Proqram Kodunun Yazılması Zamanı Yaranan Səhvlərin Təhlili

Tamilla Bayramova[1], Kərim Elləzov[2]
[1,2]AMEA İnformasiya Texnologiyaları İnstitutu, Bakı, Azərbaycan
[1]*tamilla@iit.ab.az*, [2]*kerim.ellezov@gmail.com*

***Xülasə*— İşdə proqram kodunun yazılması zamanı yaranan səhvlər və onların mənbəyi araşdırılmış, kodun dinamik və statik analizinin üstünlükləri göstərilmişdir. Kodun statik analizinin əhəmiyyəti, proqram təminatının hazırlanması prosesinə və proqram kodunun keyfiyyətinə təsiri araşdırılmışdır.**

Açar sözlər— proqram kodu, proqramın keyfiyyəti, proqram xətaları, kodların analizi, statik analiz

I. GİRİŞ

Mürəkkəb proqram sistemlərində proqram kodunun çox olması onda səhvlərin sayının artmasına səbəb olur. Proqram kodunun yazılması zamanı səhvlərin yaranması qaçılmazdır. Belə səhvlərin axtarılması və düzəldilməsi çox zəhmət və vaxt tələb edən bir iş olduğundan ona çəkilən xərclər proqram təminatının (PT) işlənilməsinə çəkilən xərclərin 45%-ni təşkil edir [1].

PT-də olan səhvlərin vaxtında aşkarlanması və aradan qaldırılması üçün proqramın işlənilməsi prosesi sona çatanda onun alfa-versiyası hazırlanaraq əvvəlcədən müəyyən olunmuş testerlər tərəfindən sınaqdan keçirilir. Adətən bu versiyanın imkanları məhdud olur və özündə ancaq əsas proqram modullarını birləşdirir. Mütəxəssislər testerlərin sınaq nəticələrini təhlil etdikdən və səhvlər düzəldikdən sonra PT-nin beta-versiyası hazırlanır. Bu PT-nin tam imkanlı versiyası olur və yenidən sınaq üçün mütəxəssislərə göndərilir. Beta versiyanın sınağından və səhvlərin düzəlişindən sonra PT istifadəyə verilir. Lakin bu o demək deyil ki, proqramda heç bir səhv olmayacaqdır. Səhvlərin axtarılması və düzəldilməsi fasiləsiz proses olub, proqramın müşayiət edilməsi sona çatanadək davam edir. Çünki səhvləri düzəldərkən PT-də yeni səhvlər meydana çıxa bilər.

Proqramda olan səhvlər kritik cəhətdən vacib olan sistemlərin (hərbi komplekslər, tibbi avadanlıqlar, mühəndis infrastrukturu və s.) işinə təsir edir və bəzi hallarda səhvlər faciə ilə nəticələnə bilir.

Kompüter proqramlaşdırması müxtəlif proqramların yaradılması üçün istifadə edilən yüzlərlə proqramlaşdırma dilindən ibarət böyük bir sahədir. Buraya əməliyyat sisteminin, tətbiqi proqramların, mobil platformalar üçün proqram kodunun yazılması, veb-proqramlaşdırma və s. daxildir.

Proqram kodunun yazılması zamanı ən çox rast gəlinən səhvlər aşağıdakılardır:

- ***məntiqi səhvlər.*** Bu səhvlərin ən ciddisidir. İstənilən bir proqramlaşdırma dilində yazılmış proqram düzgün kompilyasiya edilir və işləyirsə, lakin çıxışda səhv nəticə verirsə bu proqramlaşdırmanın məntiqi cəhətdən səhv yerinə yetirildiyini göstərir. Bunun səbəbi isə baza alqoritminin səhv olmasıdır. Bu xətanı aradan qaldırmaq üçün səhvi alqoritmdə axtarmaq lazımdır, bəzən alqoritm təmamilə yenidən işlənilməlidir və proqram kodu yenidən yazılmalıdır.

- ***sintaksis səhvləri.*** C, Java, Perl və Python kimi bütün proqramlaşdırma dillərinin proqram kodunun yazılması üçün öz sintaksisi var. Proqramçı kodu düzgün yazmadıqda sintaksis səhvi yaranır. Belə səhvlər kodun yazılması zamanı İnteqrallaşmış Proqramlaşdırma Mühitinin köməyi ilə asanlıqla aradan qaldırılır.

- ***kompilyasiya səhvləri***. Kompilyasiya zamanı yuxarı səviyyəli dillərdə yazılmış proqram kodu maşın dilinə çevrilir. Bu mərhələdə sintaksis səhvi də daxil olmaqla müxtəlif səhvlər yarana bilər. Bəzən sintaksis tam olaraq düzgün olur, lakin yenə də səhv yaranır. Bu, kompilyatorun özündə olan xətalarla da əlaqədar ola bilər. Bu səhvlər proqramın işlənilmə prosesində aradan qaldırılır.

- ***icra edilmə mühitinin səhvləri*** (*RunTime*). Proqram kodu müvəffəqiyyətlə kompilyasiya olunduqdan sonra icra olunan fayl yaradılır. Proqramı yoxlamaq üçün onu real mühitdə icra edirlər. Bu zaman resursların kifayət etməməsi və ya hər hansı bir qəza nəticəsində yenə proqram öz işinin öhdəsindən gəlməyə bilər. Proqramçı proqramın işləyəcəyi real mühitin şərtlərini əvvəlcədən nəzərə almalıdır. Bu səhvi yenidən kodlaşma mərhələsinə qayıtmaqla aradan qaldırmaq olar.

- ***riyazi səhvlər.*** Proqramların əksəriyyəti ədədi verilənlərdən istifadə edir və alqoritmdə riyazi hesablamalar ola bilər. Riyazi səhvlər yerinə yetirilməsi mümkün olmayan hesablamalar (məsələn, sıfra bölmə, nəticənin sonsuzluq alınması və s.) nəticəsində yaranır. Bu da məntiqi səhvdir və alqoritmdə dəyişiklik etməklə aradan qaldırılır.

- ***resurs səhvləri.*** Bu səhvlər dəyişənin ala biləcəyi mümkün maksimal qiyməti aşdıqda, proqram işini bitirdikdən sonra yaratdığı resursları buferdən silmədikdə, müəyyən kod blokuna düzgün olmayan şəkildə rekursiv müraciət etdikdə və digər hallarda yaranır.

- ***qarşılıqlı əlaqə zamanı yaranan səhvlər.*** Bu səhvlər proqram təminatının aparat interfeysi və ya tətbiqi proqramlarla uyğunsuzluğu zamanı yaranır. Veb proqramlaşdırmada isə bu səhvlər veb-protokoldan düzgün istifadə edilməməsi zamanı yarana bilər.

DOI: 10.25045/NCSoftEng.2017.85

Ardıcıl sınaqlar və sazlama mərhələləri PT-nin işlənilməsinin ayrılmaz hissələri olub səhvlərin vaxtında aşkar edilməsini təmin edir. PT-nin işlənilmə prosesinin düzgün planlaşdırılması da səhvlərin azalmasına gətirir. Proqram kodunda olan səhvlər qaçılmazdır və onların aradan qaldırılması üçün yeni metodlar və üsullar işlənilir.

Proqram kodunda olan xətaların aşkar edilməsi və düzəldilməsi məqsədilə bir neçə statik metoddan istifadə olunur. Bu metodlar ayrı-ayrılıqda proqram kodundakı bütün səhvləri aşkarlamağa imkan vermir. Buna görə də onlardan bir-birinin tamamlayıcısı kimi istifadə edirlər. Proqram kodunun işlənilməsi zamanı xətaların aşkarlanması üçün əsas metodlara kodun statik və dinamik analizi metodları daxildir.

II. KODUN STATİK ANALİZİ

Kodun statik analizi (KSA) proqram kodlarını icra etmədən, onların yazılması zamanı analiz edilməsini və yarana biləcək xətaların aşkar edilməsini təmin edir.

Hal-hazırda proqram təminatının getdikcə mürəkkəbləşməsi və onların istismara çıxarılma müddətinin qısaldılması proqram kodlarında səhvlərin mümkün qədər tez və nisbətən az maddi vəsaitlə effektiv şəkildə aradan qaldırılmasını tələb edir [2, 3]. KSA proqramın icrasını və müəyyən bir testləşdirmə parametrlərinin təyin edilməsini tələb etmədiyi üçün kodun yazılması prosesinin hər pilləsində asanlıqla tətbiq edilə bilər. KSA avtomatlaşdırılmış alətlərin köməyi ilə yerinə yetirilir. Avtomatlaşdırılmış KSA ilə proqram kodlarındakı səhvlər tez və az maddi vəsait xərcləməklə aşkar edilir və funksional testlərin daha effektiv şəkildə yerinə yetirilməsinə imkan verir. Nəticədə daha keyfiyyətli və təhlükəsiz proqram təminatının yaradılması mümkün olur.

KSA vasitəsilə təhlil edilməli bir sıra səhvlər vardır və avtomatlaşdırılmış alətlər bu səhvləri təhlil edərək hesabatlar təqdim edir. Açıq kodlu və ya kommersiya məqsədi ilə hazırlanmış KSA vasitəsilə təhlil edilərək vaxtında aradan qaldırıla bilən səhvlərə yuxarıda adlarını çəkdiyimiz səhvlər və xətalarla yanaşı aşağıda göstərilənlər də aiddir:

- ***təhlükəsizlik boşluqları.*** KSA-da əsas məqsəd proqram kodundakı xəta və təhlükəsizlik boşluqlarını müəyyən edərək aradan qaldırmaqdır. KSA alətləri proqram kodlarını təhlil edərək müxtəlif tipli xətaları aşkar edir. Bu xətalardan ən çox qarşıya çıxanlara buferin dolub daşması, yaddaş itkiləri, ilkin qiyməti verilməmiş dəyişənlər, kodların yazılması zamanı istifadə olunan kitabxanaların və ya onların mənbələrinin uyğunsuzluğu, tiplərin uyğunsuzluğu, istifadə olunmayan ölü kodlar, sıfıra bölmə halı yarada biləcək proqramlaşdırma xətaları və s. misal göstərmək olar. Bu xətalar proqramda hər hansı bir şərt daxilində xidmətin dayanması, verilənlərə icazəsiz əlyetərlilik əldə edilərək dəyişdirilməsi və ya silinməsi ilə nəticələndikdə təhlükəsizlik boşluğu hesab edilir [4]. Proqram sistemlərində baş verən fəlakətlərin bir çoxu KSA-nın düzgün aparılmaması nəticəsində baş vermişdir. Bu da xətaların müəyyən edilməsində KSA-nın nə dərəcədə əhəmiyyətli olduğunu bir daha sübut edir [5].
- ***kodlaşdırma standardları.*** Proqram sistemlərinin daha keyfiyyətli və təhlükəsiz hazırlanması üçün bir çox kodlaşdırma standardları işlənilmişdir [6]. Eyni zamanda proqram kodlarının fərqli proqramçılar tərəfindən oxunaqlı və anlaşıqlı olması üçün bir sıra kodlaşdırma formaları hazırlanmışdır. Bu forma və standardlara uyğun hazırlanan proqramlar modullar və ya daha kiçik hissələrə ayrılmış kod blokları şəklində olur. Bu da hər hansı bir modulu rahatlıqla dəyişməyə, yeni və daha təhlükəsiz variantı ilə əvəz etməyə imkan verir. Bu cür yazılmış proqram kodları KSA alətləri tərəfindən daha səmərəli və effektiv şəkildə təhlil edilir.
- ***arxitekturun analizi.*** Burada arxitektur dedikdə, əməliyyat sisteminin arxitekturu, kompyuterin daxili arxitekturu, proqram təminatının daxili arxitekturu və onun istifadə olunacağı avadanlıqların arxitekturu və s. nəzərdə tutulur. Bəzi KSA alətləri, təhlil etdikləri proqram sistemlərini arxitektur baxımından qiymətləndirə bilir. Hazırlanan proqramın məlum arxitektura uyğun olub olmamasını, yarana biləcək konfliktlər və s. bu alətlər vasitəsilə təyin etmək mümkündür. Bu alətlər həm də proqram kodunun işlənilməsi zamanı arxitekturu başa düşərək proqramı inkişaf etdirmək, lazımsız və zərərli asılılıqları və əlaqələri aradan qaldırmaq məqsədi ilə də istifadə olunur.
- ***əks mühəndislik*** *(reverse engineering).* Bəzi hallarda əvvəlcədən yazılmış proqram kodları müəyyən səbəblərdən əlçatmaz olur. (Misal üçün, kodların oğurlanaraq silinməsi, kodların korlanması və s.) Bu halda proqram kodlarının proqramçılar tərəfindən işlənilməsi mümkün olmur. Tərsinə mühəndislik prosesində bir qrup proqramçı maşın kodunu analiz edir və onun əsasında proqramın alqoritmi və psevdokodu bərpa edilir [7]. Bəzi proqramlar müəllif hüququ və patentlə qorunduğu üçün tərsinə mühəndisliyin icazəsiz şəkildə tətbiq olunması qadağan edilir. KSA alətlərinə daxil olan tərsinə mühəndislik modulu, proqramı təhlil edərək kodların yazılış forması, siniflər arası əlaqə, interfeys və iş prinsipi kimi xüsusiyyətləri əyani şəkildə göstərir və proqramçı tərəfindən yaxşı başa düşülməsinə zəmin yaradır [8, 9].

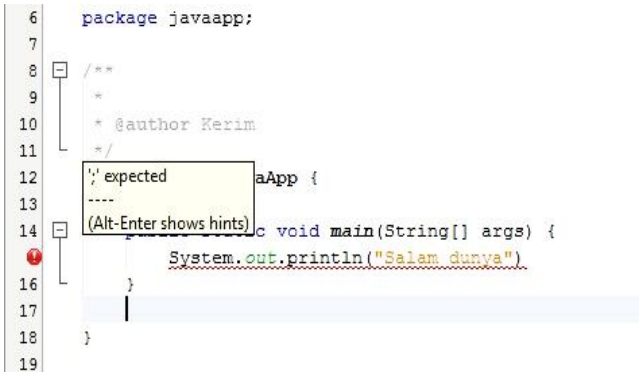
A. Avtomatlaşdırılmış KSA alətlərinin tətbiqi

İnteqrallaşmış Proqramlaşdırma Mühiti (*İntegrated Development Environment - IDE*) proqramçılar üçün hazırlanmış, bir sıra KSA alətlərini özündə birləşdirən çoxfunksiyalı proqramlaşdırma alətidir. Bu mühit proqramçıların kod yazarkən işini asanlaşdırmaq üçün proqram kodlarını müxtəlif rənglərlə ayırmaq, diaqnostika aparmaq, proqram kodunu maşın dilinə çevirmək kimi funksiyaları özündə birləşdirir.

Sintaksis səhvlər proqram kodunu yazan zaman meydana gəlir və IDE-nin mətn prosessoru tərəfindən tapılaraq işarə olunur. Məlumdur ki, hər bir proqramlaşdırma dilinin öz ciddi sintaksisi vardır. Kodları hazırlayan zaman əlavə işarələr yazdıqda və ya dilin sintaksisinə riayət etmədikdə sintaksis xətalar meydana çıxır.

Şəkil 1-də NetBeans mühitində Java proqramlaşdırma dilində yazılmış sadə kod fraqmenti göstərilib. Koda baxdıqda asanlıqla 15 – ci sətirin altından dalğalı xətt çəkildiyini görə bilərik. Burada mühit, sintaksis səhvi aşkarlayaraq proqramçıya dilin sintaksisinə görə sətir sonunda “;” işarəsini unutduğunu göstərir.

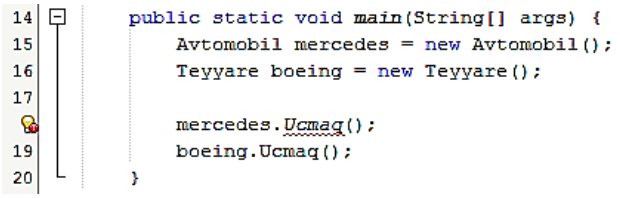
DOI: 10.25045/NCSoftEng.2017.85



Şəkil 1. Sintaksis səhvinin aşkar edildiyi kod fraqmenti

Kompliyasiya səhvləri də inteqrallaşmış mühit tərəfindən tapılaraq işarə olunur. Sintaksis səhvlərindən fərqi ondadır ki, müəyyən bir kod blokunu, müəyyən bir sinifin metodunu düzgün istifadə etmədikdə meydana çıxır.

Məsələn, şəkil 2-də verilmiş kod fraqmentində hər ikisi nəqliyyat interfeysindən törəyən, “Avtomobil” və “Teyyare” sinfinə daxil olan iki, “mercedes” və “boeing” obyekti yaradılıb. Burada “boeing” obyekti üçün “Uçmaq” metodu əvvəlcədən verildiyi halda (implementation), “mercedes” obyekti üçün bu metod verilməmişdir. Şəkildə aydın görünür ki, mühit, meydana çıxmış səhvi işarələyərək düzgün istifadə edilməyən kodu göstərir. Hazırda IDE – lərin əksəriyyəti KSA alətlərini özündə birləşdirir.



Şəkil 2.Kompilyasiya səhvinin aşkar edildiyi kod fraqmenti

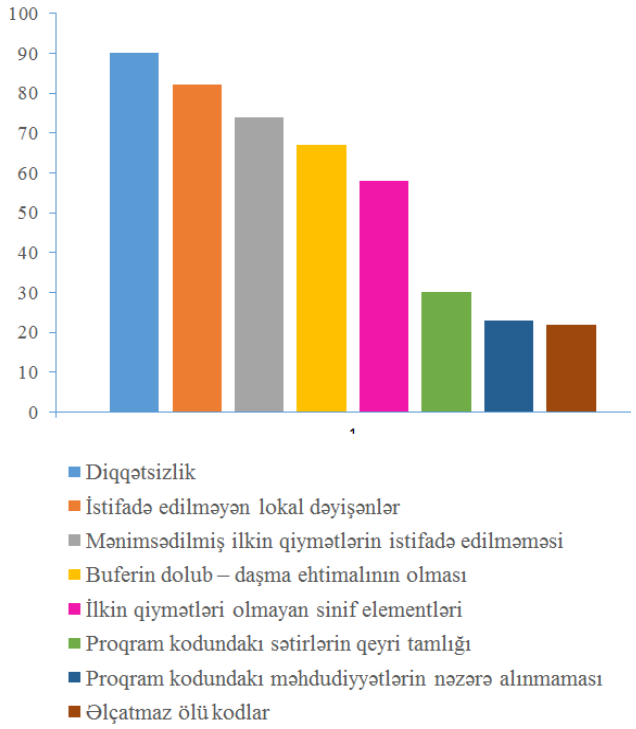
Müasir IDE-lər yuxarıda göstərilənlərlə yanaşı, kodun intellektual tamamlanması funksiyasını, Kodların Dinamik Analizi alətlərini də özündə birləşdirir (məsələn, NetBeans IDE, Visual Studio, IntelliJ IDEA və s.). Kodun intellektual tamamlanması proqramçının sintaksis və kompliyasiya səhvlərinə yol verməsinin qarşısını xeyli alır və kodların yazılmasına sərf olunan zamanı azaldır. Proqramçı dilin açar sözlərinin ilk hərflərini yazmağa başladıqda intellektual tamamlayıcı həmin açar sözlərə uyğun gələ biləcək kodlar təklif edir. Proqramçı təklif olunan bu kodlardan birini klaviaturadan istifadə edərək qısa yolla seçir. İntellektual tamamlayıcı müasir IDE-lərə daxil olan mətn prosessorlarının hər birində vardır. Misal üçün Visual Studio IDE mühitində İntelliSense kod tamamlayıcı modulunu göstərmək olar.

Mövcud KSA alətlərinin bir qismi proqramın kodlarını yazılı şəkildə təhlil edir və əlavə biliyə ehtiyac olmur. Daha funksional analiz edən alətlər isə istifadə edilən platforma və sazlayıcı məlumatlarına ehtiyac duyur. Proqram kodlarının sazlanma (debug) zamanı analiz edilməsi daha etibarlı və ərtaflı nəticələrin alınmasına gətirib çıxarır[9]. Bu alətlərə misal olaraq müasir IDE-ləri, Google CodePro Analytix, Findbugs, PMD, Checkstyle, Sonar və s. göstərmək olar.

B. KSA nəticələrinin qiymətləndirilməsi

Proqram kodlarını statik analiz etdikdən sonra əldə olunan nəticələrin necə qiymətləndirilməsi də çox önəmlidir. Aşkar olunmuş xətalar risk səviyyələrinə görə sistemləşdirilir və bu səviyyəyə uyğun tədbirlər görülərək aradan qaldırılır. Bəzi xətaların risk səviyyəsi aşağı olduğu üçün onların maddi və ya əhəmiyyətlilik analizi aparıldıqdan sonra düzəliş edilib edilməməsinə qərar verilir. KSA zamanı məhdud sayda KSA alətlərindən istifadə edildiyindən nəticələr ümumiləşdirərək qiymətləndirilir. Bu alətlərdən istifadə etdikdə orta hesabla 17% maddi vəsaitə qənaət etmək mümkündür. Ümumiləşmiş yekun nəticə, alətlərin hər birinin nəticəsinin orta qiyməti kimi götürülür.

Aparılan araşdırmalara görə real layihələrə tətbiq olunan KSA alətlərinin ən çox müəyyən etdiyi xətalar şəkil 3-də göstərilmişdir.



Şəkil 3. KSA alətlərinin ən çox müəyyən etdiyi xətaların təsvir edildiyi diaqram

III. KODUN DİNAMİK ANALİZİ

Proqramın icrası zamanı yaranan səhvlərə bəzən *baqlar* da deyilir. Onların yaranmasına bir çox amillər səbəb olur. Məsələn, sistemin qeyri-stabil olması, virusa yoluxma, proqrama gözlənilməyən kənar müdaxilə edilməsi, proqramın hər hansı bir modulunun işinin dayanması və s. belə xətaların meydana gəlməsinə səbəb ola bilər. Bu cür xətalar böyük fəlakətlərə səbəb ola bilər. Buna görə də onların aradan qaldırılması statik analizə nisbətən mürəkkəb bir prosesdir. Bu xətalar real sistemlərdə və ya virtual maşınlarda kodların dinamik analizi (KDA) alətlərindən istifadə etməklə aradan qaldırılır.

DOI: 10.25045/NCSoftEng.2017.85

KDA metodu vasitəsilə verilən proqramlar avtomatik generasiya edilən giriş verilənləri ilə bir neçə dəfə icra edilir, xətaların yaranması zamanı qeydə alınır. Bu metodlar adətən kiçik proqram kodlarına (adətən bir neçə min sətir) tətbiq edilir.

İşləyən proqramın təhlükəsizlik texnologiyaları daha aktualdır, çünki heç bir analiz sistemi PT-də olan səhvlərin tamamilə aşkarlanmasına təminat verə bilməz. Bu təhlükəsizlik metodları hücumların və hətta proqramda aşkarlanan zəif nöqtələrin istifadə edilməsinin qarşısının alınması üçün əsas vasitədir.

NƏTİCƏ

Proqram kodunun yazılmasının ilkin mərhələlərində avtomatlaşdırılmış KSA alətləri vasitəsilə səhvlərin əksəriyyəti aşkar edilərək aradan qaldırıldığına görə proqramın dinamik analizinə sərf olunan zaman müddəti xeyli azalmış olur. Səhvlərin vaxtında aşkarlanması proqram təmiatının işlənilməsinə çəkilən xərcləri də azaldır. Sınaq mərhələsinə keçməzdən əvvəl tətbiq ediləcək KSA alətləri müəyyənləşdirilir. Avtomatlaşdırılmış KSA alətlərinin bir neçəsini tətbiq edərək analiz aparılması effektli nəticələrin əldə olunmasına gətirir.

Bütün bunlara baxmayaraq KSA alətlərinin istifadəsində bəzi məhdudiyyətlər mövcuddur. Elm və texnikanın sürətli inkişafı yeni metod və alətlərin işlənilməyinə ümid verir. Zamanla bu məhdudiyyətlərin aradan qaldırılacağı və KSA alətlərinin proqramçılar arasında daha geniş yayılacağı ehtimal edilir.

ƏDƏBİYYAT

[1] «Виды ошибок в программировании» http://juice-health.ru/programming/102-vidy-oshibok

[2] R. E. Park, W. B. Goethert, and W. A. Florac, , Goal- Driven Software Measurement – A Guidebook, CMU/SEI- 96-HB-002, 1996

[3] G. D. Everett, and R. McLeod, Software Testing: Testing Across the Entire Software Development Life Cycle, IEEEPres, 2007

[4] D. Baca, B. Carlsson, and L. Lundberg, “Evaluating the Cost Reduction of Static Code Analysis for Software Security”, Proceedings of the third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS’08), 2008

[5] J. Ganssle, “Learning From Disaster”, Military and Aerospace Programmable Logic Devices Conference (MAPLD 2005), 2005

[6] J.W. Moore, and R.C. Seacord, “Secure Coding Standards”, CrossTalk, 2007

[7] R. Robbes, R. Oliveto, M.Di Penta, Guest editorial: special section on software reverse engineering // Empir Software Eng, pp.749–752, 2016

[8] Q. Wang, N. Meng, Z. Zhou, J. Li and H. Mei, “Towards SOA-based Code Defect Analysis”, IEEE International Symposium on Service-Oriented System Engineering, (SOSE’08), 2008

[9] K.A. Olson, and C.M. Overstreet,“Enhancing Model Understanding Through Static Analysis”, Virginia Space Grant Consortium Student Research Conference, 2006

[10] P. Anderson, “The Use and Limitations of Static- Analysis Tools to Improve Software Quality”, CrossTalk, 2008

DOI: 10.25045/NCSoftEng.2017.85