УДК 004.415.5

О.В. ПОМОРОВА, Д.О. ІВАНЧИШИН

Хмельницький національний університет, Хмельницький, Україна

ПОРІВНЯЛЬНИЙ АНАЛІЗ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ СТАТИЧНИХ АНАЛІЗАТОРІВ С++ ВИХІДНОГО КОДУ

Проведено аналіз результатів функціонування найбільш розповсюджених засобів для статичного аналізу вихідного коду. Досліджено ефективність статичних аналізаторів вихідного коду . У результаті дослідження з'ясовано, що навіть у робочих та якісних з точки зору компілятора програмних продуктах засоби статичного аналізу дозволяють виявити велику кількість помилок, які можливо та доцільно було б усунути на більш ранніх етапах життєвого циклу, а саме на етапах реалізації та тестування. Також визначені найбільш ефективні із розглянутих засоби статичного аналізу вихідного коду.

***Ключові слова**: статичний аналіз, оцінка якості, програмне забезпечення.*

Вступ

На сьогодні однією з мов програмування, що найчастіше застосовується при створенні системного та критично важливого програмного забезпечення є С++. Вона містить набір засобів для створення ефективних програм різного рівня складності, від низькорівневих утиліт до складних програмних комплексів. Будучи однією з найпотужніших мов програмування, С++ вимагає високої кваліфікації програмістів, що її застосовують.

Висока складність та значний розмір сучасних програмних систем спонукають до розробки та застосування нових методів виявлення наявних дефектів.

Статичний аналіз вихідного коду [1] є одним з методів оцінки якості програмного забезпечення (ПЗ), що застосовується на ранніх етапах розробки і не вимагає повної завершеності циклу розробки. Для критичного ПЗ статичний аналіз надає змогу підвищити якість вихідного коду, його безпеку та надійність. З економічної точки зору використання статичного аналізу є найбільш ефективним засобом і дозволяє значно скоротити витрати часу та ресурсів на етапах тестування та впровадження систем. На рис.1 подано залежність середньої вартості виправлення дефектів від часу їх внесення та виявлення [2].

Звичайно, витрати на придбання статичних аналізаторів (СА) та їх впровадження є високими, однак вони виправдані економією засобів на подальших етапах розробки. На рис.2 наведено залежність вартості усунення та розподіл дефектів ПЗ в залежності від стадії розробки. По мірі проходження проектом етапів розробки вона може зростати експоненційно [3].

На сьогодні додатки для статичного аналізу С++ вихідного коду представлені широким колом засобів від Lint-базованих синтаксичних аналізаторів відповідності стандартів до аналізаторів, що використовують формальні методи верифікації. Відмінність алгоритмів функціонування СА призводить до значної різниці та диференціації результатів їх роботи. В свою чергу це створює потребу дослідження ефективності та доцільності використання подібних додатків.

1. Застосування статичного аналізу для оцінки якості ПЗ

Основний відсоток помилок у програмних засобах виявляється на етапах тестування та експлуатації. Поява помилок на етапі експлуатації часто призводить до катастрофічних наслідків, тому актуальною задачею є виявлення помилок на більш ранніх етапах розробки ПЗ.

На сьогодні існує декілька основних методів, що дозволяють цього досягти:

- Unit-тестування;
- верифікація програм;
- статичний аналіз;
- динамічний аналіз.

Статичний аналіз є одним із засобів усунення критичних помилок, котрі важко виявити іншими методами тестування. Він включає наступні типи перевірок дефектів:

семантичну,
виділення пам'яті,
логічних операторів,
проблем включень, безпеки,
жорстку перевірку типів,
аналіз метрик.

© О.В. Поморова, Д.О. Іванчишин

	Час виявлення дефекту				
Час внесення дефекту	**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

Статичний аналіз

Рис. 1. Залежність середньої вартості виправлення дефектів від часу їх внесення та виявлення

Рис.2. Вартість усунення та розподіл дефектів ПЗ в залежності від етапу розробки

Статичний аналіз коду не вимагає завершеності ПЗ та його придатності до виконання. В цій особливості полягає основна перевага даного підходу в порівняні з динамічним аналізом, який виконує дослідження програм тільки під час їх виконання. СА віднаходять помилки на ранній стадії створення проекту, зазвичай, перед побудовою виконуваного файлу. Це є суттєвою перевагою для проектів великих вбудованих систем, де розробники не можуть використовувати засоби динамічного аналізу до тих пір, поки програмне забезпечення не буде придатне до запуску на цільовій системі.

В залежності від інструментів, що застосовуються, глибина аналізу може варіюватись від визначення поведінки окремих операторів до аналізу усього вихідного коду.

На етапі статичного аналізу виявляються та описуються області вихідного коду, що містять приховані вразливості, логічні помилки, дефекти реалізації, некоректні ситуації при виконанні паралельних операцій, рідко виникаючі граничні умови та ін. Інструменти статичного аналізу можуть передбачити всі потенційні шляхи виконуваного коду – динамічний аналіз, зазвичай, охоплює не більше 80% коду.

Місце статичного аналізу у розробці ПЗ представлене на рис. 3.

Для засобів статичного аналізу характерний ряд недоліків. Оскільки, під час статичного аналізу виконується передбачення поведінки програми, засноване на моделі вихідного коду, то інколи можуть виявлятись дефекти та помилки, які в дійсності не існують. Хоча розробники сучасних додатків для статичного аналізу декларують для своїх продуктів покращені техніки, котрі дозволяють уникнути хибних спрацювань, повністю усунути цей недолік неможливо.

Незважаючи на вказані недоліки, статичний аналіз є обов'язковим елементом при розробці програмного забезпечення у багатьох компаніях, включаючи компанії, що займаються розробкою критичного програмного забезпечення. У вимогах NASA до розробки програмного забезпечення вказано, що кожна зміна коду для критичних додатків повинна проходити перевірку статичними аналізаторами[4]. В стандартах ESA також приводяться вимоги щодо їх застосування [5].

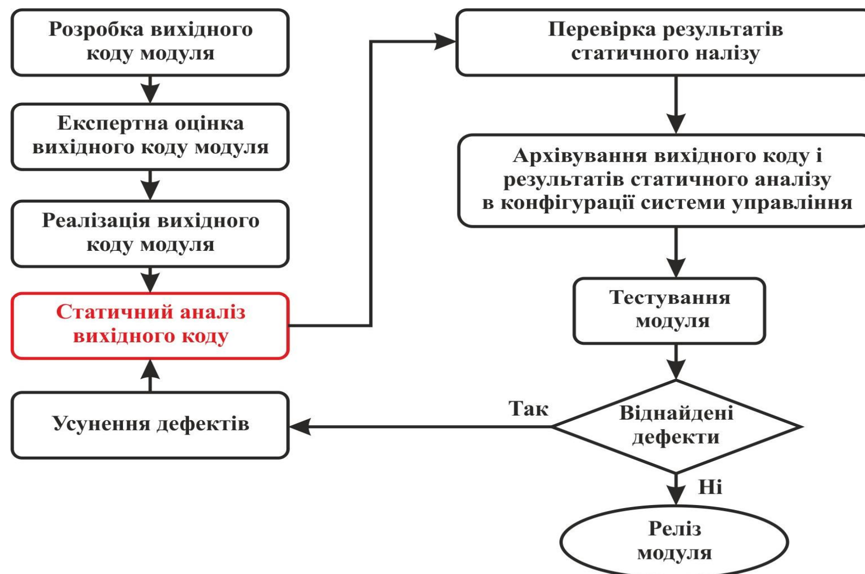

Рис.3. Місце СА у розробці ПЗ

2. Засоби для статичного аналізу програмного забезпечення

На сьогодні найбільш поширеними та розвинутими СА вважаються наступні продукти: Gimpel Software FlexeLint(PC-Lint), PVS-Studio, Red Lizard Software Goanna Studio, Parasoft C++ Test, CppCheck, Klocwork Insight, Coverity Static Analysis, Programming Research QA·C++ Source Code Analyzer.

Дослідження ефективності використана СА відбувалось для ПЗ середньостатистичного розміру від десяти до двадцяти тисяч стрічок програмного коду. Такий розмір ПЗ є широко поширеним серед прикладних додатків або завершених модулів складних програмних комплексів.

При виборі додатків для дослідження був вироблений наступний список вимог:

– можливість проведення автоматичного аналізу вихідного коду без його попередньої спеціальної підготовки;

– відслідковування впливу різних функцій програми на її поведінку при пошуку заданих ситуацій;

– відсутність обмежень на розмір вихідного коду.

Також додатковим критерієм для вибору одного з додатків став принцип його розповсюдження - freeware. Оскільки, більшість засобів для статичного аналізу мають достатньо високу вартість, доцільним є порівняння та оцінка ефективності їх безкоштовних аналогів.

За зазначеними вимогами був сформований наступний список статичних аналізаторів для дослідження:

– Gimpel PC-Lint в комплексі з Visual Lint для інтеграції з IDE. Аналіз вихідного коду з метою виявлення помилок різного типу. 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 в залежності 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, зазвичай, не виявляють компілятори. Ціна – Freeware.

3. Результати статичного аналізу програмного забезпечення

Для порівняння ефективності використання обраних СА були визначені основні вимоги, що ставились до тестових зразків:

– завершені продукти компіляція та виконання яких відбувається без помилок;

– вихідний код має від десяти до двадцяти тисяч стрічок коду;

– відсутність попереджень, помилок та повідомлень пов’язаних з якістю вихідного коду зі сторо-

ни середовища розробки. В дослідженні використовувався вбудований засіб статичного аналізу IDE Microsoft Visual Studio 2010;

– вік проекту не більше трьох років та відсутність використання застарілих бібліотек;

– відкритий вихідний код.

Відповідно, на базі визначених вимог для дослідження були обрані два C++ програмні проекти, розроблені в середовищі Microsoft Visual Studio 2010. Розмір проектів складав чотирнадцять та шістнадцять тисяч стрічок коду відповідно. Перший програмний засіб (Тест 1) призначений для створення нескладних тривимірних об'єктів, а другий (Тест 2) - відображає тривимірну модель сонячної системи. Обидва проекти створені з використання графічних бібліотек, що входять в Microsoft DirectX SDK.

В налаштуваннях СА вказувався режим функціонування з максимальною кількістю попереджень. Для другого тестового зразка додаток PC-Lint, що містить найбільшу базу правил, в результаті аналізу сформував звіт, що містив 72697 повідомлень, пов'язаних з якістю вихідного коду. Тобто, кількість повідомлень приблизно в п'ять раз перевищила обсяг самого вихідного коду. В першу чергу це пов'язано з великою кількістю повідомлень, що несуть інформацію про відповідність коду стандартам подібним до MISRA[6], рекомендації яких не завжди дійсно допомагають розробникам. Діаграму, що відображає результати аналізу за допомогою PC-Lint, подано на рис.4.

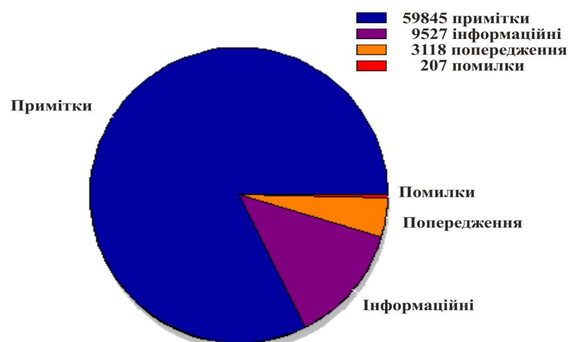

Рис.4. Результати аналізу PC-Lint, Тест 2

Подібні надлишкові для дослідження дані надавали і інші засоби, тому при аналізі результатів роботи СА розглядались лише попередження, що відносились аналізаторами до категорії важливих та мали безпосереднє відношення до працездатності програмного засобу. Результати аналізу тесових зразків представлено у табл. 1.

Microsoft Visual Studio 2010 при компіляції та виконанні двох тестових зразків ПЗ не виявила в них жодної помилки та не відрапортувала про жодне попередження, що відповідає поставленим до ПЗ вимогам.

Таблиця 1

Результати статичного аналізу

Пом. та попер. / Додатки	Тест 1		Тест 2	
	Помилки	Попередження	Помилки	Попередження
Visual Studio 2010	0	0	0	0
PVS-Studio	5	7	6	7
Goanna Studio	0	60	0	38
Visual Lint	0	206	0	207
CppCheck	30	69	44	75

Далі слідує додаток PVS-Studio, що загалом сигналізував про 119 попереджень для першого тестового зразка та 138 – для другого. Засіб має систему розподілу важливості помилок на три рівні. Попередження третього рівня важливості переважно відносились до методів покращення продуктивності роботи програми. Вони брались до розгляду тільки в аспекті порівняння наявності даних попереджень у результатах інших аналізаторів. До таблиці 1 занесені повідомлення лише з першої та другої групи. Критичних та першочергових для виправлення дефектів перший проект містив п'ять, а другий – шість. Частина з них пов'язані з потенційними помилками розіменування вказівників, а саме виконання операцій над ними після чого виконувалась перевірка їх значень на NULL. Про подібні проблеми сигналізували усі додатки крім самого IDE, але конкретно виявлені PVS-Studio попередження помічені, як критичні, були відмічені лише у PC-Lint і не містились у результатах роботи інших двох продуктів. Важливо відмітити, що хоча PC-Lint містив приведені повідомлення, загальний список подібних включень перевищував тисячу тільки для Тест 1 і всі вони відносились до категорії попереджень, а не критичних помилок.

Інші попередження були унікальними. Два з них пов'язані з втратою інформації з буфера даних при використанні функції memcpy. Декілька інших розглянемо більш детально, оскільки вони демонструють функціональні особливості роботи додатку PVS-Studio. Повідомлення про використання однакової конструкції у операторі IF:

```
else if( DeviceType ==
D3D10_DRIVER_TYPE_SOFTWARE )
      wcscpy_s( pstrDeviceStats, 256, L"WARP" );
   else if( DeviceType ==
D3D10_DRIVER_TYPE_HARDWARE )
      wcscpy_s( pstrDeviceStats, 256,
L"HARDWARE" );
   else if( DeviceType ==
D3D10_DRIVER_TYPE_SOFTWARE )
     wcscpy_s( pstrDeviceStats, 256,
L"SOFTWARE" );
```

Як слідує з фрагмента вихідного коду, розробники припустились помилки при наборі, відповідно, перша умова в операторі IF ідентична третій, яка в свою чергу ніколи не зможе бути виконаною.

Наступне повідомлення сигналізувало про використання однакових змінних у розділі умови оператора IF:

```
if( m_GamePad[iUserIndex].wButtons ||
        m_GamePad[iUserIndex].sThumbLX ||
m_GamePad[iUserIndex].sThumbLX ||
        m_GamePad[iUserIndex].sThumbRX ||
m_GamePad[iUserIndex].sThumbRY)
```

У даному випадку в складній умові оператора IF двічі використовується той самий параметр m_GamePad[iUserIndex].sThumbLX, хоча з контексту вихідного коду слідує, що у другому випадку повинен був вказуватись параметр m_GamePad[iUserIndex].sThumbLY.

Ще одне повідомлення пов'язане з оператором IF:

```
uint t;
…
if(t>=0...){…}else…;
```

Оскільки змінна t була оголошена, як беззнакова і не може приймати значення менше нуля, дана умова немає сенсу, а відповідно ELSE частина виразу ніколи не може бути виконаною. Ні конкретно цих, ні помилок подібного класу жоден з інших СА коду не віднаходив.

Відповідно, попередження також містили вказівки на ситуації, що потенційно можуть бути помилковими, але не критичними. До них відносяться порівняння різних типів, переприсвоєння значень, вказівники на локальні зміні та ін. Одна з вищевказаних проблем віднайдених PVS-Studio:

```
CDXUTControl::CDXUTControl(
CDXUTDialog* pDialog )
{
  m_pDialog = pDialog;
  …
/*дії з m_pDialog в пропущеному коді відсутні/*
  m_pDialog = NULL;
```

Тобто зміна m_pDialog втрачала початкове значення жодного разу його не використавши. Таких ситуацій було віднайдено 14 для двох проектів.

Розглянемо результати функціонування СА Red Lizard Goanna Studio. Усі повідомлення даного аналізатора відносились до категорії попереджень(warnings) і для обох проектів їх було відмічено 98. Більше 90% з них відносять до можливих проблем, пов'язаних з перетворенням типів. Наприклад, перетворення беззнакових типів у знакові, що може призвести до виходу за межі значень типу. З розглянутих додатків про подібні помилки рапортував лише PC-Lint. Переважна більшість інших помилок сигналізувала про можливі проблеми з розіменування вказівників. Декілька повідомлень сигналізували про вказівники на локальні зміні, коли додаток віднаходив ситуації у яких адреса локальних змінних зберігалась в середині класу в змінній з вищим рівним видимості. Подібні повідомлення були присутні і PVS-studio.

Gimpel PC-Lint, що використовувався в комплексі з Visual Lint для інтеграції з IDE Microsoft Visual Studio 2010. В сумі для двох проектів звіт даного додатку включав 134366 повідомлень, що поділяються на шість категорій за важливістю: примітки, інформаційні повідомлення, попередження, помилки, внутрішні помилки (баги) та критичні помилки. Після аналізу додаток не виявив у тестових зразках помилок останніх двох категорій. Повідомлень про помилки було по 206 та 207 для першого та другого проекту відповідно. Для прикладу, для першого тестового проекту велика кількість застережень відносно інших засобів пояснюється значною кількістю повідомлень про надлишкове оголошення типів та правила MISRA, невиконання яких даний продукт класифікує, як помилку. Помилок, що не стосувались оголошень типів та правил MISRA виявилось лише три. Вони також були пов'язані з певними синтаксичними вимогами до C++ коду, а не з його функціональними властивостями. Саме тому в Таблиці 1 помилки в класифікації PC-Lint представлені в графі попередження. Подібна ситуація повторилась і для другого проекту.

CppCheck – єдиний безкоштовний додаток у дослідженні. Всього було відрапортовано про 214 підозрілих ситуацій у першому проекті та 238 – у другому. Важливих виявилось 30 та 44 відповідно. Більша частина з них пов'язана з розіменуванням вказівників, що вже відзначалось у інших додатках, а також виявлені проблеми з втратами пам'яті. В класі попереджень основними підозрілими ситуаціями були порівняння різних типів, для прикладу boolean та integer, і відсутність ініціалізації членів класів. Особливістю даного продукту є те, що подібні помилки найчастіше зустрічались по декілька разів для різних частин вихідного коду і в результаті загальна кількість різних помилок, про які рапортував додаток, не перевищила десяти для двох проектів.

Для коректності порівняння результатів функціонування СА в табл. 2 були занесені значення загального числа віднайдених додатками помилок для обох проектів, що відносились тільки до потенційних функціональних проблем програмних продуктів. Отже, з табл. 2 слідує, що найбільший відсоток помилок, що дійсно пов'язані з проблемами у вихідному коді у PVS-Studio, при чому більшість з них були унікальними.

Таблиця 2
Відсоток функціональних помилок

Додатки	Загальна к-ть помилок	% від заг. к-ті
PVS-Studio	21	84
Goanna Studio	11	11,2
Visual Lint	0	0
CppCheck	110	50,4

Також даний додаток здатен віднаходити функціональні проблеми пов'язані саме з логікою побудови коду, на що не здатен жоден інший з протестованих СА. Дослідження вказують на те, що саме він є найбільш ефективним засобом статичного аналізу для виявлення логічних помилок, помилок набору, невизначених ситуацій, невірного використання базових функцій та фрагментів вихідного коду, що не мають сенсу.

Висновок

На сьогодні статичний аналіз є одним з ключових засобів оцінки якості програмного забезпечення. Дослідження показало, що навіть у робочих та якісних з точки зору компілятора програмних продуктах засоби статичного аналізу дозволяють виявити велику кількість помилок, які можливо та доцільно було б усунути на більш ранніх етапах життєвого циклу, а саме на етапах реалізації та тестування.

Подальших досліджень потребують питання ефективності використання СА для ПЗ різного призначення: бізнес додатки, прикладне ПЗ, ігрові додатки, системне ПЗ, пакети для наукових розрахунків та, звичайно, ПЗ критичного застосування.

Література

1. *Rui, Lopes. Static Analysis tools, a practical approach for safety-critical software verification [Text]/ Lopes Rui, Vicente Diogo, Silva Nuno. – Critical Software SA Parque Industrial de Taveiro, Lote 48, 3045-054 Coimbra, Portugal.*
2. *Совершенный код. Мастер-класс [Текст]: пер. с англ. – М.: Издательско-торговый дом «Русская Редакция» ; СПб.: Питер, 2005. – 896 с.*
3. *Capers J. Applied Software Measurement [Text] / J. Capers. McGraw Hill, 3rd edition, 2008. – 662 p.*
4. *Design Development Test and Evaluation (DDT&E) Considerations for Safe and Reliable Human Rated Spacecraft Systems. Volume II, April, 2008.*
5. *Software – Part 2: Document requirements definitions (DRDs), 31 March 2005; ECSS--E--40 Part 2B.*
6. *MISRA AC INT: Introduction to the MISRA guidelines for the use of automatic code generation in automotive systems, ISBN 978-906400-00-2 (PDF), November 2007.*

Поступила в редакцію 23.02.2012

Рецензент: д-р техн. наук, проф., зав. каф. программной инженерии И.Б. Туркин, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков, Украина.

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ СТАТИЧЕСКИХ АНАЛИЗАТОРОВ С + + КОДА

О.В. Поморова, Д.А. Иванчишин

Проведен анализ результатов функционирования наиболее распространенных средств для статического анализа исходного кода. Исследована эффективность статических анализаторов исходного кода. В результате исследования выяснено, что даже в рабочих и качественных с точки зрения компилятора программных продуктах средства статического анализа позволяют выявить большое количество ошибок, которые возможно и целесообразно было бы заменить на более ранних этапах жизненного цикла, а именно на этапах реализации и тестирования. Также определены наиболее эффективные из рассмотренных средства статического анализа исходного кода.

Ключевые слова: статический анализ, оценка качества, программное обеспечение.

COMPARATIVE ANALYSIS OF C + + SOURCE CODE STATIC ANALYZERS EFFECTIVENESS

O.V. Pomorova, D.O. Ivanchyshyn

The analysis of the functioning results of most common tools for static analysis of source code was investigated. The effectiveness of static analyzers of source code was researched. Our study found that even in working and quality in terms of compiler software products static analysis is able to detect many errors that can and should be removed at earlier stages of the life cycle, namely at the stages of implementation and testing. Also the most effective of proposed static source code analysis tool is identified.

Key words: static analysis, quality assurance, software.

Поморова Оксана Вікторівна – д-р техн. наук, професор, завідувач кафедри системного програмування Хмельницького національного університету, e-mail: o.pomorova@gmail.com.

Іванчишин Дмитро Олександрович – аспірант кафедри системного програмування Хмельницького національного університету, e-mail: dmytro_ivanchyshyn@ukr.net.