УДК 004.415.5

О.В. ПОМОРОВА, Д.О. ІВАНЧИШИН
Хмельницький національний університет

ПОРІВНЯЛЬНИЙ АНАЛІЗ РЕЗУЛЬТАТІВ ФУНКЦІОНУВАННЯ СТАТИЧНИХ АНАЛІЗАТОРІВ С++ ВИХІДНОГО КОДУ

Проведено аналіз результатів функціонування найбільш розповсюджених засобів для статичного аналізу С++ вихідного коду. Досліджено особливості підрахунку та класифікації дефектів у програмному забезпечені різними засобами статичного аналізу. Встановлено відсутність стандартизації на оцінку результатів функціонування СА.

The analysis of the results of most common tools for C++ source code static analysis was investigated. The features of defects counting and classification in the software by various means of static analysis were researched. The Absence of the SA results standardization was found.

Ключові слова: статичний аналіз, тестування, якість ПЗ, С++.

Вступ

Однією із актуальних проблем у галузі інженерії програмного забезпечення (ПЗ) на сьогодні є підвищення якості ПЗ [1]. Причинами проблеми є зростаюча складність програмних систем, застосування ПЗ при розробці та експлуатації критичних систем, скороченням часу на розробку нових версій ПЗ і т.і. Одним із типів помилок в ПЗ є нефункційні помилки – такі, що пов'язані з порушеннями правил мови програмування, некоректним використанням бібліотечних функцій, системних викликів.

Значна частина системного ПЗ комп'ютерних систем, ПЗ для мобільних і вбудованих систем, систем управління та інших критично важливих додатків розроблені з використанням мови С++. За даними компанії Coverity в ПЗ, написаному на мові С++, міститься 0.25 нефункційних помилок на 1000 рядків вихідного коду [2]. Нефункційні помилки, характерні для послідовних програм, прийнято називати програмними дефектами. Прояв програмних дефектів призводить до невірних результатів, зависання або аварійного завершення програм.

Виявлення і виправлення програмних дефектів (налагодження програми) є одним з найбільш складних і трудомістких етапів у процесі розробки ПЗ [3]. За оцінками, на виявлення дефекту витрачається до 95% часу налагодження, на його виправлення – тільки 5%. Отже, задача автоматизації процесу виявлення програмних дефектів є актуальною, а її вирішення забезпечить підвищення ефективності процесу розробки ПЗ.

Сучасні методи автоматизованого виявлення програмних дефектів розділяють на наступні класи [4]:

- динамічні методи – використовують результати виконання програми для виявлення дефектів;
- статичні методи – використовують вихідний код, моделі, специфікації та інші артефакти, створювані в процесі проектування ПЗ.

Застосування методів статичного аналізу (СА) дозволяє виявляти основні типи програмних дефектів, при цьому забезпечується повна автоматизації процесу їх виявлення.

На сьогодні ринок розробок СА для С++ програм представлений рядом засобів, що мають різну функціональність та вартість. Відмінність алгоритмів функціонування статичних аналізаторів коду призводить до значної різниці та диференціації результатів їх роботи, що створює потребу у дослідженні ефективності та доцільності використання подібних додатків для ПЗ різного призначення.

Постановка задачі

Для підвищення ефективності застосування статичних аналізаторів С++ вихідного коду необхідно провести порівняльний аналіз результатів функціонування статичних аналізаторів для різного типу ПЗ та встановити залежність між кількістю повідомлень про дефекти, котрі генеруються СА, та кількістю реально виявлених дефектів.

Статичний аналіз вихідного коду

Статичний аналіз (СА) вихідного коду [5] є одним із методів оцінки якості програмного забезпечення, що застосовується на ранніх етапах розробки і не вимагає повної завершеності циклу розробки. СА є автоматизованим процесом огляду коду. Огляд коду (code review) – один з найстаріших методів виявлення дефектів у ПЗ. Він полягає у спільному читанні вихідного коду командою розробників і наданню рекомендацій щодо його поліпшення. В процесі читання коду у ньому виявляються помилки або частини, які можуть стати помилковими в майбутньому. Автор коду під час огляду не повинен давати пояснень як працює та або інша частина програми. Алгоритм функціювання має бути зрозумілий безпосередньо з тексту програми і коментарів. Якщо ця умова не виконується, то код потрібно доопрацювати. Як правило, огляд коду дає позитивні результати і дозволяє виявити значну кількість помилок у ПЗ.

Істотним недоліком методології спільного огляду коду є висока ціна. Необхідно збирати команду з декількох програмістів для огляду нового або повторно написаного коду. Програмісти мають регулярно робити перерви для відпочинку, оскільки, перегляд відразу великих фрагментів коду призводить до зниження уваги та падіння ефективності огляду коду.

Компромісним рішенням є розроблення інструментів статичного аналізу, що в автоматичному

режимі проводять ревізію вихідного коду з метою виявлення дефектів. Такий підхід суттєво знижує вартість виявлення помилок і є рушійною силою до швидкого розвитку ринку подібних додатків та їх застосування компаніями, що розробляють ПЗ.

Завдання, що вирішуються програмами статичного аналізу коду ділять на три категорії :

- виявлення помилок у вихідному коді програм;
- рекомендації щодо оформлення вихідного коду. Деякі статичні аналізатори дозволяють перевіряти, чи відповідає вихідний код стандартам оформлення, наприклад MISRA C++ 2008, Sutter - Alexandrescu Rules, Meyers - Klaus Rules та ін.;
- підрахунок метрик. Метрика програмного забезпечення – це міра, що дозволяє отримати числове значення деякої властивості програмного забезпечення або його специфікацій.

Також статичний аналіз використовується як метод контролю і навчання нових співробітників, котрі недостатньо знайомі з правилами програмування.

На сьогодні не існує ідеального методу тестування програмного забезпечення. Для різних типів програмного забезпечення різні методики дають різні результати. Поєднання різних методик дозволяє підвищити якість ПЗ.

Головною перевагою статичного аналізу є істотне зниження вартості усунення дефектів у ПЗ. Чим раніше виявлена помилка, тим менша вартість її виправлення. Так, виправлення помилки на етапі тестування коштує вдесятеро дорожче, ніж на етапі написання коду (рисунок 1) [6] :

Етап на якому внесено дефекту	Етап на якому виявлено дефект				
	Розробка вимог	Проектування архітектури	Побудова (кодування)	Тестування	Після випуску ПЗ
Розробка вимог	1	3	5-10	10	10-100
Проектування архітектури	-	1	10	15	25-100
Побудова (кодування)	-	-	1	10	10-25

Рис. 1. Залежність середньої вартості виправлення дефекту від етапу, на якому він виявлений

Інші переваги статичного аналізу коду :

- повне покриття коду. Перевірка фрагментів коду, що рідко отримують управління. Такі фрагменти коду, зазвичай, не вдається протестувати іншими методами;
- статичний аналіз не залежить від використовуваного компілятора і середовища, в якому виконуватиметься скомпільована програма. Це дозволяє знаходити приховані помилки, які можуть з'являтись при зміні версії компілятора або при використанні інших ключів для оптимізації коду. Наприклад, помилки невизначеної поведінки;
- виявлення помилок набору і однакових конструкцій.

Недоліки статичного аналізу коду:

- статичний аналіз не забезпечує достатньо високих результатів у діагностиці витоків пам'яті (memory leak) і паралельних помилок. Щоб виявляти подібні помилки, необхідно віртуально виконати частину програми, що вимагає складної реалізації і значних апаратних ресурсів. Як правило, статичні аналізатори обмежуються діагностикою простих випадків. Ефективнішим способом виявлення витоків пам'яті і паралельних помилок є використання інструментів динамічного аналізу;
- засоби статичного аналізу генерують попередження про підозрілі місця вихідного коду, які насправді можуть бути цілком коректні. Це називається хибними спрацьовуваннями. Зрозуміти, вказує аналізатор на реальний дефект коду чи згенерував хибне спрацьовування, може тільки програміст. Необхідність опрацьовувати хибні спрацьовування потребує значного часу та послаблює увагу до тих фрагментів коду, де насправді містяться помилки.

Напрям статичного аналізу коду активно розвивається, з'являються нові діагностичні правила і стандарти, деякі правила застарівають, тому вибір засобів СА коду не може ґрунтуватись на базі дефектів, котрі вони можуть виявити. Єдиним способом порівняння результатів функціювання інструментів СА є аналіз коду програмних проектів і підрахунок кількості віднайдених у них помилок.

Вибір засобів статичного аналізу та програмного забезпечення, що тестується

Проведемо дослідження функціювання СА для різних типів ПЗ. Найбільш поширеними та розвинутими СА засобами на сьогодні є наступні продукти: Gimpel Software FlexeLint(PC-Lint), PVS-Studio, Red Lizard Software Goanna Studio, Parasoft C++ Test, CppCheck, Klocwork Insight, Coverity Static Analysis,

Programming Researc QA·C++ Source Code Analyzer, CppCheck.

При виборі програмних додатків для дослідження був вироблений наступний список вимог:

- можливість проведення автоматичного аналізу вихідного коду без його попередньої спеціальної підготовки;
- відслідковування впливу різних функцій аналізатора на його поведінку при пошуку заданих ситуацій;
- відсутність обмежень на розмір вихідного коду;
- наявність актуальної бази дефектів ПЗ.

Для порівняння зі спеціалізованими засобами було обрано вбудований в IDE Microsoft Visual Studio 2010 статичний аналізатор вихідного коду. Додатковим критерієм для вибору СА став принцип розповсюдження – freeware. Оскільки більшість засобів статичного аналізу мають достатньо високу вартість, доцільним є порівняння та оцінка ефективності їх безкоштовних аналогів.

У результаті врахування вищевказаних вимог для дослідження були обрані такі СА:

- Microsoft Visual Studio 2010 Static Analyzer – вбудований в IDE засіб статичного аналізу;
- Gimpel PC-Lint в комплексі з Visual Lint для інтеграції з IDE. Додаток проводить семантичний аналіз вихідного коду, аналіз потоків даних і управління. Ціна – $389;
- PVS-Studio – статичний аналізатор, з можливістю діагностування 64-бітових помилок (Viva64), діагностування паралельних помилок (VivaMP) та діагностування загального призначення. Використовувався в інтеграції з IDE. Ціна - €3500;
- Red Lizard Goanna Studio - інструмент статичного аналізу для виявлення помилок, вразливостей і загальних недоліків вихідного коду, наприклад: переповнення буфера, витоки пам'яті та ін. Використовувався в інтеграції з IDE. Ціна – від $999 залежно від компонування;
- CppCheck. На відміну від C/C++ компіляторів та інших інструментів аналізу не виявляє синтаксичні помилки в коді. Cppcheck в першу чергу визначає типи помилок, котрі не виявляють компілятори. Ціна – Freeware.

Для порівняння результатів функціонування обраних СА був сформований перелік вимог до тестованого ПЗ:

- завершені продукти, компіляція та виконання яких відбувається без помилок;
- відсутність попереджень, помилок та повідомлень, пов’язаних з якістю вихідного коду зі сторони середовища розробки. В досліджені використовувався вбудований засіб статичного аналізу IDE Microsoft Visual Studio 2010;
- відсутність використання застарілих бібліотек;
- різні типи ПЗ: системне та прикладне;
- відкритий вихідний код.

В результаті було обрано наступні програмні засоби:

системне ПЗ:

- утиліти: 7Z, Crystal DiskMark 3.0.1, Crystal DiskInfo 4.3.0, Ac3filter 1.63b, Wintarball 1.2, FirewallPApi 1.4, Antispy 3.2.2;
- мережеві менеджери: NetBIOS Enumerator 1.017, Remote Control Center 4.03;
- віконні менеджери: BlackBox 0.82, Hide That Windows 0.4;
- менеджери задач: TaskSwitchXP 2.0.11;
- файлові менеджери: ultraMaGe 0.7.2;

прикладне ПЗ:

- загального призначення: Asteroid, SolarSystem, Folder Size 2.0;
- спеціалізованого призначення: Notepad++ 6.0, emule 0.50a, Loganalyzer, Shareaza 2.6.0, Yafe 0.9.8.

Результати статичного аналізу

На сьогодні відсутня стандартизація, що чітко визначає критерії оцінки важливості помилок у ПЗ. Більшість розробників додатків для СА створюють власні системи оцінювання віднайдених дефектів. Кожен з представлених у досліджені додатків СА містить власну класифікацію помилок, що віднаходяться у ПЗ. Microsoft Visual Studio 2010 розподіляє повідомлення на дві категорії: warnings та informational. Gimpel PC-Lint усі повідомлення розподіляє на шість категорій: fatal errors, internal errors, errors, warnings, informational та elective notes. CppCheck також містить шість категорій повідомлень, але позначені вони як errors, warnings, style, performance, portability і informational. Goanna Studio є додатком до Microsoft Visual Studio 2010 і повідомлення про віднайдені недоліки виводяться у системну консоль даного IDE. Усі вони відносяться до категорії warnings. PVS-Studio містить три рівні важливості повідомлень, котрі на відміну від інших СА позначені: 1 lvl, 2 lvl та 3 lvl.

Для співставлення результатів функціонування різних засобів СА усі повідомлення були згруповані у три категорії: помилки, попередження та інші. Для PVS-Studio повідомлення категорії 1 lvl були віднесені до помилок, 2 lvl – до попереджень, 3 lvl – до інших. Для PC-Lint повідомлення класу informational і elective notes були віднесені до типу інші. Для CppCheck до інших були віднесені повідомлення з класу performance, style та portability. У таблиці 2 наведено результати аналізу СА обраного ПЗ. У дужках поряд з назвою проекту вказано кількість стрічок його вихідного коду. Усі СА використовувались з настроюваннями за замовчуванням.

З таблиці 1 слідує, що кількість повідомлень про віднайдені помилки в залежності від СА відрізнялась на декілька порядків. Так для проекту Emule, що є мережевим файлообмінним клієнтом і нараховує близько ста сорока тисяч стрічок вихідного коду, СА PC-Lint згенерував звіт, що налічував 366 512 повідомлень. Goanna Studio для цього ж проекту надав лише 554 повідомлення про помилки. Отже, СА PC-Lint для програмного продукту, що не містить помилок компіляції, згенерував звіт, що налічує у три рази більше включень ніж кількість стрічок вихідного коду ПЗ, що тестувалось. Це вказує на те, що числові показники результатів функціонування СА лише частково характеризують якість проведеного аналізу. Детального дослідження потребують якісні характеристики отриманих результатів.

Важливим критерієм порівняння результатів функціонування СА є кількість унікальних віднайдених помилок. Рисунок 2 відображає загальну кількість помилок віднайдену при аналізі ПЗ Firewall App, частину помилок, що одночасно містились у результатах функціонування інших аналізаторів та частину помилок, що були віднайдені лише певними аналізаторами. Звіт PC-Lint містив як найбільшу загальну кількість так і найбільший відсоток повідомлень, що були також включені у звіти інших засобів СА. З усіх 1119 виявлених дефектів 31 містився у звіті PVS-Studio, 26 – CppCheck, 12 – Visual Studio 2012. В загальному подібна картина розподілу характерна і для іншого протестованого ПЗ. Хоча СА PC-Lint надавав найбільшу кількість повідомлень, значний їх відсоток містив інформацію про певні стилістичні та синтаксичні проблеми, пов'язані з оформленням вихідного коду. І саме такий тип повідомлень позначався даним аналізатором, як помилки. Подібні повідомлення надавались навіть стосовно стандартних бібліотек IDE Microsoft Visual Studio 2010, що пояснює загальну кількість повідомлень для певних проектів, котра перевищує кількість стрічок вихідного коду у них. Повідомлення, що класифікувались як попередження, на відміну від помилок, містили інформацію про перевірку логічних конструкцій, виходи за межі типів, масивів та інше.

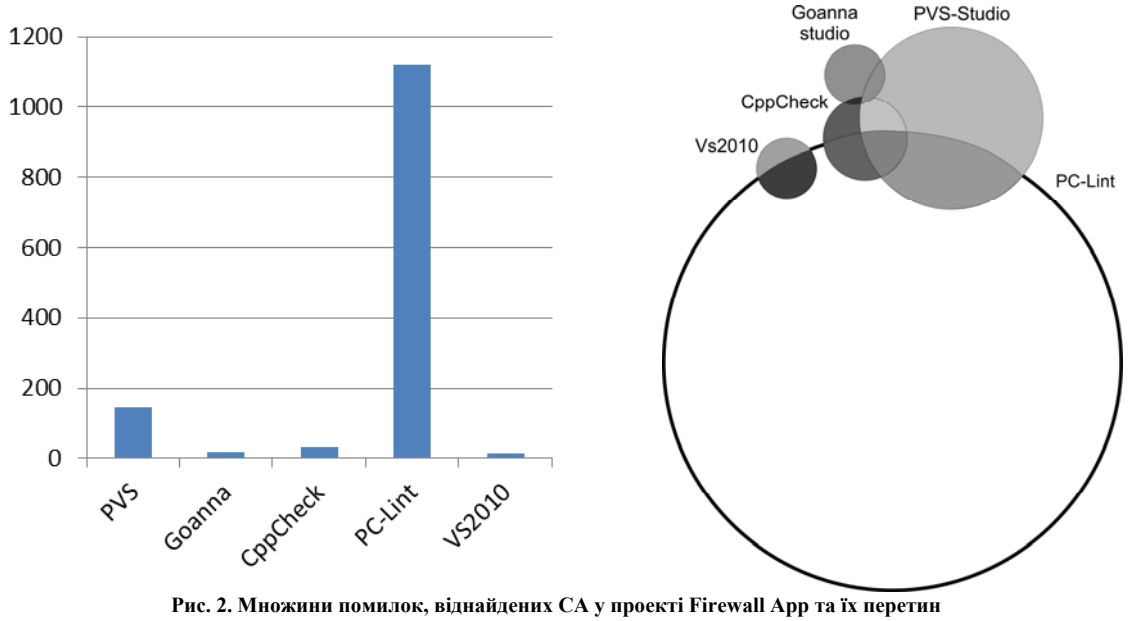

Рис. 2. Множини помилок, віднайдених СА у проекті Firewall App та їх перетин

Наступний фрагмент вихідного коду з проекту Notepad++ демонструє проблему співставлення результатів функціонування різних СА:

```
if(nCode < 0)     {CallNextHookEx(hookMouse, nCode, wParam, lParam);
                   return 0;          }
```

Змінна nCode у даному випадку була оголошена як беззнакова, тому операція порівняння для неї немає сенсу. Чотири СА віднайшли цю помилку та включили її у свої звіти. Повідомлення, що були сформовані СА наведені у таблиці 2.

З таблиці 2 слідує, що статичні аналізатори надали вказаній помилці різний рівень важливості та видали різні інформаційні повідомлення про неї. Так СА PVS-Studio відніс її до категорії найбільш критичних помилок, а CppCheck – стилістичних. Останній засіб при виявленні даного дефекту вказав на стрічку вихідного коду, де змінна була оголошена, в той час, як інші аналізатори вказували на місця її використання. В результаті CppCheck повідомив про одну помилку пов'язану з даною ситуацією, інші аналізатори – про чотири. Обидва підходи надають можливість для усунення дефекту, але роблять некоректним пряме співставлення загальної кількості помилок, віднайденими різними СА.

Таблиця 1

Результати функціонування статичних аналізаторів

ПЗ \ СА	Visual Studio 2010			PVS-Studio			Goanna Studio			PC-Lint			CppCheck		
	Помил.	Попер.	Інш.	Помил.	Попер.	Інш.	Помил.	Попер.	Інш.	Помил.	Попер.	Інш.	Помил.	Попер.	Інш.
Asteroid (14357)	0	0	0	5	7	107	0	60	0	206	2881	14003	7	24	70
Solar Systerm (16789)	0	0	0	6	7	125	0	38	0	207	3247	15493	14	24	78
FolderSize (1118)	0	18	0	12	16	5	0	3	0	3948	240	917	0	0	20
Notepad++ (31174)	0	2	0	1789	529	2040	0	85	0	65040	6400	39242	4	155	132
Emule (139119)	0	142	0	2954	3089	760	0	554	0	91199	46532	228780	18	473	1233
Firewall App(1862)	0	19	0	5	11	135	0	17	0	432	165	522	0	3	35
LogAnalyzer (866)	0	2	0	4	9	0	0	0	0	664	42	153	1	0	1
Shareaza	0	0	0	1098	1660	689	0	1879	0	406465	96682	54727	15	11	1055
Yafe (1388)	0	19	0	43	36	4	0	14	0	3936	497	1085	0	19	9
ultraMaGE (9483)	0	133	0	200	539	47	0	129	0	4143	2254	5964	3	1	46
NetBIOS Enumerater (4592)	0	473	0	107	182	84	0	273	0	290	997	2274	3	3	104
RemoteControlCenter (5369)	0	10	0	121	42	108	0	0	0	181	1607	2690	1	20	109
Antispy (2736)	0	9	0	1	0	0	0	0	0	82	4	11	0	1	4
TaskSwitchXP (3862)	0	3	0	72	61	217	0	24	0	54	498	879	0	1	11
7Z (6424)	0	1	0	92	140	6	0	153	0	568	3404	7230	0	244	259
Ac3 Filter (14609)	0	0	0	288	1725	766	0	31	0	9	4784	10049	71	402	92
Crystal DiskINFO (9007)	0	0	0	244	1238	544	0	350	0	2582	3044	5878	6	30	89
Crystal DiskMark (1785)	0	0	0	13	38	294	0	11	0	902	533	1235	0	23	138
Waintraball (5487)	0	37	0	197	474	5	0	111	0	1084	381	3006	0	0	63
BlackBox (6123)	0	988	0	88	303	27	0	362	0	5261	2084	4564	4	136	25
HideThatWindow (615)	0	3	0	13	2	0	0	2	0	716	153	466	0	0	7

Таблиця 2

Класифікація помилок різними засобами СА

№	СА	Категорія	Повідомлення	Кількість відмічених помилок
1	PC-Lint	Warning	Relational operator '<' always evaluates to 'false'	4
2	PVS-Studio	1 lvl	Variable `nCode' is unsigned and checked to be positive	4
3	Goanna Studio	Warning	Expression 'nCode < 0' is always false. Unsigned type value is never < 0	4
4	CppCheck	Style	UnsignedLessThanZero	1

Дослідження виявило, що не завжди помилки, що є у наборі правил СА ними виявляються. Так помилка, що розглядалась у таблиці 2, була виявлена чотирма СА. Вона пов'язана з логічним оператором порівняння. Розробник при виборі засобу СА схильний вважати, що якщо СА містить правило для виявлення певного типу помилок і воно декілька разів дійсно спрацювало, то варто очікувати виявлення усіх помилок подібного типу. Результати вказують на те, що таке твердження не є вірним. Наведений фрагмент вихідного коду був відмічений як помилковий лише двома СА PC-Lint та PVS-Studio:

```
BOOL MATCH;
        …
        if((!MATCH)&&(empty_space >= 0))
               {      BGHS[empty_space].gridmenu = menuid;
                  returnvalue=empty_space;          }
            if(MATCH)                          //PC-Lint
                {          return returnvalue+MAX_GRIDS;          }
            if((!MATCH)&&(empty_space == -1))          //PVS, PC-Lint
                {          return -1;          }
```

Змінна MATCH оголошена як логічна типу BOOL і використовується у операторі порівняння. З трьох операторів порівняння PVS-Studio вказав, що умова завжди вірна лише для третього порівняння, а PC-Lint сформував додаткове повідомлення і для другого. Інші СА на дану помилку не виявили.

Отже, кількість віднайдених помилок у ПЗ не є критерієм для оцінки якості проведеного статичного аналізу. Ряд аналізаторів виявляє лише частину дефектів у програмному забезпечені, тому використання одного додатку не гарантує виявлення усіх помилок. Для забезпечення більшої ефективності виявлення помилок у ПЗ потрібне використання декількох засобів СА. Вибір СА повинен ґрунтуватись на базі правил пошуку дефектів, перевірку яких вони забезпечують.

Висновки

Застосування статичного аналізу є одним з засобів оцінки якості програмного забезпечення. В процесі дослідження ефективності використання СА було виявлено, що на сьогодні класифікація типів помилок не стандартизована. Бази правил не налаштовуються на тип ПЗ, що в сукупності не дає можливості в повній мірі використовувати усі переваги статичного аналізу.

Напрямом подальших досліджень є розробка інтелектуальної технології вибору СА відповідно до типу та функціональних особливостей програмного забезпечення, що тестується.

Література

1. Основы инженерии качества программных систем / [Ф.И. Андон, Г.И. Коваль, Т.М. Коротун та ін. – К. : Академпериодика.– 2007 р. – 678 с..

2. Coverity Scan 2011 Open Source Integrity Report. / Coverity, Inc. - Coverity White Paper. – 2011. – 24 р.

3. Соммервілл Іан. Инженерия программного обеспечения 6-е издание / Соммервілл Іан. – Вільям, 2002 р. – 624 с.

4. SWEBOK Guide to the Software Engineering Body of Knowledge. / Washington: IEEE Computer Society — 2004 р. —37р.

5. Rui Lopes, Diogo Vicente, Nuno Silva. Static Analysis tools, a practical approach for safety-critical software verification. / Critical Software SA Parque Industrial de Taveiro. - Lote 48, 3045-054 Coimbra, Portugal. – 2009. – 12 р.

6. Совершенный код. Мастер-класс. – М. : Видавницько-торговий дім «Русская Редакция» ; СПб. : Питер, 2005 р. – 896 с.

Надійшла 14.9.2012 р.
Статтю представляє: д.т.н. Поморова О.В.