УДК 004.053

An approach to assessment of dynamic software variability in mobile applications development

R.O. Gamzayev [1], E. Karaçuha [2], M.V. Tkachuk [1], O.Y. Tovstokorenko [3]
[1]*V. N. Karazin Kharkiv National University, Svobody Square 4, Kharkov, 61022, Ukraine*
[2]*Istanbul Technical University, Maslak, 34467 Sarıyer/İstanbul, Turkey*
[3] *National Technical University "Kharkiv Polytechnic Institute", Kyrpychova str., 2, Kharkiv, 61002 Ukraine*
e-mail: tka.mobile@gmail.com

The article describes the approach to the assessment of code reuse in Dynamic Product Line lines (DSPL). Some existing mechanisms to realize software variability in DSPL, such as machine learning, adaptive configurations based on Java programming tools which allow developing DSPL, especially in mobile applications domain, have been reviewed. During the development, some methods for the implementation of the variability specific to the selected programming language have been tested. For each of these mechanisms, such as Weighted Methods per Class, Response for a Class, Depth of Inheritance Tree, Coupling Between Objects, Number of Children, the code complexity metrics have been calculated. Based on these results the code reusability extent can be estimated for each of given variation mechanisms.

Keywords: *software product line, dynamic variability, complexity metrics, code reusability, mobile application.*

Для забезпечення ефективного проектування будь-якої складної розподіленої програмної системи необхідно забезпечити можливість повторного використання певних ресурсів проекту, особливо вихідних кодів і виконуваних компонентів програмного забезпечення. Це дозволяє скоротити час розробки та зберегти інші важливі проектні ресурси. У багатьох сучасних джерелах вивчаються питання, пов'язані зі здатністю оцінювати можливість повторного використання коду вже на етапі проектування програмного забезпечення. Стаття описує підхід до аналізу ступеня повторного використання коду у динамічних лінійках програмних продуктів (DSPL). Авторами зроблено стислий огляд існуючих підходів до побудови DSPL, зокрема, такі як машинне навчання та використання адаптивних конфігурацій. На основі цього була розроблена доменна модель для аналізу альтернативних механізмів варіабельності у DSPL для мобільних застосунків із використання мови програмування Java. Під час розробки були використанні деякі методи імплементації варіабельності специфічні для обраної мови. Перший механізм варіації виконання визначає список серверів, що відповідають за конфігурації збирання: debug або release. Другий визначає, чи має працювати ручне визначення сервера (коли вибирається тестовий сервер) або використовується попередньо визначений сервер. Третій механізм визначає сервер за умовчанням, який буде використовуватися, якщо нічого не вибрано. Для кожного з цих підходів були розраховані такі метрики якості як "Дерево глибини успадкування", "Зв'язок між об'єктами", "Кількість дочірніх класів", а також була оцінена ступінь повторного використання коду кожним з цих методів.

Ключові слова: *лінійка програмних продуктів, варіабельність, метрики якості, повторне використання коду, мобільна розробка.*

Для обеспечения эффективного проектирования любой сложной распределенной программной системы необходимо обеспечить возможность повторного использования определенных ресурсов проекта, особенно исходных кодов и выполняемых компонентов программного обеспечения. Это позволяет сократить время разработки и сохранить другие важные проектные ресурсы. Во многих современных источниках изучаются вопросы, связанные со способностью оценивать возможность повторного использования кода уже на этапе проектирования программного обеспечения. Статья описывает подход к анализу степени повторного использования кода в динамических линейках программных продуктов (DSPL). Авторами сделан краткий обзор существующих подходов к построению DSPL, в частности, такие как машинное обучение и использование адаптивных изменений. На основе этого была разработана доменная модель для анализа альтернативных механизмов вариабельности в DSPL для мобильных приложений с использование языка программирования Java. При разработке были использованы некоторые методы имплементации вариабельности специфические для выбранного языка. Первый механизм вариации выполнения определяет список серверов, отвечающих за конфигурации сборки: debug или release. Второй определяет, имеет ли работать ручное определение сервера (когда выбирается тестовый сервер) или используется предварительно определенный сервер. Третий механизм определяет сервер по умолчанию, который будет использоваться, если ничего не выбрано. Для каждого из этих подходов были рассчитаны такие метрики качества как "Дерево глубины наследования", "Связь между объектами", "Количество дочерних классов", а также была оценена степень повторного использования кода каждым из этих методов.

Ключевые слова: *линейка программных продуктов, вариабельность, метрики качества, повторное использование кода, мобильная разработка*

1 Introduction

Nowadays with the growth of businesses and expansion of markets, it is necessary to produce software products at a lower cost, in shorter time, but provide higher quality at the same time. Also, the practice of using the so-called production line takes over the old-fashioned individual product creation. The main disadvantage of the traditional production line approach is lack of diversification. This is where the software product lines come into play. They enable developers to create families of systems

faster and cheaper as in the production line approach, but at the same time give an opportunity to customization. The motivations for product line engineering include reduction of a development cost, an improvement of the quality and a reduction of product delivering time. It also allows for coping with evolution and complexity.

One of the most important activities in the early stages of software product line development process is the definition of commonality and variability of the product line. While the commonality simply describes the reusable assets that systems have in common, what we like to investigate is the variability, i.e. the features that are unique for each system belonging to the family. It is worth mentioning that research on handling the variability at runtime has led to the creation of dynamic software product line engineering, which is the modern approach to developing software product lines.

A lot of researches in software product line engineering area are concerned with the variability management. The following main questions have been recently considered:

- which software components can be varied;
- how to handle variability on different levels efficiently;
- how variability can be achieved.

Considering the speed of development and growing complexity of mobile application market [1], the DSPL can be introduced on it as well. Taking this into account, the questions concerning a common set of software requirements, development approaches and technological complexity are still open. So the goal of current research is to analyze the reusability evaluation tasks during the DSPL development and developing the approach to its management.

This paper is structured in the following way: in the first section, some technological approaches to design variable software are presented; in the second section Dynamic Software Product Lines (DSPL) is overviewed, in the third section, the usage of some Android-related technologies for variability management is discussed. Then our case study is presented and in the conclusion our future work is discussed.

2 Related work

The problem of the system adaptation in response to context changes is covered in [2]. This is why traditional feature models that describe DSPL are augmented with a set of rules that explicitly determine under which condition a reconfiguration should occur. Developing an effective set of adaptation policies is a challenge for developers due to the complexity of the context and its dynamic nature. While usually in DSPL context changes can be anticipated at design time, in some systems it is not possible due to the uncertainty of runtime context changes.

There are several approaches that may help with this kind of uncertainty [2]. The first one, machine learning, aims at learning unanticipated context changes and creating new adaptation rules dynamically. The second approach, evolution, focuses on changing variability of the DSPL, while derived configurations are running. Both learning and evolution address specific problems of uncertainty, which is why they are not used in isolation [3].

The overall adaptive system model to compare with [3] is shown in Fig. 1. Learning and evolution both are the parts of the process.

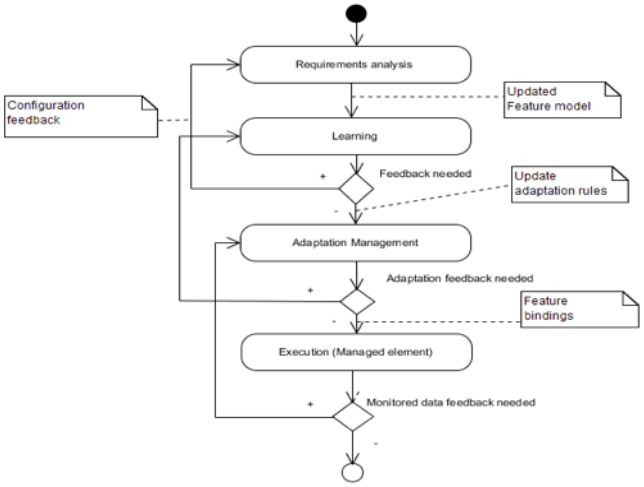

Fig. 1 Workflow Learning and Evolution in DSPLs (to compare with [3])

In the mobile development, this topic can be presented by a number of tasks that could be solved using variability mechanisms, such as multilingual support, the presence of several feature sets for different types of users, as well as the access levels of these users (paid / free accounts). Moreover, taking into account the growing popularity of cross-platform languages for mobile development, it is necessary to pay attention to the specifics of adaptation possibilities for each of the platforms and, in accordance with their requirements, to limit some functionality on one or another end platform.

3 The problem statement: Assessment of code reusability extent in DSPL development

To ensure an efficient design of any complex distributed software system it is necessary to ensure the possibility of reuse of certain project assets, especially a source code and executable software components. This can help to reduce the development time and save other equally important resources.

In many modern sources, issues related to the ability to assess the code reusability already at the design stage of the software development are studied. In particular, the relationship between the complexity of the code and the reusability is indicated in [5].

The complexity of the software code is determined on the basis of the values of the following structural complexity metrics [6]:

- Depth of Inheritance Tree (DIT);
- Response for a Class (RFC);
- Number of Children (NOC);
- Coupling Between Objects (CBO);
- Weighted Methods per Class (WMC).

Also, the correlation between the values of the above metrics and the level of code reusability using the following analytic expression is determined in [6]:

$$C_r = \frac{\beta \times 100}{\alpha}, \quad (1)$$

where α is the total number of classes available for each of the aforementioned metrics; β is the number of newly redesigned classes with the help of reuse.

According to the stated assertion about the correlation between the code reusability level and the values of code structural complexity metrics, an approach to assess the reusability level based on the metrics of the structural complexity of the code is proposed in [7,11]. The evaluation process is shown in Fig. 2.

The input will be the target domain in the form of user stories. Next, based on the requirements, a domain model is constructed using each of the chosen domain-driven design methods and source code is then generated. Then the resulting code complexity is analyzed using structural complexity metrics.

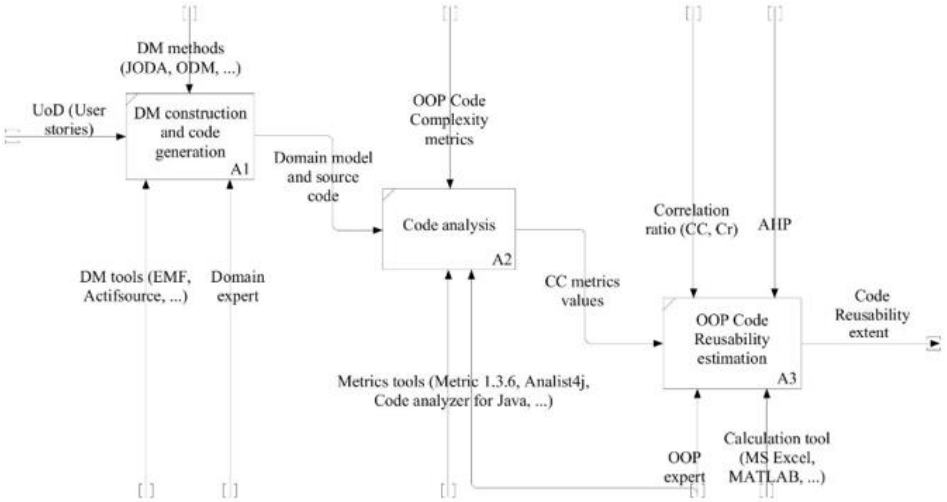

Fig. 2 The information technology to provide a code reuse assessment

In order to obtain the reusability extent, the analytical hierarchy process (AHP) is used [8], which supposes that a synthesis of priorities is to be calculated on the basis of subjective expert assessments. [5] presents statistical data on the values of structural complexity metrics and the reusability level. Applying the weight coefficients the following formula for estimating the integrated value of the reusability level can be used:

$$CR_{extent} = 0{,}1198WMC + 0{,}0398RFC + 0{,}2801DIT + 0{,}3603NOC + 0{,}2CBO \qquad (2)$$

The approach presented in the formula (2) allows computing the numerical results of assessing the complexity at different variation points. Therefore, it is possible to provide the appropriate data for a further decision on the particular option usage in any module of DSPL to be developed.

4 The proposed approach to assessment of code reusability in DSPL development

Software, specifically in mobile DSPL, supposes to provide the appropriate options in different variability points, so that the complexity of managing the amount of variability becomes a main problem in their development. In [9] it is recognized that a unified approach to variability management still has to be defined.

There are four characteristics that are essential to this approach and therefore have to be defined: consistency, scalability, traceability and means for visualization.

Software engineering for single systems has thus far been done in two dimensions. One dimension represents the phases of software lifecycle and the other, levels of abstraction. All development artifacts can be placed somewhere in these dimensions. With the addition of variability in software product line engineering, the third dimension is added that explicitly captures variability information between members of the product line.

One of the most effective ways to resolve this problem is reusing different project solutions (assets): domain knowledge, requirements specifications, software architectures, design patterns, and finally source (program) code.

The conceptual variability model that addresses traceability of variations at different levels of abstraction and across various generic development artifacts will be presented later. In this model different software development stages will be shown with the respect to variability modeling space and level of abstraction.

The information technology to support the approach to evaluation of code complexity based on (1) can be modified for the purposes of DSPL. It should also be noted that for the final conclusion on the effectiveness of using one or another variability method in order to increase the level of reuse CR_{extent}, it is also necessary to estimate the costs associated with the construction of the appropriate domain models in the process of developing SPL[10].

In order to do so it is necessary to introduce the concept of variation mechanism. These mechanisms are applied at the source code level so it will be done just after the code generation. Let us decompose the activity A1 in order to show where the variation mechanisms will appear (Fig. 3).

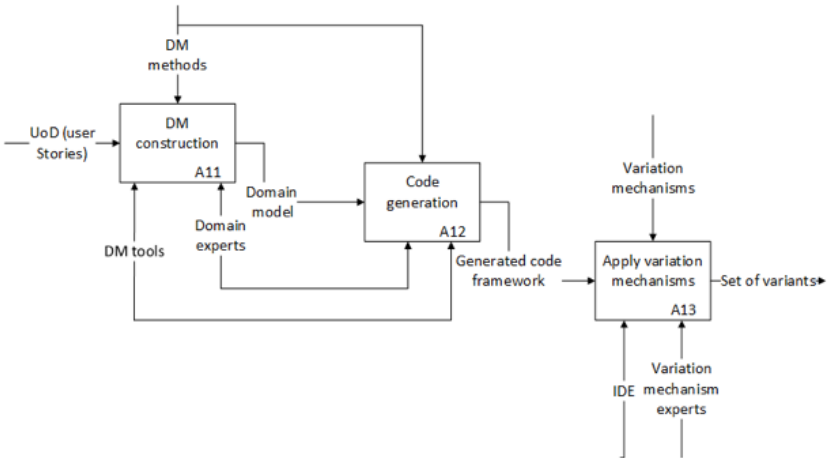

Fig. 3 Introducing variation mechanisms

So after “Code generation” stage, the variability is introduced to the generated code framework. To apply variation mechanisms they are the first to be selected. To select appropriate mechanisms variation mechanism experts are to be consulted with. As a result the approach analyzes not just the generated code framework, but variants obtained after application of some variation mechanisms.

In the next chapter variation mechanism and dynamic adaptation techniques will be analyzed in the context of DSPL implementation on the Android (Java platform) with respect to the code reusability.

5 Case study: dynamic variability used to implement the prototype

As mentioned in the previous chapter, the experiment to support the proposed approach to the assessment of reusability extent has been carried out. For this purpose the sophisticated class has been selected from the developed DSPL, where the variation mechanisms are applied at runtime, namely ServerSelectionActivity. The purpose of this class is to let a user choose the server endpoint or manually define it, which can lead to different system behavior depending on the selected server.

As already mentioned, we have decided to choose 2 variation mechanism types for the experiment: runtime conditional (implemented in Java through “if” statement) and inheritance. These mechanisms have been chosen because of relative simplicity of the implementation and also they affect the value of metrics that are used to calculate the reusability extent with the help of the proposed approach. Runtime conditionals affect WMC and inheritance affects DIT.

Also we have decided to choose one more variation mechanism, namely the configurator, and show how little it affects the code reusability. In this case the configurator has been implemented in the form of a property file.

The visual representation of the possible variants after application of these variation mechanisms is given in Fig. 4 in the form of a feature model.

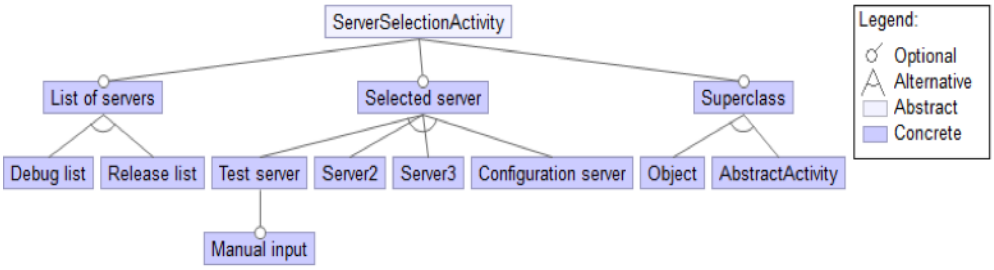

Fig. 4 Feature model for ServerSelectionActivity

The first runtime conditional determines the list of servers to be shown with respect to a build configuration: debug or release (Fig. 5). The second one determines whether the manual server definition should work (when test server is selected) or the predefined server should be used.

The third variation mechanism determines whether ServerSelectionActivity extends java.lang.Object or has an intermediate superclass AbstractActivity.

```
// we use different list of servers for debug and release v
if (BuildConfig.DEBUG) {
    serversArrayId = R.array.servers_array_debug;
    serverNamesArrayId = R.array.servers_names_array_debug;
} else {
    serversArrayId = R.array.servers_array;
    serverNamesArrayId = R.array.servers_names_array;
}
```

Fig. 5 Assigning different server list based on version type

The fourth variation mechanism determines the default server to be used if nothing is selected. Two ways of implementing this mechanism can be observed: property file (Fig. 6) and configuration class (Fig. 7). It is worth mentioning that the second approach cannot be considered as a runtime adaptation technique, because it is a compile-time case. The absence of this property file means that these

properties are configured in fields. Using the configuration class only moves these fields out of this class to the other class. In this case one more dependency appears and the CBO value changes.

```
#Configuration file
protocol = http
ip_port = 192.168.0.100:8080
folder = test
```

Fig. 6 Configurator based on property file

```
public class Configurator {

    public static final String protocol = "http";
    public static final String ipPort = "192.168.0.100:8080";
    public static final String folder = "test";
}
```

Fig. 7 Configurator based on configuration class

The experiment has been conducted in the following way. First, all the variation mechanisms have been removed and the code reusability extend (CRE) has been calculated for this situation. Then the CRE have been determined for each variation mechanism separately and finally they have been applied simultaneously. To calculate the appropriate values of CRE the software tool presented in the previous section have been used.

The results of the experiments are shown in Table 1.

Table 1 – Results of the experiments

Variation mechanisms / Complexity metrics	WMC	RFC	DIT	CBO	NOC	CRE
Without variability mechanisms	2	1	1	3	0	1.159
With runtime conditionals	5	1	1	3	0	1.518
With inheritance	2	1	2	3	0	1.439
With configurator (property file)	2	1	1	3	0	1.159
With configurator (configuration class)	2	1	1	4	0	1.359
Together (with property file)	5	1	2	3	0	1.79
Together (with configuration class)	5	1	2	4	0	1.999

To sum it up, applying variation mechanisms increases the CRE. Moreover if applied simultaneously they provide for higher CRE than when applied separately. Applying runtime conditionals and inheritance separately in the case of this class shows similar results. Applying the configurator implemented as a configuration class gives a higher CRE than the one implemented as a property file. It results from the fact that obtained results don't take the property file into account, as it is not a part of the code. The results of the experiment may help developers to solve the problem of variation mechanism selection.

6 Results and future work

With respect to external validity, the main issue is that the experiment has been conducted in context of Android Java platform. There is the possibility that implementing different variation mechanisms in other platforms, such as .NET, PHP, will result in different CRE values. So, this work does not answers the question whether the same results can be applied to other platforms and programming languages.

The second issue in the category of external validity is the CRE formula introduced in the paper. Since it takes into account only code complexity metrics on class or package level, it cannot be applied to some variation mechanisms like aspects or plugins.

Internal validity focuses on how sure a researcher can be that the results of the experiment unambiguously determine which variation mechanisms should be selected. Code reusability is not the only criteria according to which the given mechanism should be chosen. The selection of an appropriate variation mechanism also depends on the application domain where DSPL has to be designed, on specific user' requirements, etc. and these issues can be the focus of our future work.

Acknowledgments

We would like to thank student of NTU "Kharkiv Polytechnic Institute" Andrii Tkachuk for his support in our research activities, and for the software implementation for the testing of our approach.

REFERENCIES

1. Sharma Y., "Emerging trends in mobile apps market and their potential impact on mobile users engagement in the global economy", Sharma Y., Kumar B., Nivadit A./ *Annual Research Journal of SCMS* / Vol. 5, March 2017
2. Eleutério J., "A Comparative Study of Dynamic Software Product Line Solutions for Building Self-Adaptive Systems", *Technical Report in Eniversidade estadual de Campinas*, 2017. – 30 p.
3. A. M. Sharifloo, A. Metzger, C. Quinton, L. Baresi and K. Pohl, "Learning and Evolution in Dynamic Software Product Lines," *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Austin, TX, 2016, pp. 158-164.
4. Wenhao W. Master thesis: "React Native vs Flutter, cross-platform mobile application frameworks", *Metropolia University of Applied Sciences.* / 01 March 2018.
5. Nandakumar A.N., "Constructing Relationship between Software Metrics and Code Reusability in Object Oriented Design", *International Journal of Advanced Computer Science and Applications*, Vol. 7, No. 2, 2016.
6. Fangjun Wu, Tong Yi, "A Structural Complexity Metric for Software Components", *First International Symposium on Data Privacy and e-Commerce*, pp. 161-163, 2007.
7. Tkachuk M., "An Inte{grated Approach to Evaluation of Domain Modeling Methods and Tools for Improvement of Code Reusability in Software Development", Tkachuk M., Martinkus I., Gamzayev R., Tkachuk A. // Heinrich C. Mayr, Martin Pinzger (Eds.): *INFORMATIK 2016, Lecture Notes in Informatics*, Vol. P-259: Kollen Druck+Verlag GmbH, Bonn, pp. 143-156, 2016.
8. Saaty T. L., "Relative Measurement and its Generalization in Decision Making: Why Pairwise Comparisons are Central in Mathematics for the Measurement of Intangible Factors - The Analytic Hierarchy/Network Process", *RACSAM (Review of the Royal Spanish Academy of Sciences, Series A, Mathematics)*, 2008.
9. Berg K., "A Critical Analysis of Using Feature Models for Variability Management", Berg K., Muthig D., *Submitted to SPLC-Europe* 2005.
10. Martinkus I., "Designing software product lines using domain modeling and code reuse metrics", Iryna Martinkus, Mykola Tkachuk, Rustam Gamzaev, *Systems of control, navigation and communication*,Vol. 3(43)., 2017. – pp. 93-97.
11. Mykola Tkachuk, Rustam Gamzaev, Iryna Martinkus et al., "Towards Effectiveness Assessment of Domain Modelling Methods and Tools in Software Product Lines Development", *Enterprise Modelling and Information Systems Architectures – International Journal of Conceptual Modeling*, Vol. 13 (2018), Germany. - pp. 190-206.

ЛІТЕРАТУРА

1. Sharma Y., "Emerging trends in mobile apps market and their potential impact on mobile users engagement in the global economy", Sharma Y., Kumar B., Nivadit A./ *Annual Research Journal of SCMS* / Vol. 5, March 2017
2. Eleutério J., "A Comparative Study of Dynamic Software Product Line Solutions for Building Self-Adaptive Systems", *Technical Report in Eniversidade estadual de Campinas*, 2017. – 30 p.

3. A. M. Sharifloo, A. Metzger, C. Quinton, L. Baresi and K. Pohl, "Learning and Evolution in Dynamic Software Product Lines," *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Austin, TX, 2016, pp. 158-164.
4. Wenhao W. Master thesis: "React Native vs Flutter, cross-platform mobile application frameworks", *Metropolia University of Applied Sciences.* / 01 March 2018.
5. Nandakumar A.N., "Constructing Relationship between Software Metrics and Code Reusability in Object Oriented Design", *International Journal of Advanced Computer Science and Applications*, Vol. 7, No. 2, 2016.
6. Fangjun Wu, Tong Yi, "A Structural Complexity Metric for Software Components", *First International Symposium on Data Privacy and e-Commerce*, pp. 161-163, 2007.
7. Tkachuk M., "An Inte{grated Approach to Evaluation of Domain Modeling Methods and Tools for Improvement of Code Reusability in Software Development", Tkachuk M., Martinkus I., Gamzayev R., Tkachuk A. // Heinrich C. Mayr, Martin Pinzger (Eds.): *INFORMATIK 2016, Lecture Notes in Informatics*, Vol. P-259: Kollen Druck+Verlag GmbH, Bonn, pp. 143-156, 2016.
8. Saaty T. L., "Relative Measurement and its Generalization in Decision Making: Why Pairwise Comparisons are Central in Mathematics for the Measurement of Intangible Factors - The Analytic Hierarchy/Network Process", *RACSAM (Review of the Royal Spanish Academy of Sciences, Series A, Mathematics)*, 2008.
9. Berg K., "A Critical Analysis of Using Feature Models for Variability Management", Berg K., Muthig D., *Submitted to SPLC-Europe* 2005.
10. Martinkus I., "Designing software product lines using domain modeling and code reuse metrics", Iryna Martinkus, Mykola Tkachuk, Rustam Gamzaev, *Systems of control, navigation and communication*,Vol. 3(43)., 2017. – pp. 93-97.
11. Mykola Tkachuk, Rustam Gamzaev, Iryna Martinkus et al., "Towards Effectiveness Assessment of Domain Modelling Methods and Tools in Software Product Lines Development", *Enterprise Modelling and Information Systems Architectures – International Journal of Conceptual Modeling*, Vol. 13 (2018), Germany. - pp. 190-206.

Гамзаєв Рустам Олександрович *– кандидат технічних наук, доцент; Харківський національний університет імені В.Н. Каразіна, м. Харків, пл. Свободи 4, 61022; e-mail: rustam.gamzayev@gmail.com ; ORCID: 0000-0002-2713-5664.*

Ертугрул Карачуа (Ertuğrul Karaçuha) – *доктор наук, професор; Стамбульський технічний університет (*Istanbul Technical University*), м. Стамбул, Турція; İTÜ Ayazağa Kampüsü; e-mail: karacuhae@itu.edu.tr; ORCID: https://orcid.org/0000-0002-7555-8952.*

Ткачук Микола Вячелавович *– доктор технічних наук, професор; Харківський національний університет імені В.Н. Каразіна, м. Харків, пл. Свободи 4, 61022; e-mail: tka.mobile@gmail.com ; ORCID: 0000-0003-0852-108.*

Товстокоренко Олег Юрійович *– аспірант; Національний технічний університет «Харківський політехнічний інститут», вул. Кирпичова, 2, Харків, 61002 Україна; e-mail: tovstokorenko@gmail.com ; ORCID: 0000-0003-2664-1650.*

Надійшла –10.11.2018.