УДК 004.89

Іспірянц А.Р., магістрант, Заяц В.І., к.т.н., доцент, науковий керівник

РОЗРОБКА ТА ДОСЛІДЖЕННЯ СТАТИЧНОГО АНАЛІЗТОРА КОДУ

Запорізька державна інженерна академія, кафедра ПЗАС

Тестування є важливою частиною процесу розробки додатків. Існує декількарізних видів тестування, в тому числі і два види, що стосуються програмного коду: статичний аналіз і динамічний аналіз.

Динамічний аналіз проводиться над виконуваним кодом компільованою програми. При цьому перевіряється тільки поведінка застосунку, залежна від користувача, тобто тільки той код, який виконується під час тесту. Динамічний аналізатор може знаходити проблеми виділення пам'яті, вимірювати продуктивність програми, отримувати стек викликів, тощо.

Статичний аналіз дозволяє перевіряти вихідний код програми до її виконання. Зокрема, будь-який компілятор проводить статичний аналіз при компіляції. Однак, у великих реальних проектах часто виникає необхідність перевірити весь код на предмет відповідності деяким додатковим вимогам.

Статичний аналізатор знаходить рядки вихідного коду, які, імовірно, не відповідають прийнятому стандарту кодування і відображає діагностичні повідомлення, щоб розробник міг зрозуміти причину проблеми. Процес статичного аналізу аналогічний компіляції, тільки при цьому не генерується ні об'єктний, ні виконуваний код.

Однією з найскладніших, найцікавіших та найменш розвинутих областей статичного аналізу є пошук неявних помилок. Ці помилки дуже складно помітити, вони створюються внаслідок копіювання коду, неуважності розробника або специфіки мови програмування. Зазвичай, на пошук таких помилок витрачаються години, тому що їх складно помітити або ж вони були виявлені через великий проміжок часу з моменту написання.

Число методів статичного аналізу і самих аналізаторів зростає з року в рік, і це означає, що інтерес до статичних аналізаторів коду зростає. Причина зацікавленості полягає в тому, що розробляється програмне забезпечення стає все більш і більш складним і, отже, перевіряти код вручну стає неможливо.

Залежно від використовуваного інструменту глибина аналізу може варіюватися від визначення поведінки окремих операторів до аналізу, що включає весь наявний вихідний код. Способи використання отриманої в ході аналізу інформації також різні - від виявлення місць з можливими з помилками до формальних методів, що дозволяють математично довести будь-які властивості програми).

Головна перевага статичного аналізу полягає в можливості суттєвого зниженні вартості усунення дефектів в програмі. Чим раніше помилку виявлено, тим менше вартість її виправлення. Так згідно з даними, наведеними в книзі Макконнелла "Досконалий Код", виправлення помилки на етапі тестування обійдеться в десять разів дорожче, ніж на етапі конструювання (написання коду) [1]. Інструменти статичного аналізу дозволяють виявити велику кількість помилок етапу конструювання, що істотно знижує вартість розробки всього проекту.

Метою роботи є проектування і розробка системи аналізу і виявлення помилок у вихідному коді програми. Подібне завдання включає в себе використання багатьох методів статичного аналізу.

Досягнення поставленої мети передбачає:

- Дослідження та вибір компонента для компіляції даних та побудови синтаксичного дерева.
- Розробку алгоритмів статичного аналізу.
- Класифікацію основних видів помилок.
- Інтеграцію аналізатора з середовищем розробки.
- Візуалізацію знайдених помилок.

- Аналіз відкритих проектів (Github).

Важливим моментом є вибір інструментів реалізації та експериментальне порівняння їх ефективності, гнучкості та зручності використання. Під час вибору стеку технологій для розробки системи порівнювались такі платформи для компіляції:

- Resharper.
- Roslyn.

Не зважаючи на швидкодію та зручний API Resharper, даний інструмент був відкинутий через неможливість запуску платформи окремо від середовища розробки та через ліцензійні умови [3]. Roslyn у свою чергу є відкритим, швидко розвивається та дозволяє запускати статичний аналізатор як самостійний за стосунок [4].

Синтаксичне дерево Roslyn є базовим елементом, необхідним для аналізу коду. Саме по ньому відбувається переміщення в ході аналізу. Дерево будується на основі коду, наведеного у файлі.

Семантична модель Roslyn надає інформацію про об'єкти і про типи об'єктів. Це дуже потужний інструмент, що дозволяє проводити глибокий і складний аналіз. Саме тому важливо мати коректну компіляцію і коректну семантичну модель.

Спроектована система дозволяє алалізувати та виконувати пошук неявних помилок у вихідному коді C# 7. Вона інтегрована в Visual Studio 2017, але також може бути викликана як самостійний застосунок, що дає можливість автоматизувати процес статичного аналізу та налаштувати його як окремий крок Continuous Integration).

Висновки:

1. Було детально досліджено предметну область статичного аналізу для пошуку помилок у вихідному коді програми.
2. Розглянуто існуючі платформи компіляції та детально вивчено базові принципи, такі як синтаксичне дерево, семантична модель, лексеми, вузли.
3. Основним інструментом для розробки статичного аналізатора був обраний Roslyn, який дозволяє розбирати і аналізувати код до найдрібніших подробиць. Це відкриває простір для створення різних додатків на його основі, у тому числі статичних аналізаторів.

Література

1. Steve McConnell, "Code Complete, 2nd Edition" Microsoft Press, Paperback, 2nd edition, Published June 2004, 914 pages, ISBN: 0-7356-1967-0. (Part 24.3. Reasons to Refactor)
2. James Alan Farrell Compiler Basics [Електронний ресурс]. - Режим доступу до ресурсу: http://www.cs.man.ac.uk/~pjj/farrell/compmain.html (дата звернення 24.05.2017).
3. Joel Jones Abstract syntax tree implementation idioms[Електроннийресурс]. - Режимдоступудоресурсу: https://www.researchgate.net/publication/245153015_Abstract_Syntax_Tree_Implementation_Idioms (датазвернення 24.05.2017).
4. .NET Compiler Platform ("Roslyn") [Електроннийресурс]. - Режимдоступудоресурсу: https://github.com/dotnet/roslyn (датазвернення 24.05.2017).