its further reuse in application engineering processes [1–4].
2. Existing approaches of software artifacts reuse estimation do not contain formal apparatus of choosing the best software development artifact from the set of possible ones. [5–8].
3. Complex of tasks needed to be solved for software artifact reuse usually performed by means of different software development tools that use different formats of data representation. Inaccurate data transition between formats can be a cause of their partially lost or appearing some not expected elements [9–12].
4. Absence of formal methods allowing synchronizing domain models structure when initial information about domain analysis is changed (text, audio, video, web-site etc.) [13–16].
5. Difficulty to collaborate results of software models processing in text and graphical representation [17–19].
6. Absence of formal approaches of preparation and reuse meta-information about software development artifacts [20–22].

Proposed approach

Proposed approach is grounded on collaboration of knowledge about problem domain that were accumulated in domain analysis procedure [23] and improvement of requirement analysis procedures. The aim of improvement requirement analysis procedure is to spread information about Use Case Diagram and design communication diagram that satisfy the requirements and store the semantics of requirement specification.

From domain analysis artifacts, controlled vocabulary is used. Requirement analysis of artifacts consists of requirement specification and Use Case Diagram.

Proposed approach is based on performing transformation from Use Case to Communication Diagram, transforming whole structure of Use Case. Graph representation of UML diagram is chosen. Initial information for transformation is prepared composing all graph paths from textual representation (XMI) of UML diagram. The concept of “text to model” transformation is proposed in paper [24].

Data flow of the proposed approach is represented in the figure 1.

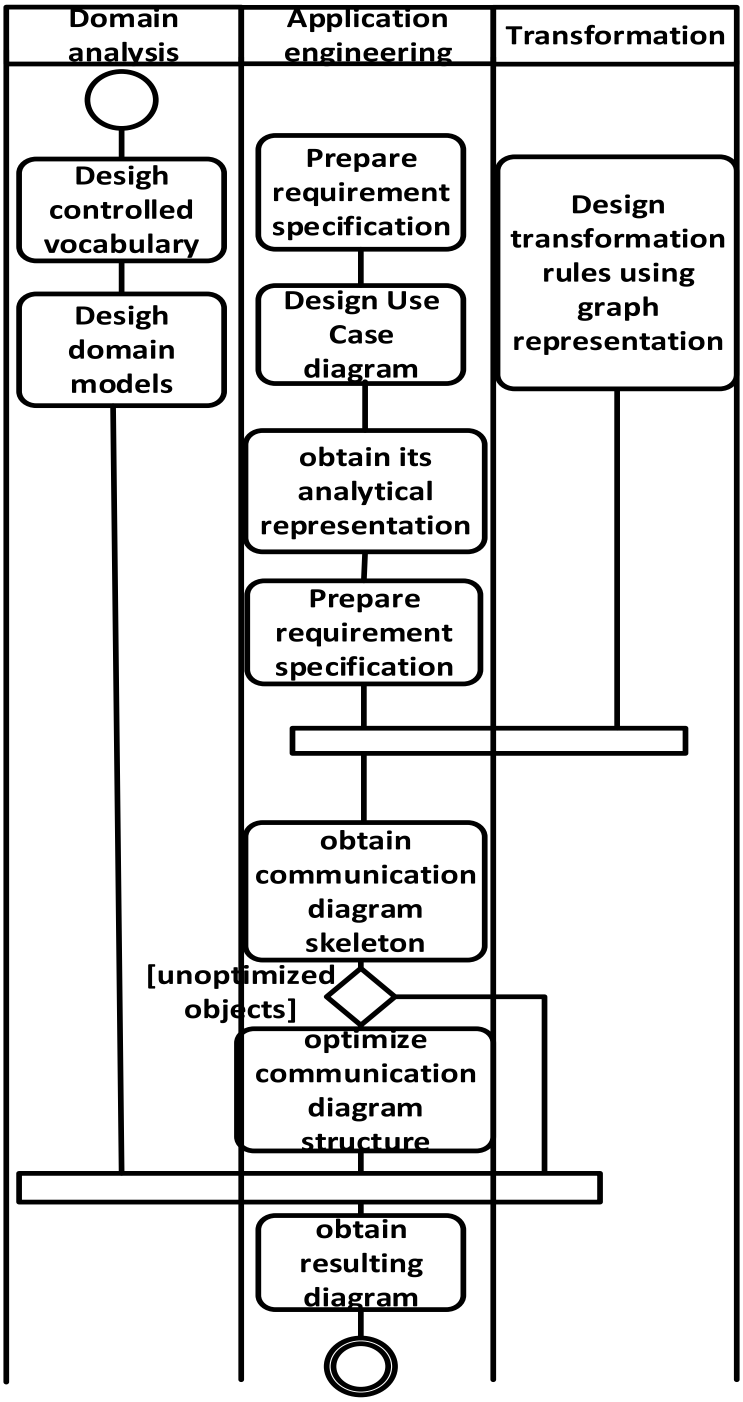

Figure 1. Data flow of the proposed approach

In order to solve this task propose the ***next denotations:***
Graph representation of Use Case Diagram, that consider data streams

$$\begin{aligned}
&SM_{use_case} = \{path_1, path_2, ..., path_n\}, n = |SM_{use_case}| \\
&path = (esg_1, esg_2, ..., esg_p) \\
&esg = (ob_1, link, ob_2) \\
&ob_1, ob_2 \in \{p, a, c\} \\
&link \in \{l, l(include), l(extends), l(inh)\}
\end{aligned},$$

where SM_{use_case} – denotation of whole Use Case Diagram,

$path_1$ – path in the Use Case Diagram representing one data stream (path in graph),

esg – elementary sub-graph, describing two directly linked objects ob_1 *and* ob_2 by means of link.

Objects (ob) in notation of Use Case Diagram can be the next type *a* – actors; *p* – precedents; *c* – comments.

Links in Use Case diagram can be the next types – *l(include)* – include, *l(extends)* – extends, *l(inh)* – inheritance.

$$
\begin{aligned}
&SM_{com} = \{path_1, path_2, ..., path_n\}, n = |SM_{com}| \\
&path = (esg_1, esg_2, ..., esg_p) \\
&esg = (ob_1, m, ob_2) \\
&ob_1, ob_2 \in \{a, c, obj\}
\end{aligned},
$$

where SM_{com} – Communication Diagram,

ob_1, ob_2 – Communication Diagram objects,

m – Communication Diagram message.

Denote transformation operation from Use Case Diagram to Communication one as: SM_{ini} y SM_{rez} as:

$$SM_{use\_case} \xrightarrow{TRANS} SM_{com},$$

where $\xrightarrow{TRANS}$ is a set of transformations rules, which are applied when Use Case diagram is transformed into communication one.

A set of domain entities in controlled vocabulary (*ConVoc*)

$$ConVoc = \{c_1, c_2, ..., c_n\}, n = |ConVoc|.$$

A set of Use Case diagram precedents is:

$$
\begin{aligned}
&P_{use\_case} = \{p_1, p_2, ..., p_k\} \\
&p = (w_1, w_2, ..., w_t), p \in P_{use\_case}
\end{aligned}.
$$

Let define the transformation rules using proposed denotations.

In order to perform transformation from Use case to Communication diagrams.

Transformation rules represented in the paper [25] are used. Grounding on these rules, it is proposed rule for transforming whole Use Case diagram into communication one.

Rules for obtaining skeleton of communication diagram

$$
\begin{aligned}
&PATH_{use\_case} \xrightarrow{TRANS} PATH_{com} \\
&path_{use\_case} \xrightarrow{TRANS} path_{com} \\
&esg_{use\_case} \xrightarrow{TRANS} esg_{com} \\
&TRANS = \{trans_1, trans_2\} \\
&trans_1 : (a, l, p) \to (a, m, obj) \\
&trans_2 : (p_1, l, p_2) \to (obj_1, m, obj_2)
\end{aligned},
$$

where $path_{use\_case}$ – path in Use Case Diagram,

$path_{com}$ – path in Communication Diagram,

esg_{use_case} – elementary sub-graph in Use Case Diagram,

esg_{com} – elementary sub-graph in Communication Diagram,

obj – Communication Diagram object.

After performing such a transformation, the next task is to give a name for obtained objects. Denote named objects as *obj(name)*. The rule of naming object is written in the following way:

$$
\begin{aligned}
&ConVoc \cap p = \{w = c \mid w \in p, c \in Convoc\} \\
&ConVoc \cap p \neq \varnothing \to obj(name) = ConVoc \cap p
\end{aligned}. \quad (1)
$$

The last transformation task is to optimize Communication Diagram structure by means of applying "self-message" rule. Self-message is the message that is outcomes and incomes to the same communication diagram object.

$$\begin{array}{l} if \;\; obj_1 = obj_2 \; in(obj_1, m, obj_2) \\ \quad then \;\; (obj_1, m, obj_2) \rightarrow (obj, m(self), obj) \end{array}.$$

Graphically such communication diagram fragment (figure 2,a) is changed to the next (figure 2,b).

a b

Figure 2. "Self-message" optimization rule: a – obtained communication diagram fragment;
b – optimized communication diagram fragment

Describe the steps of the proposed approach of communication diagram designing that based on Use Case diagram (application engineering artifact) and controlled vocabulary (domain analysis artifact).

1. Compose of a problem domain controlled vocabulary.
2. Design Use case diagrams from requirement specification.
3. Obtain a skeleton of communication diagram from the Use Case using proposed transformation rules

$$SM_{use\_case} \xrightarrow{TRANS} SM_{com}.$$

4. Fill communication diagram skeleton by means of objects names using.
5. Entities from controlled vocabulary in Use Case diagram using (1).
6. Optimize structure of communication diagram using self-message rule.

Case study

Consider example of Use case diagram for visualizing data of accounting reports. Report settings are stored in profiles. Reports visualized in using graphics. Graphics are obtained considering time settings. Use Case Diagram is represented in the figure 3.

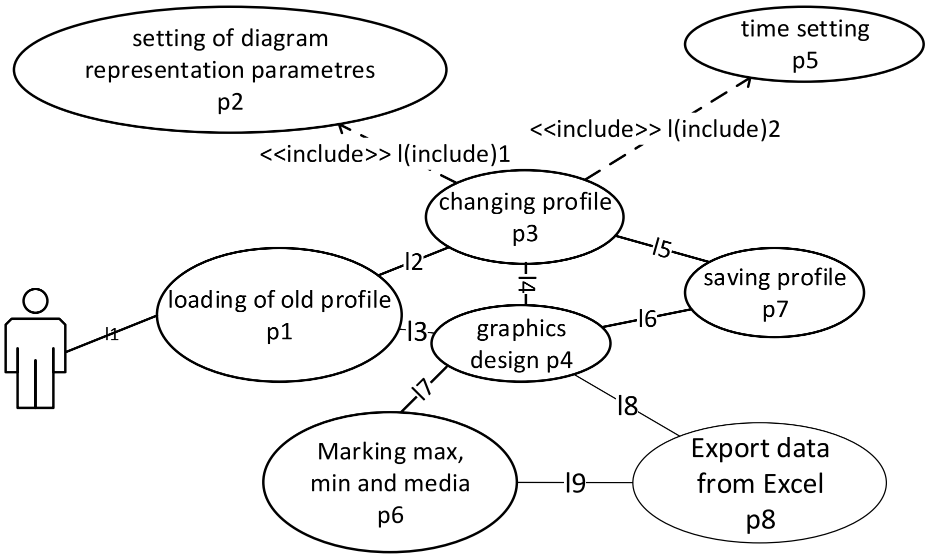

Figure 3. Use case diagram of visualizing accounting reports

Analytical representation of this diagram is prepared using approach represented in [24]. A set of Path is containing from six elements. Some part of paths are duplicated. Analytical representation of Use case diagram contains the initial information for designing of communication diagram structure.

$$chain_1 = (a_1, l_1, p_1)$$
$$chain_2 = chain_1(p_1, l_3, p_4) \quad chain_3 = chain_1(p_2, l_2, p_3)$$
$$path_1 = chain_3(p_3, l(include)_1, p_2)$$
$$path_1 = chain_1(p_3, l(include)_2, p_5)$$
$$chain_4 = chain_3(p_3, l_4, p_4) \quad chain_5 = chain_3(p_3, l_5, p_7)$$
$$chain_6 = chain_1(p_1, l_3, p_4) \quad chain_7 = chain_6(p_4, l_7, p_6)$$
$$chain_8 = chain_3(p_7, l_6, p_4) \quad chain_9 = chain_8(p_7, l_4, p_4)$$
$$path_3 = chain_3(p_7, l_6, p_8)$$
$$path_4 = chain_8(p\ , l\ , p_8)$$
$$path_5 = chain_9(p_4, l_8, p_8)$$
$$path_6 = chain_6(p_6, l_8, p_8)$$

Expression (2) represents example of transformation Use case diagram path into communication one.

$$\begin{aligned} &path_{1(use\_case)} = (a_1, l_1, p_1), (p_1, l_2, p_3), (p_3, l(include)_1, p_2) \\ &path_{1(com)} = (a_1, m_1, obj_1), (obj_1, m_2, obj_3), (obj_3, m_3, obj_4)\ . \\ &obj_1 = profile, obj_2 = profile, obj_3 = "to\,define" \end{aligned} \quad (2)$$

The note according to transformation rule names of different objects can be the same. It is pointed to the fact that the diagram needs the further optimization. Name of object “to define” points, that in order to define the name of communication diagram object the information from domain knowledge is used. After designing all paths of communication diagram, its skeleton is composed (figure 4).

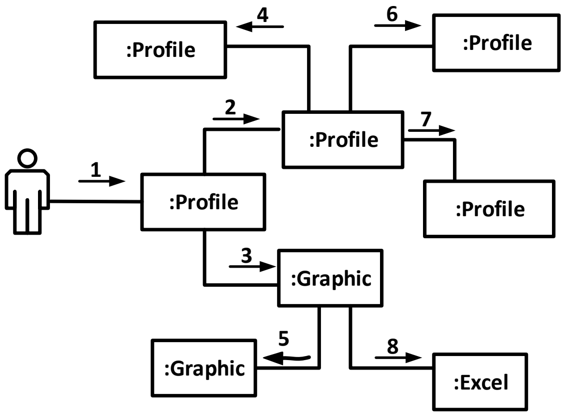

Figure 4. Unoptimized “skeleton” of the Communication Diagram

After performing sequence of Communication Diagram refinement (implementing self-object messaging rule) obtain diagram that is represented in figure 5 and 6.

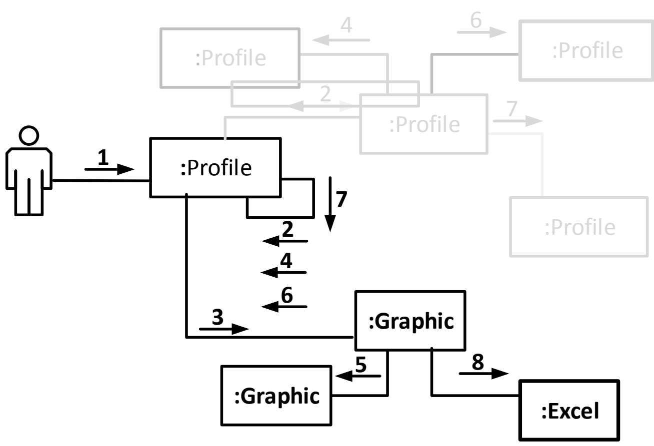

Figure 5. First- step of communication diagram skeleton optimization (Object profile is optimized)

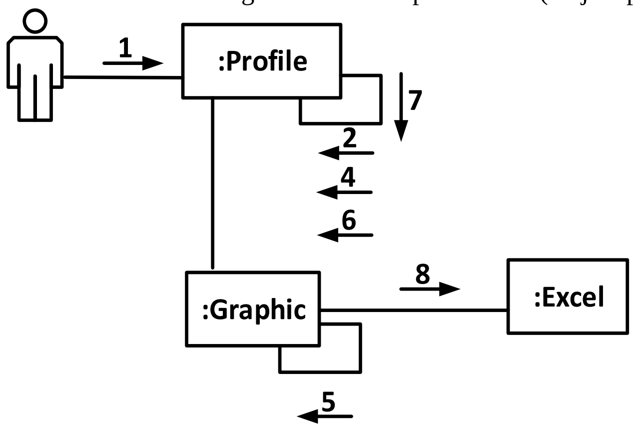

Figure 6. Refined Communication Diagram

The next step is to complete diagram structure by problem domain entities and their properties (figure 7). Performing this step it is defined, which data streams can be organized in parallel.

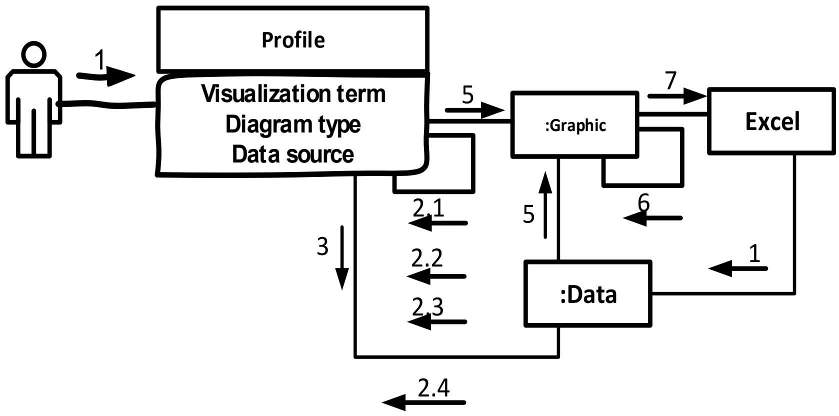

Figure 7. Communication Diagram that is complicated from domain knowledge

Conclusion

Known "model to model" transformation approaches do not use the whole structure of initial diagram. It may be cause of losing some information or performing additional efforts of domain analytics to organize the structure of resulting diagram. From the other hand, such approaches require additional time and efforts.

Proposed approach aimed to designing of Communication Diagram from Use Case one. It is grounded on usage of whole Use Case diagram structure while transformation operation is performed. Such a fact allows saving Use Case semantics after transformation. As proposed approach implements vertical transformation, resulting diagram complicated by information about problem domain from domain knowledge.

Further research

It is planned to design formal approach allowing reuse domain knowledge while designing different types of UML diagrams in Software Product Line.

References

1. Hooper J.W., & Chester R.O. (1991). *Software reuse: guidelines and methods*. Springer Science & Business Media.
2. Marshall J.J., & Downs R.R. (2008, July). *Reuse readiness levels as a measure of software reusability*. In IGARSS 2008-2008 IEEE International Geoscience and Remote Sensing Symposium (Vol. 3, pp. III-1414). IEEE.
3. Smith M., & Sodhi J. (1994). *Marching Towards a Software Reuse Future*. ACM SIGAda Ada Letters, 14(6), 62–72.
4. Vieira M., Madeira H., Cruz S., Costa M., & Cunha J.C. (2011, June). *Integrating GQM and Data Warehousing for the Definition of Software Reuse Metrics*. In 2011 IEEE 34th Software Engineering Workshop (P. 112–116). IEEE.
5. Maga C., & Jazdi N. (2009, June). Concept of a domain repository for industrial automation. In Proceedings of the First International Workshop on Domain Engineering.

6. Komissarchik J., & Komissarchik E. (2008). U.S. Patent N 7,454,430. Washington, DC: U.S. Patent and Trademark Office.
7. Van der Meij L., Isaac A., & Zinn C. (2010, May). *A web-based repository service for vocabularies and alignments in the cultural heritage domain*. In Extended Semantic Web Conference (P. 394–409). Springer, Berlin, Heidelberg.
8. Dwyer M.B., Hatcliff J., Robby R., Pasareanu C.S., & Visser W. (2007, May). *Formal software analysis emerging trends in software model checking*. In 2007 Future of Software Engineering (P. 120–136). IEEE Computer Society.
9. Whalen M., Cofer D., Miller S., Krogh B.H., & Storm W. (2007, July). *Integration of formal analysis into a model-based software development process*. In International Workshop on Formal Methods for Industrial Critical Systems (P. 68–84). Springer, Berlin, Heidelberg.
10. Qin W., Rajagopalan S., & Malik S. (2004, June). *A formal concurrency model based architecture description language for synthesis of software development tools*. In ACM SIGPLAN Notices (Vol. 39, N 7, P. 47–56). ACM.
11. Fraser M.D., & Vaishnavi V.K. (1997). *A formal specifications maturity model.* Communications of the ACM, 40(12), 95–104.
12. Satyananda T.K., Lee D., Kang S., & Hashmi S.I. (2007, August). *Identifying traceability between feature model and software architecture in software product line using formal concept analysis.* In 2007 International Conference on Computational Science and its Applications (ICCSA 2007) (P. 380–388). IEEE.
13. Markopoulos P. (2013). *A compositional model for the formal specification of user interface software* (Doctoral dissertation).
14. Bjorner D. (2019). *Domain analysis and description principles, techniques, and modelling languages*. ACM Transactions on Software Engineering and Methodology (TOSEM), 28(2), 1-67.
15. Cao L., Liu J., Wang Q., Jiang C., & Zhang L. (2019). *An efficient structural uncertainty propagation method based on evidence domain analysis*. Engineering Structures, 194, 26–35.
16. Rabiser R., Schmid K., Eichelberger H., Vierhauser M., Guinea S., & Grünbacher P. (2019). *A domain analysis of resource and requirements monitoring: Towards a comprehensive model of the software monitoring domain*. Information and Software Technology, 111, 86–109.
17. D'silva V., Kroening D., & Weissenbacher G. (2008). *A survey of automated techniques for formal software verification*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(7), 1165–1178.
18. Ouimet M., & Lundqvist K. (2007). *Formal software verification: Model checking and theorem proving. Embedded Systems Laboratory Technical Report* ESL-TIK-00214, Cambridge USA.
19. Ammann P., & Black P. E. (1999, October). *Abstracting formal specifications to generate software tests via model checking*. In Gateway to the New Millennium. 18th Digital Avionics Systems Conference. Proceedings (Cat. No. 99CH37033) (Vol. 2, P. 10-A). IEEE.
20. Bennion M., & Habli I. (2014, May). *A candid industrial evaluation of formal software verification using model checking.* In Companion Proceedings of the 36th International Conference on Software Engineering (P. 175–184). ACM.
21. Jetley R., Iyer S.P., & Jones P. (2006). *A formal methods approach to medical device review*. Computer, 39(4), 61–67.
22. Broy M., Krüger I.H., & Meisinger M. (2007). *A formal model of services*. ACM Transactions on Software Engineering and Methodology (TOSEM), 16(1), 5.
23. Chebanyuk O. & Palahin O. (2019) *Domain Analysis Approach.* International journal "Informational Content and Processing". Volume 6, Number 2, 2019, 3–20.
24. Chebanyuk O. (2018) *An Approach of Text to Model Transformation of Software Models.* In Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018), 432–439 (видання індексується у SCOPUS)
25. Chebanyuk E. (2014) *An approach to class diagram designing.* Proceedings of the 2st International Conference on Model-Driven Engineering and Software Development, 7–9 January 2014 y. Portugal, Lisbon. 579–583.

Література

1. Hooper J.W., & Chester R.O. (1991). *Software reuse: guidelines and methods*. Springer Science & Business Media.
2. Marshall J.J., & Downs R.R. (2008, July). *Reuse readiness levels as a measure of software reusability*. In IGARSS 2008-2008 IEEE International Geoscience and Remote Sensing Symposium (Vol. 3, pp. III-1414). IEEE.
3. Smith M., & Sodhi J. (1994). *Marching Towards a Software Reuse Future*. ACM SIGAda Ada Letters, 14(6), 62–72.
4. Vieira M., Madeira H., Cruz S., Costa M., & Cunha J.C. (2011, June). *Integrating GQM and Data Warehousing for the Definition of Software Reuse Metrics*. In 2011 IEEE 34th Software Engineering Workshop (P. 112–116). IEEE.
5. Maga C., & Jazdi N. (2009, June). Concept of a domain repository for industrial automation. In Proceedings of the First International Workshop on Domain Engineering.
6. Komissarchik J., & Komissarchik E. (2008). U.S. Patent N 7,454,430. Washington, DC: U.S. Patent and Trademark Office.
7. Van der Meij L., Isaac A., & Zinn C. (2010, May). *A web-based repository service for vocabularies and alignments in the cultural heritage domain*. In Extended Semantic Web Conference (P. 394–409). Springer, Berlin, Heidelberg.
8. Dwyer M.B., Hatcliff J., Robby R., Pasareanu C.S., & Visser W. (2007, May). *Formal software analysis emerging trends in software model checking*. In 2007 Future of Software Engineering (P. 120–136). IEEE Computer Society.
9. Whalen M., Cofer D., Miller S., Krogh B.H., & Storm W. (2007, July). *Integration of formal analysis into a model-based software development process*. In International Workshop on Formal Methods for Industrial Critical Systems (P. 68–84). Springer, Berlin, Heidelberg.
10. Qin W., Rajagopalan S., & Malik S. (2004, June). *A formal concurrency model based architecture description language for synthesis of software development tools*. In ACM SIGPLAN Notices (Vol. 39, N 7, P. 47–56). ACM.
11. Fraser M.D., & Vaishnavi V.K. (1997). *A formal specifications maturity model.* Communications of the ACM, 40(12), 95–104.
12. Satyananda T.K., Lee D., Kang S., & Hashmi S.I. (2007, August). *Identifying traceability between feature model and software architecture in software product line using formal concept analysis.* In 2007 International Conference on Computational Science and its Applications (ICCSA 2007) (P. 380–388). IEEE.
13. Markopoulos P. (2013). *A compositional model for the formal specification of user interface software* (Doctoral dissertation).
14. Bjorner D. (2019). *Domain analysis and description principles, techniques, and modelling languages*. ACM Transactions on Software Engineering and Methodology (TOSEM), 28(2), 1-67.
15. Cao L., Liu J., Wang Q., Jiang C., & Zhang L. (2019). *An efficient structural uncertainty propagation method based on evidence domain analysis*. Engineering Structures, 194, 26–35.
16. Rabiser R., Schmid K., Eichelberger H., Vierhauser M., Guinea S., & Grünbacher P. (2019). *A domain analysis of resource and requirements monitoring: Towards a comprehensive model of the software monitoring domain*. Information and Software Technology, 111, 86–109.
17. D'silva V., Kroening D., & Weissenbacher G. (2008). *A survey of automated techniques for formal software verification*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(7), 1165–1178.
18. Ouimet M., & Lundqvist K. (2007). *Formal software verification: Model checking and theorem proving. Embedded Systems Laboratory Technical Report* ESL-TIK-00214, Cambridge USA.
19. Ammann P., & Black P. E. (1999, October). *Abstracting formal specifications to generate software tests via model checking*. In Gateway to the New Millennium. 18th Digital Avionics Systems Conference. Proceedings (Cat. No. 99CH37033) (Vol. 2, P. 10-A). IEEE.

20. Bennion M., & Habli I. (2014, May). *A candid industrial evaluation of formal software verification using model checking.* In Companion Proceedings of the 36th International Conference on Software Engineering (P. 175–184). ACM.
21. Jetley R., Iyer S.P., & Jones P. (2006). *A formal methods approach to medical device review*. Computer, 39(4), 61–67.
22. Broy M., Krüger I.H., & Meisinger M. (2007). *A formal model of services.* ACM Transactions on Software Engineering and Methodology (TOSEM), 16(1), 5.
23. Chebanyuk O. & Palahin O. (2019) *Domain Analysis Approach.* International journal “Informational Content and Processing”. Volume 6, Number 2, 2019, 3–20.
24. Chebanyuk O. (2018) *An Approach of Text to Model Transformation of Software Models.* In Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018), 432–439 (видання індексується у SCOPUS)
25. Chebanyuk E. (2014) *An approach to class diagram designing.* Proceedings of the 2st International Conference on Model-Driven Engineering and Software Development, 7–9 January 2014 y. Portugal, Lisbon. 579–583.

Received 02.03.2020

Information about the authors:

Chebanyuk Olena Viktorivna,
PhD, associate professor of software engineering department,
PhD, associate professor.
Number of publications – approximately 75.
Publications in Ukrainian journals – 35.
Publications in foreign journals – 35.
PP Hirsh index=4, Scopus – 1.
https://orcid.org/0000-0002-9873-6010 (ORCID name Elena Chebanyuk),

Palahin Olexander Vasyliovych,
Doctor of Sciences, Academician of National Academy of Sciences of Ukraine,
Deputy director of Glushkov Institute of Cybernetics, head of department 205.
Publications in Ukrainian journals – 290.
Publications in foreign journals – 45.
H-index: Google Scholar – 15, Scopus – 3.
http://orcid.org/0000-0003-3223-1391,

Markov Krassimir K.,
Professor Dr.
Number of publications: more than 135; 5 monographs.
PP Hirsh index – 11.
https://orcid.org/0000-0001-5041-1498 (ORCID name Krassimir Markov)
WoS ResearcherID L-6845-2018.

Authors' place of work:

National Aviation University,
03058 ave. Lubomira Guzara 1,
Phone: 044-406-76-41,
E-mail: chebanyuk.elena@gmail.com (chebanyuk.elena@ithea.org)

V.M. Glushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine
40, Academician Glushkov Avenue, Kyiv, 03187, Ukraine

Institute of Information Theories and Applications, Sofia, 1000, P.O. Box 775, Bulgaria.
E-mail: markov@ithea.org