DOI: 10.26565/2304-6201-2019-41-06

UDC 519.72

The impact of usage of post object-oriented technologies on defect reduction in software maintenance

K.A. Nagornyi[1], I.O. Martinkus[1], M.V. Tkachuk [1]
[1]V. N. Karazin Kharkiv National University, Svobody Square 4, Kharkov, 61022, Ukraine
e-mail: imartinkus@gmail.com

The article is dedicated to software quality improvement research within the maintenance phase based on post-object-oriented technologies. An important problem of the maintenance phase is surveyed, namely, the crosscutting functionality problem. Mechanisms of post-object-oriented technologies have been reviewed and basic tasks to be resolved have been formulated in order to reach the final goal of the research: defect reduction during the maintenance phase. The post object-oriented technologies utilization framework for software quality improvement based on a collection of 4 heuristic assumptions has been introduced. The conceptual scheme of the framework has been presented. An applied 2-steps procedure for defect reduction assessment based on quantitative crosscutting-functionality and defect metrics has been described. Twelve results of the experiments concerning calculation of the residual defect number have been presented and analyzed.

Keywords: *post object-oriented technology, crosscutting functionality, software, defect, maintenance, metric.*

Стаття присвячена дослідженню підвищення якості розробки програмного забезпечення на фазі супроводу із використанням пост об’єктно-орієнтованих технологій. Однією із особливостей фази супроводу програмного забезпечення є постійна зміна вимог користувачів, що на рівні вихідного коду призводить до проблеми наскрізної функціональності, яка в свою чергу викликає зростання рівня дефектів. Для вирішення проблеми наскрізної функціональності запропоновано використати механізми пост об’єктно-орієнтовані технологій та сформульовані базові завдання для досягнення фінальної цілі дослідження: зниження рівня дефектів під час фази супроводу. Представлений фреймворк для підвищення якості програмних систем із використанням пост об’єктно-орієнтованих технологій, який базується на концепті 3-вимірного простору, що поєднує у собі тип системи та вид пост об’єктно-орієнтованої технології, із використанням якої передбачається зниження рівня наскрізної функціональності у цільовій системі. Даний фреймворк базується на чотирьох базових евристичних припущеннях, що пов’язують рівень дефектів, наскрізну функціональность, пост об’єктно-орієнтовані технології та цільову програмну систему. Наведена концептуальна схема та запропонована двошагова процедура із використання даної пост об’єктно-орієнтованої технології для зменшення рівня дефектів на фазі супроводу цільової програмної системи. Процедура передбачає отримання кількісних оцінок на базі метрик наскрізної функціональності, таких як: рівень присутності наскрізної функціональності CF_{ratio}, її ступінь розсіювання DOS, залишковий рівень наскрізної функціональності RCR_{ratio}, та кількість дефектів у цільовій програмній системі NoD. Наведені дані 12 практичних експериментів із визначення залишкового рівня дефектів, що згруповані за чотирма типіми програмних систем та трьома пост об’єктно-орієнтованими технологіями, такими як: аспектно-орієнтована технологія, властивість-орієнтована технологія та контекстно-орієнтована технологія. Результати відповідних експериментів були проаналізовані, що дало змогу проранжувати згадані вище пост об’єктно-орієнтовані технології за залишковим рівнем дефектів в досліджених системах та визначити найменш та найбільш ефективну технологію для використання в процесі супроводу програмних систем.

Ключові слова: *пост об’єктно-орієнтовані технології, наскрізна функціональність, програмне забезпечення, дефекти, супроводжуваність, метрики.*

Статья посвящена исследованию повышения качества разработки программного обеспечения на фазе сопровождения с использованием пост объектно-ориентированных технологий. Одной из особенностей фазы сопровождения программного обеспечения является постоянное изменение требований пользователей, на уровне исходного кода приводит к проблеме сквозной функциональности, которая в свою очередь вызывает рост уровня дефектов. Для решения проблемы сквозной функциональности предложено использовать механизмы пост объектно-ориентированные технологии и сформулированы базовые задачи для достижения финальной цели исследования: снижение уровня дефектов во время фазы сопровождения. Представленный фреймворк для повышения качества программных систем с использованием пост объектно-ориентированных технологий, основанный на концепте 3-мерного пространства, сочетает в себе тип системы и вид пост объектно-ориентированной технологии, с использованием которой предполагается снижение уровня сквозной функциональности в целевой системе. Данный фреймворк базируется на четырех базовых эвристических предположениях, связывающих уровень дефектов, сквозную функциональность, пост объектно-ориентированные технологии и целевую программную систему. Приведенная концептуальная схема и предложена двухєтапная процедура по использованию данной пост объектно-ориентированной технологии для уменьшения уровня дефектов на фазе сопровождения целевой программной системы. Процедура предусматривает получение количественных оценок на базе метрик сквозной функциональности, таких как: уровень присутствия сквозной функциональности CFratio, ее степень рассеивания DOS, остаточный уровень сквозной функциональности RCRratio, и количество дефектов в целевой программной системе NoD. Приведенные данные 12 практических экспериментов по определению остаточного уровня дефектов, сгруппированы по четырем типам программных систем и тремя пост объектно-ориентированными технологиями, такими как: аспектно-ориентированная технология, свойство-ориентированная технология и контекстно-ориентированная технология. Результаты соответствующих экспериментов были проанализированы, что позволило проранжировать упомянутые выше пост объектно-ориентированные технологии по остаточному уровню Дефекты в исследованных системах и определить наименее и наиболее эффективную технологию для использования в процессе сопровождения программных систем.

Ключевые слова: *пост объектно-ориентированные технологии, сквозная функциональность, программное обеспечение, дефекты, супроводжуванисть, метрики.*

1 Introduction. Problem actuality and research goals

Nowadays object-oriented programming (OOP) is the most popular technology for software system development and maintenance [1]. For the maintenance of legacy software systems (LSS) one of the important problems is continuous modifications of major part of their sub-systems due to changes in user requirements and the development of new components to meet new user requirements. Permanent changes in LSS components lead to design instability which causes a so-called crosscutting functionality (CF) problem [2; 3]. The OOP approach increases source code complexity and does not resolve this issue in effective way, especially during a maintenance process of large legacy software systems.

During the last two decades post object-oriented technologies (POOT) have emerged and been intensively designed. The most known fully-fledged POOT are: aspect-oriented software design (AOSD) [4], feature-oriented software design (FOSD) [5] and context-oriented software development (COSD) [6]. These POOTs use core principles of the object-oriented software design but additionally include a complementary feature-set to resolve the crosscutting functionality problem. On the other hand, utilization of any of mentioned POOT for LSS maintenance results in extra time and efforts interconnecting software development. Hence most of researchers accentuate the necessity to elaborate approaches for the complex estimation of POOT’s effectiveness usage in real-life software projects, see e.g. in [7; 8; 9]. Besides, the issues of relationship between specific features of different POOTs and software design defects caused by CF [10] as well as development of such sophisticated solutions as software product lines with FOSD [11] and evaluation of software quality with usage of AOSD [12] are presented and discussed intensively nowadays. Nevertheless a lack of applied researches related to the impact of POOTs on software defects behavior with respect to the factor of crosscutting functionality has to be emphasized.

Taking into account the above-mentioned issues the research goal of this paper is to assess the impact of different POOTs on defect reduction in software maintenance and to provide some methodological recommendations for choosing an appropriate POOT in real-life projects. In order to reach this research goal the following tasks are to be resolved:

- to analyse some critical CF-issues in maintenance of LSS by utilizing traditional OOP-methodology;
- to give a short review of the knowledge-oriented approach to the estimation of POOTs effectiveness;
- to define metrics to estimate a CF level and a method to calculate a number of software defects in LSS;
- to propose the conceptual scheme of quality improvement of software maintenance using POOTs;
- to check the proposed approach experimentally, to analyse the results obtained and to formulate some praxis-oriented recommendation for usage of POOTs in maintenance of different LSS.

The possible solutions for these tasks are presented below in more details, as well as, the outlook of the next steps concerning this research.

2 A framework for usage of post object-oriented technologies to improve a software quality in software maintenance.

A lot of studies investigate complexity of the maintenance process of OOP-based legacy software systems [13; 14], especially a crosscutting functionality (CF) problem. The crosscutting problem is a functionality which can’t be modularized at the source code level, although represents a particular businesses feature from the requirement perspective. Some well-known representatives of the CF are: data validation, transaction management, logging, exception handling, etc. Complexity of the maintenance process of a software system which includes the CF increases dramatically [14]. There are some peculiarities which are common to those crosscutting features, namely: complication of software requirements traceability; decrease of readability and understandability of diverse design artifacts; source code redundancy; lack of modularity which prevents further reuse of such CF-solutions.

Separation of Concerns (SoC) [10] is a principle which resolves CF problem. It presents a decomposition phase and a non-invasive composition phase of the CF-source code and the LSS basic source code. At the decomposition phase the source code of the crosscutting functionality should be localized, extracted and isolated, from the rest of the LSS-code, into well-structured modules (CF-modules). The composition phase involves reassembling of well-structured CF-modules with the LSS CF-free modules. Realization of the SoC principle allows solving CF-peculiarities listed above and implementing software system configuration in order to add or remove functionality, if it is required.

As mentioned in Section 1, there are three fully-fledged approaches in POOT-domain, namely: AOSD, COSD, FOSD. To represent main features of these approaches an interaction between basic OO-components and POOT-components should be restated [15]. AOSD was proposed about two decades ago in Xerox PARC research center, and now it is implemented in vast majority of programming languages, like Java, .Net, C++, JavaScript and so on. CF-related source code should be isolated in a special module, which is called the aspect. Further composition of this module and non-crosscutting functionality source code is based on the idea of intersection point – the point-cut and the injection. Schematically this interaction is shown in Fig. 1(a), where the white vertical rectangles C1, C2, C3 represent OOP-classes and gray horizontal rectangles A1, A2, A3 represent the aspects.

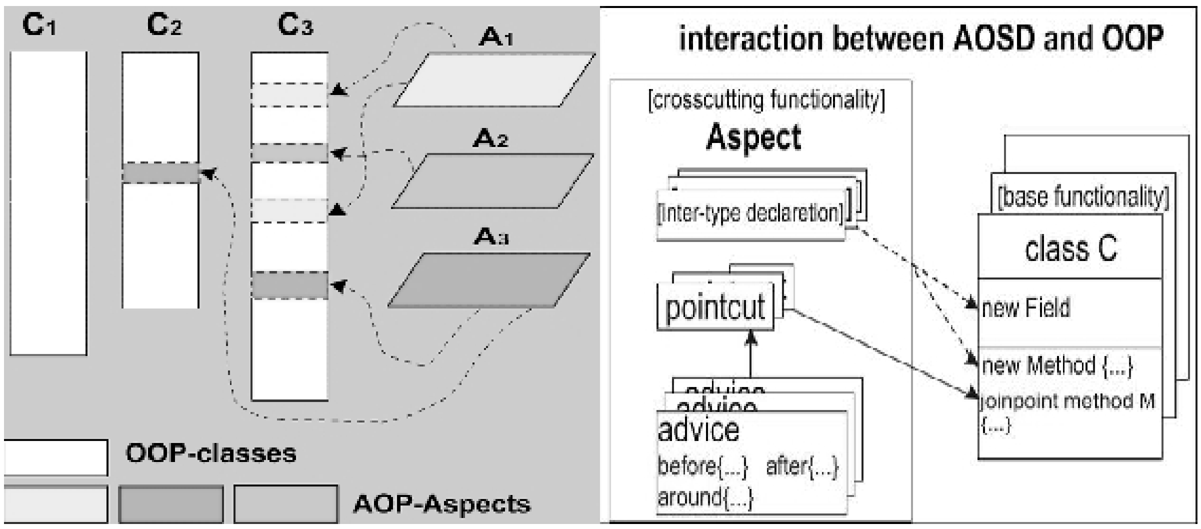

Fig. 1 AOSD: (a) – the conceptual scheme; (b) – the implementation facets [15]

A specific structure of the aspect is represented in Fig. 1(b). It includes point-cut, advice and inner-type declarations. The core function of point-cut is to define a set of join points between the aspect and the basic methods in OO-classes in order to inject advice source code. Advice represents a piece of source code of former CF, in other words it is a special kind of function written in a OO-programming language (e.g. Java). There are three types of advice: before – is invoked before target method execution, after – is invoked after target method execution, around – is invoked instead of a target method execution. Also it is possible to declare additional members of a target class such as fields and methods via inner-type declarations. Two other technologies, FOSD and COSD can be represented and analyzed in the similar way (see [15] for more details).

Even a short overview of CF issues shows that to make a decision about the effectiveness of using an appropriate POOT to resolve CF-problem in a given LSS, we need to consider a number of factors, which have to be formalized and evaluated in the appropriate modeling approach. This approach is elaborated in [15] and its main idea is to use the 3-D modeling information space which is graphically shown in Fig. 2. According to this modeling framework the integrated effectiveness level of a POOT usage depends on two interplaying factors, namely: 1) what type of LSS (System Type) has to be modified with an appropriate POOT; 2) what kind of POOT is used to eliminate the CF in this LSS.

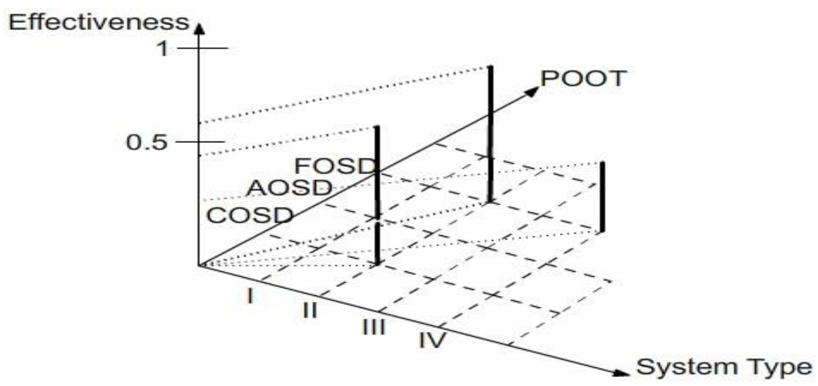

Fig. 2 The 3-D space for assessment of POOTs effectiveness [15]

In turn, an appropriate System Type for any given LSS can be defined as a set of two interconnected factors, namely: 1) the target software system Structural Complexity which can be evaluated from an appropriate collection of object-oriented source code metrics; 2) the system's Requirement Rank which depicts the complex characteristics of functional requirements in the target software system (Fig. 3).

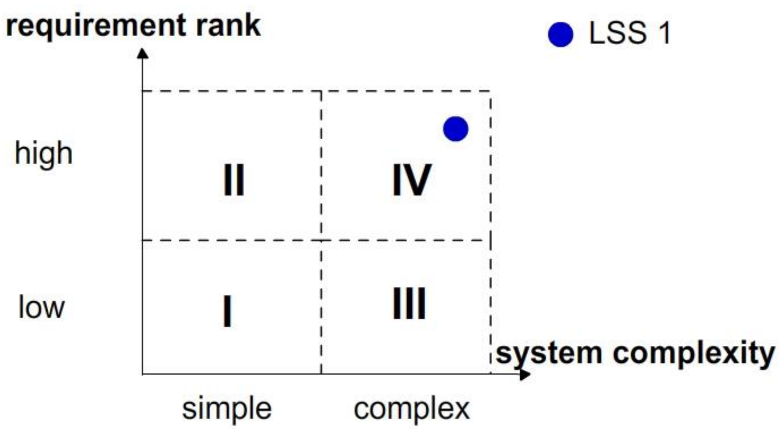

Fig. 3 The 2-D space to define the System Type for the target LSS

More details concerning the framework for estimation of effectiveness coefficient of POOTs usage in maintenance of different types of LSS can be found in [15].

In order to realize our research goal: to elaborate the approach to assessing an impact of POOT usage on defect reduction in software maintenance, we propose to formulate a collection of heuristic assumptions. Those assumptions are based on the study of modern information sources and on the generalization of our own experience of using different POOTS in software development and maintenance.

Assumption 1. A set ***T*** of post object-oriented technologies for software development does exist. AOSD, FOSD, COSD belong to this set.

$$(POOT)_j \in T,\ j = 1,2,3... \tag{1}$$

Assumption 2. Each LSS might relate to some particular system type, i.e. a set of system types ***S*** does exist.

$$(SysType)_i \in S, i = 1,2,3... \tag{2}$$

Assumption 3. CF-level decrease in a target LSS is possible by using one of the existent POOT, although it is accompanied with additional costs needed for software modification of a target LSS.

$$\Delta(CFR)_{i,j} = \delta\left((SysType)_i, (POOT)_j\right), \tag{3}$$

where $\Delta\left(CFR\right)_{i,j}$ is a ratio of CF-level decrease caused by usage of a j-th POOT within i-th LSS-type maintenance; $\delta()$ is a functional dependency between the corresponding values.

Assumption 4. There exists a positive correlation between crosscutting functionality level and an average number of defects in a target LSS, which are presented during the LSS maintenance process, i.e. there exists some functional dependency $\mu()$:

$$NoD\left(\left(SysType\right)_i,\left(POOT\right)_j\right)=\mu\left(\Delta\left(CFR\right)_{i,j}\right). \tag{4}$$

It should be mentioned that the functions δ and μ cannot be defined in an analytical way, because there are a lot of weakly-formalized and complicated factors which characterize software development process in general, and the LSS maintenance phase in particular. Therefore we have decided to construct the appropriate metrics and to elaborate a procedure of assessing the impact of POOTs on possible software defects reduction in maintenance of LSS.

Taking into account the assumptions (1) - (4) mentioned above, we propose a conceptual scheme for usage an appropriate POOT in order to reduce a number of defects during maintenance of a given LSS. The elaborated conceptual scheme is shown on Fig. 4, and it can be described briefly as a set of the following steps:

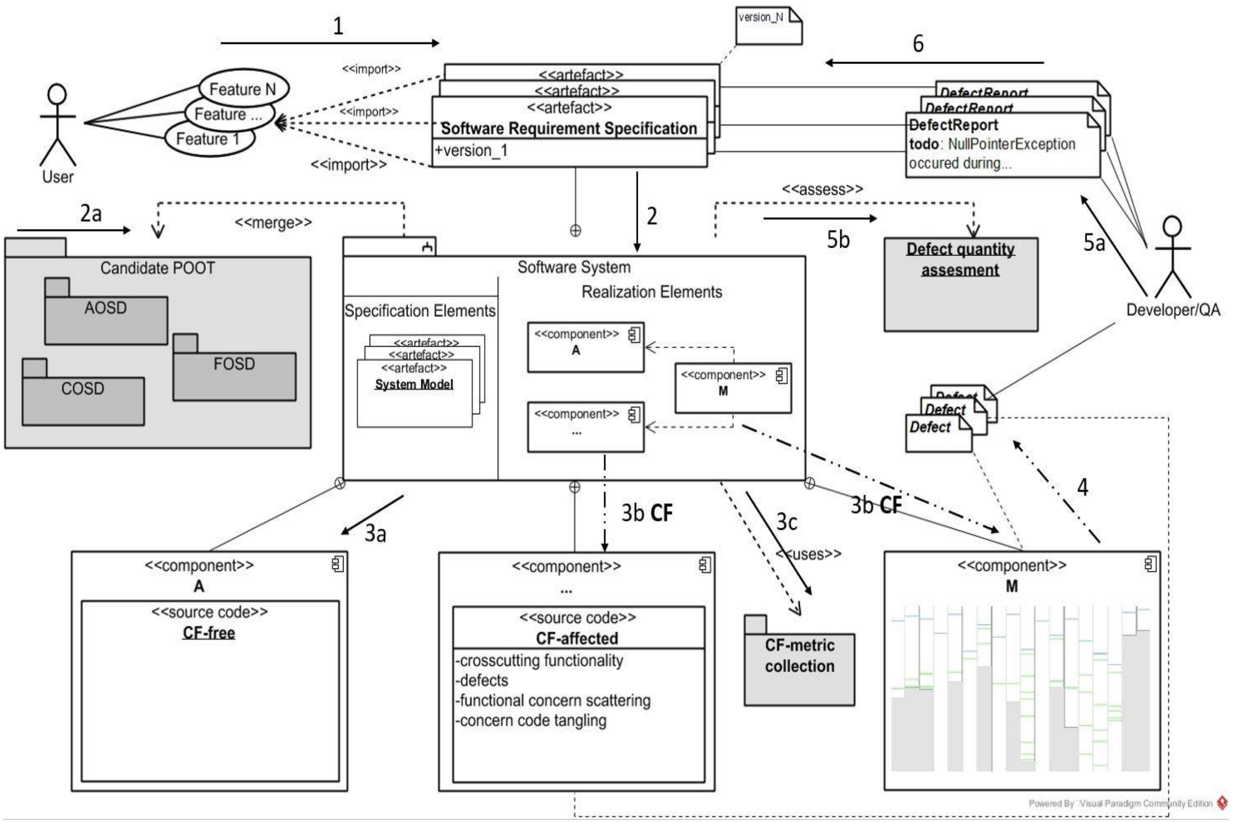

Fig. 4 The proposed conceptual scheme

1. Some User (presented as a role on the scheme) requires some additional/new features to be added to a target LSS. These user's requirements have to be transformed into a new version of a software requirement specifications (SRS).

2. A given version of SRS precipitates the development a new version of a target LSS which embraces a set of project artifacts: a system model (e.g. architectural model, information model, etc.) and its realization, namely: a set of target software components.

3. A set of developed software components (source code) can be divided into the following groups, namely:

a) a subset of components which are "healthy", that is, they do not contain source code with CF;
b) a subset of components "infected" with a CF-code which leads to code scattering and tangling with a source code of another functional concern;
c) a collection of CF-focused metrics to be constructed in order to assess a level of CF, its negative factors, etc.;
d) using this metrics an appropriate CF-status of a given LSS has to be defined.

4. A current CF-status in LSS leads to some software defects to be fixed by Developer/Quality Assurance (QA) shown in conceptual scheme (Fig. 4).

5. A Developer/QA takes a care of assessing the number of defects, namely
a) Completion of a defect reports;
b) Computing a number of defects.

6. Basing on a given defect report the appropriate changes can be done in the next SRS versions.

The steps (1) - (6) in the real LSS maintenance cycle have to be repeated iteratively. Basing on the results obtained from 3.c (usage of a CF-metrics collection) and 5.b (assessing a number of defects), an effective POOT (from AOSD, COSD, and FOSD respectively) can be chosen within 2.a (see these logically interconnected icons shown in grey in Fig. 4).

3 The 2-steps procedure to assess the defect reduction with usage of different POOTs in software maintenance

According to the proposed conceptual scheme (Fig. 4), and taking into account formulas (3) and (4) in Section 2, it is possible to construct the quantitative metrics and to elaborate the appropriate procedure to

1) assess a crosscutting functionality level in a given LSS;

2) compute a number of software defects in this system before and after the usage of an chosen POOT.

To perform these 2 steps we have to localize source code which includes a particular CF in a given LSS, and for this purpose we could use some already existing source code analysis tools for CF localization, CIDE, for example [16]. After that it is possible to define a specific crosscutting coefficient of a particular CF in the system indicated as CF_{ratio}. This coefficient shows a ratio between OOP-classes "damaged" by a particular CF and all other OOP-classes in the given LSS, e.g. business logic realization without subordinate classes of a framework. This coefficient can be represented as:

$$CF_{ratio} = \frac{C_{cf}}{C_{cf} + C} \tag{5}$$

where C_{cf} is a number of programming classes in LSS with CF, C is a number of classes free of CF. Obviously, that $CF_{ratio} \in [0;1]$, and if $CF_{ratio} = 0$, a particular functionality is free from crosscutting; and if $CF_{ratio} = 1$, all programming classes are "damaged" with a particular CF.

After obtaining CF_{ratio} it is possible to calculate a residual crosscutting ratio (RCR) indicated as RCR_{ratio}. This metric, based on DOS (Degree of Scattering) value is proposed in [15]. But this metric actually does not allow assessing a "damage" degree caused by a particular CF, therefore we propose to refine DOS-metric in the following way

$$RCR_{ratio} = DOS \cdot CF_{ratio} \tag{6}$$

where DOS is a Degree of Scattering; and a CF_{ratio} is a specific crosscutting weight ratio of a particular CF. Similarly to CF_{ratio}, a value of $RCR_{ratio} \in [0;1]$, and if $RCR_{ratio} = 0$, CF is localized in a separate module and there is no more crosscutting; if $RCR_{ratio} = 1$, CF affects a whole LSS, and is uniformly distributed. Thus the proposed quantitative metrics (5) and (6) give to an expert a possibility to assess a distribution nature of a CF and to estimate a "CF-damage" for a whole given LSS.

As a next step we need to compute a number of defects (NoD) in a given LSS, and this can be done directly by using a special function known as a DefectCount (), namely

$$NoD = DefectCount\left(LSS\right). \tag{7}$$

This function can be realized by defect tracking focused on corresponding features of the target software system before and after POOT-modification of the source code.

In our research we have used one of existent defect-tracking system as a defect collector for POOT-modified source code of features in the target LSS. Then direct observation and calculation of a number of defects have been applied to get the final result.

4 Experimental results and their analysis

Taking into account the workflow in the proposed conceptual scheme (Fig. 4) and using the quantitative metrics given in formulas (5)–(7) the appropriate computerized experiments have been performed [17]. The final results are presented briefly in Table 1 and in Fig. 5 – 6. The number of software defects has been computed basing on the project reports obtained from the defect-tracking system and grouped according to the 4 system types: I, II, III and IV.

Table 1 – Number of defects

POOT	Basic number of defects in LSS (OOP-based) NoD				Number of defects in LSS (POOT-modified version) NoD				Residual number of defects (%)			
	System type				System type				System type			
	I	II	III	IV	I	II	III	IV	I	II	III	IV
COSD	28	68	46	83	6	19	13	18	21.4	27.9	28.8	21.7
FOSD	28	68	46	83	4	24	22	56	14.3	35.3	47.8	67.4
AOSD	28	68	46	83	4	14	16	27	14.3	20.6	34.7	32.5

The first group of table columns shows NoD for LSS of corresponding types where maintenance is based on OOP approach. It is evident that when the System Type becomes more and more complex NoD increases dramatically. For further calculations this defect quantity is considered as 100%. The second group of columns represents NoD for the same LSS after their modification with usage of appropriate POOTs: COSD, FOSD and AOSD.

Fig. 5 shows the data presented in the first two groups of columns from Table 1 as a histogram, and it is obvious that NoD is decreasing with usage of any POOT for all system types (I, II, III and IV).

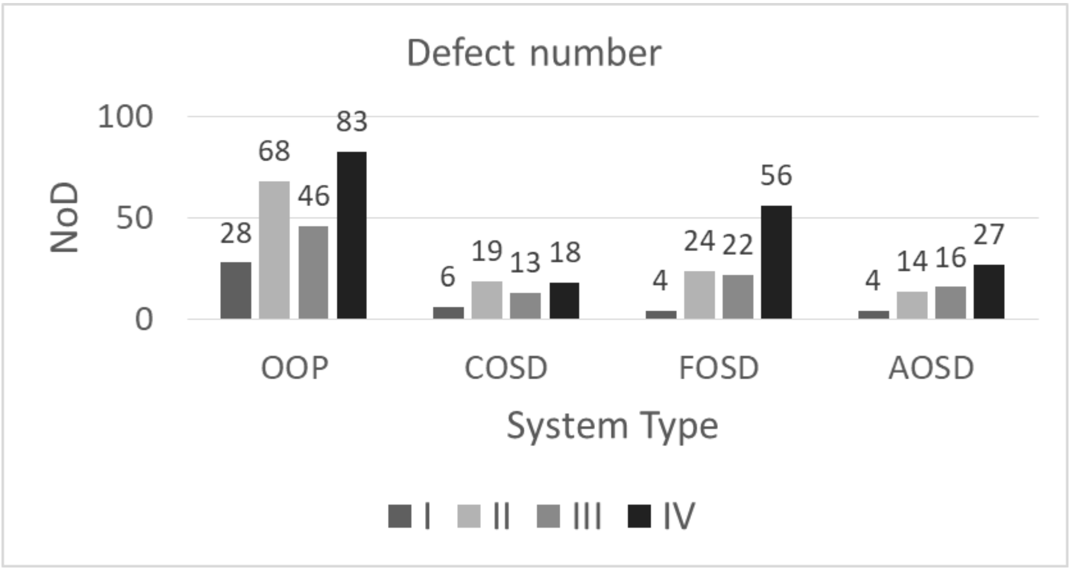

Fig. 5 Number of defects with POOTs impact

The values from the third group of columns in Table 1 are presented graphically in Fig. 5. It is possible to draw the following conclusions:

1. For LSS of the (I) and the (II) system types, all POOTs presents practically the same defect reduction. A low structural complexity of LSS could be a reason.

2. For LSS of type (III) the FOSD presents the lower defect reduction in comparison with AOSD and COSD. A difference between CF-weaving mechanisms in the corresponding POOTs is a reason.

3. For LSS of type (IV), the COSD provides the lowest level of residual number of defects, namely, about 20%.

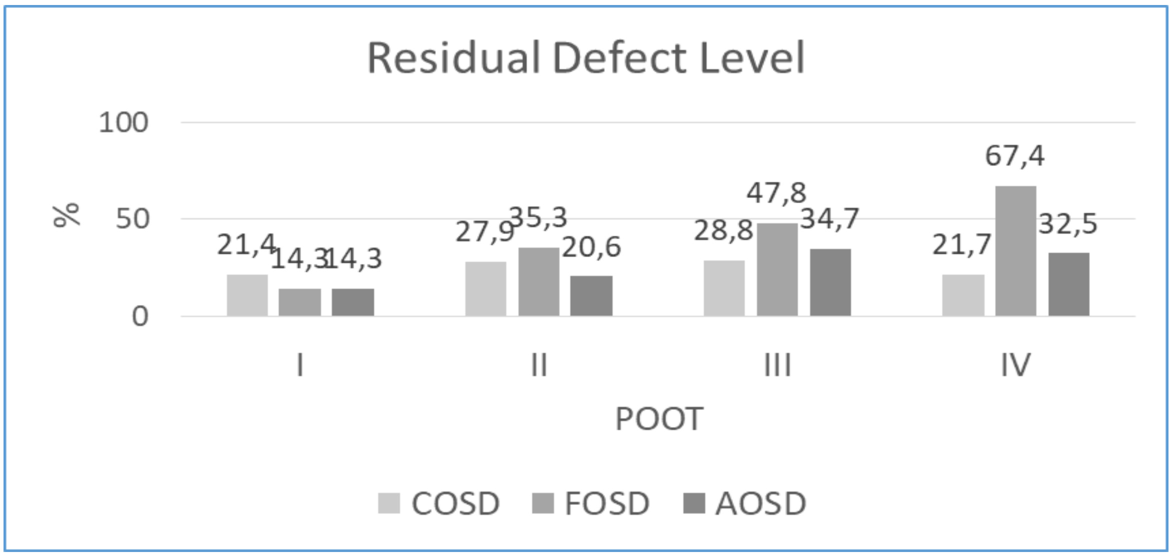

Fig. 6 Number of defects with POOTs impact

Finally, an average NoD value for all types of LSS can be computed:

$$COSD = \frac{21{,}4+27{,}9+28{,}8+21{,}7}{4} = 24{,}95;$$
$$FOSD = \frac{14{,}3+35{,}3+47{,}8+67{,}4}{4} = 41{,}20;$$
$$AOSD = \frac{14{,}3+20{,}6+34{,}7+32{,}5}{4} = 25{,}53;$$

To sum it up, COSD approach to CF-problem management has the strongest impact on a defect reduction for all System Types.

5 Conclusion and future work

This paper presents a framework for assessing the impact of post object-oriented technologies (POOT) usage on defect reduction in legacy software systems (LSS) maintenance. Some specific features of all existing POOTs, namely, AOSD, FOSD and COSD are considered. Particular attention is given to the crosscutting functionality (CF) problem. The heuristic assumptions to elaborate the assessment procedure for defects reduction are formulated. The collection of quantitative metrics to estimate of CF-level in LSS is provided. The conceptual scheme of the LSS maintenance process in respect to eliminating CF-problem by using POOTs is proposed. The scheme allows decreasing a number of software defects. The performed experiments show the impact of POOTs usage on defects reduction in real-life LSS maintenance projects.

Our future work will include advanced analysis of different defect types in LSS maintenance and detailed research of influence of each POOT on those defect types.

REFERENCES

1. I. Sommerville. *Software Engineering / 9th edition*. Addison Wesley, 2011.
2. S. Apel, et al. “On the Structure of Crosscutting Concerns: Using Aspects of Collaboration?” *In: Workshop on Aspect-Oriented Product Line Engineering*, 2006.
3. A. Przybyłek, “Post Object-oriented Paradigms in Software Development: A Comparative Analysis”, *In: Proceedings of the International Multi-conference on Computer Science and Information Technology*, pp. 1009-1020, 2007.
4. Official Web-site of Aspect-oriented Software Development community, [Online]. Available: http://aosd.net.

5. Official Web-site of Feature-oriented Software Development community, [Online]. Available: http://fosd.de.
6. Official Web-site of Context-oriented Software Development group, [Online]. Available: http://www.hpi.uni-potsdam.de/hirschfeld/cop/events.
7. S. Apel, *The Role of Features and Aspects in Software Development*. Diss., Otto-von-Guericke University Magdeburg, 2007.
8. E. Figueiredo, "Concern-Oriented Heuristic Assessment of Design Stability", PhD thesis, Lancaster University, 2009.
9. N. Tkachuk, K. Nagornyi, "Towards Effectiveness Estimation of Post Object-oriented Technologies in Software Maintenance", In: *J. Problems in Programming,* vol. 2-3 (special issue), pp.252 – 260, 2010.
10. Aversano L., Cerulo L., Penta M., Di. "The Relationship between Design Patterns Defects and Crosscutting Concern Scattering Degree: An Empirical Study", In*: J. IET Software*, vol. 3, No. 5, pp. 395–409, 2009.
11. Abilio R., Vale G., Figueiredo E., "Metrics for Feature-Oriented Programming", *Proceedings of WETSoM'16,* May 16-18, 2016, Austin, USA.
12. Mazen Ismaeel Ghareb," State of the art metrics for aspect-oriented programming", *AIP Conference Proceedings,* April 2018.
13. T. Sheldon, Kh. Jerath, H. Chung, "Metrics for Maintainability of Class Inheritance Hierarchies", In: *J. of Software Maintenance and Evolution*, Vol. 14, pp. 1-14, 2002.
14. T. Gottardi et al.: "Model-based Reuse for Crosscutting Frameworks: Assessing Reuse and Maintenance Effort", *In: J. of Software Engineering Research and Development*, pp. 1--34 (2013)
15. M. Tkachuk, "Models, Methods and Tools for Effectiveness Estimation of Post Object-Oriented Technologies in Software Maintenance" / M. Tkachuk, K. Nagorniy and R. Gamzayev // V. Yakovyna et al. (Eds.): ICTERI 2015: Revised Selected Papers, *Series title: Communications in Computer and Information Science*, Vol. 594: Springer-Verlag Berlin Heidelberg, pp. 20-37, 2016.
16. Official Web-site of CIDE-project: [Online]. Available: http://wwwiti.cs.uni-magdeburg.de/iti_db/research/cide/
17. K. Nagorniy, "Models and tools for maintenance of program systems based on post object-oriented technologies", Manuscript of PhD-dissertation thesis, NTU «Kharkiv Polytechnic Institute», Kharkiv, 2016. [in Ukrainian]

ЛІТЕРАТУРА

1. Sommerville I. Software Engineering / 9th edition. Addison Wesley, 2011.
2. Apel S. et al. On the Structure of Crosscutting Concerns: Using Aspects of Collaboration? *In: Workshop on Aspect-Oriented Product Line Engineering*, 2006.
3. Przybyłek A. Post Object-oriented Paradigms in Software Development: A Comparative Analysis. *In: Proceedings of the International Multi-conference on Computer Science and Information Technology*. 2007. pp. 1009-1020.
4. Official Web-site of Aspect-oriented Software Development community, [Online]. Available: http://aosd.net.
5. Official Web-site of Feature-oriented Software Development community, [Online]. Available: http://fosd.de.
6. Official Web-site of Context-oriented Software Development group, [Online]. Available: http://www.hpi.uni-potsdam.de/hirschfeld/cop/events.
7. Apel S. The Role of Features and Aspects in Software Development. Diss., Otto-von-Guericke University Magdeburg, 2007.
8. Figueiredo. E. Concern-Oriented Heuristic Assessment of Design Stability: PhD thesis. Lancaster University, 2009.
9. Tkachuk N., Nagornyi K. Towards Effectiveness Estimation of Post Object-oriented Technologies in Software Maintenance. In: *J. Problems in Programming.* vol. 2-3 (special issue). 2010. pp.252 – 260.
10. Aversano L., Cerulo L., Penta M., Di. The Relationship between Design Patterns Defects and Crosscutting Concern Scattering Degree: An Empirical Study. In*: J. IET Software*. vol. 3, No. 5. 2009. pp. 395–409.

11. Abilio R., Vale G., Figueiredo E. Metrics for Feature-Oriented Programming. *Proceedings of WETSoM'16,* May 16-18, 2016. Austin. USA.
12. Mazen Ismaeel Ghareb. State of the art metrics for aspect-oriented programming. *AIP Conference Proceedings.* April 2018.
13. Sheldon,T., Jerath Kh., Chung H. Metrics for Maintainability of Class Inheritance Hierarchies. In: *J. of Software Maintenance and Evolution*. Vol. 14. 2002. pp. 1-14.
14. Gottardi T. et al.: Model-based Reuse for Crosscutting Frameworks: Assessing Reuse and Maintenance Effort. *In: J. of Software Engineering Research and Development*. 2013. pp. 1-34.
15. Tkachuk M. Models, Methods and Tools for Effectiveness Estimation of Post Object-Oriented Technologies in Software Maintenance / M. Tkachuk, K. Nagorniy and R. Gamzayev // V. Yakovyna et al. (Eds.): ICTERI 2015: Revised Selected Papers, *Series title: Communications in Computer and Information Science*, Vol. 594: Springer-Verlag Berlin Heidelberg, 2016. pp. 20-37.
16. Official Web-site of CIDE-project: [Online]. Available: http://wwwiti.cs.uni-magdeburg.de/iti_db/research/cide/.
17. Нагорний К.А. Моделі та інструментальні засоби супроводу програмних систем на основі пост об’єктно-орієнтованих технологій: Автореф. Дис. на здобуття вченого ступеня кандидата технічних наук за спеціальністю: 05.13.06 – Інформаційні технології. Національний технічний університет «Харківський політехнічний інститут». Харків, 2016.

Nagornyi Kostiantyn A. *– PhD, Candidate of Technical Sciences; Associate Professor of Department of System and Technologies Modeling, V. N. Karazin Kharkiv National University, Svobody Sq 4, Kharkiv - 22, Ukraine, 61022;e-mail: k.nagornyi@gmail.com; ORCID: https://orcid.org/0000-0001-5948-3682*

Martinkus Iryna O. *– PhD, Candidate of Technical Sciences; Associate Professor of Department of System and Technologies Modeling, V. N. Karazin Kharkiv National University, Svobody Sq 4, Kharkiv- 22, Ukraine, 61022;e-mail: imartinkus@gmail.com; ORCID: https://orcid.org/0000-0003-4629-6583.*

Tkachuk Mykola V. *– PhD, Doctor of Science, Professor; Head of Department of System and Technologies Modeling, V. N. Karazin Kharkiv National University, Svobody Sq 4, Kharkiv - 22, Ukraine, 61022;e-mail: tka.mobile@gmail.com; ORCID: https://orcid.org/0000-0003-0852-1081.*

Нагорний Костянтин Анатолійович *– кандидат технічних наук, доцент; Харківський національний університет імені В.Н. Каразіна, м. Харків, пл. Свободи 4, 61022; e-mail: k.nagornyi@gmail.com ORCID :0000000159483682.*

Мартінкус Ірина Олегівна *– кандидат технічних наук; доцент кафедри моделювання систем і технологій, Харківський національний університет імені В.Н. Каразіна, майдан Свободи, 4, Харків-22, Україна, 61022; e-mail: imartinkus@gmail.com; ORCID: https://orcid.org/0000-0003-4629-6583.*

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: 0000000308521081.*

Нагорный Константин Анатольевич *– кандидат технических наук; доцент кафедры моделирования систем и технологий, Харьковский национальный университет имени В.Н. Каразина, площадь Свободы, 4, Харьков-22, Украина. 61022; e-mail: k.nagornyi@gmail.com; ORCID: https://orcid.org/0000-0001-5948-3682*

Мартинкус Ирина Олеговна *– кандидат технических наук; доцент кафедры моделирования систем и технологий, Харьковский национальный университет имени В.Н. Каразина, площадь Свободы, 4, Харьков-22, Украина. 61022; e-mail: imartinkus@gmail.com; ORCID: https://orcid.org/0000-0003-4629-6583.*

Ткачук Николай Вячелавович *– доктор технических наук, профессор; Харьковский национальный университет имени В.Н. Каразина, г.. Харьков, пл. Свободы 4, 61022; e-mail: tka.mobile@gmail.com; ORCID: 0000000308521081.*

Надійшла – 10.03.2019.