УДК 004.05

А.С. Богатиренко, А.А. Недолужко
Науковий керівник – Доренський О.П., викладач
Кіровоградський національний технічний університет

Застосування метода статичного тестування для виявлення та усунення помилок ПЗ

Тестування – невід'ємна складова процесу програмної інженерії, один з методів подальшого вдосконалювання якості розроблених програмних засобів за допомогою усунення дефектів, що залишилися не виявленими іншими видами перевірок.

Тестування можна розглядати як процес семантичного налагодження (перевірки) програми, яка полягає у виконанні послідовності різних наборів контрольних тестів, для яких заздалегідь відомий результат. Тобто тестування передбачає виконання програми і отримання конкретних результатів виконання тестів.

Упродовж свого розвитку проблематика тестування розвивалася паралельно у декількох напрямках, зокрема:

- дослідження та розроблення методів тестування й критеріїв адекватності тестування (відповідно до методів);
- визначення кількісних метрик тестування та критеріїв його завершення;
- створення програмних інструментів підтримки тестування;
- формування моделей оцінювання процесу тестування та його вдосконалення.

Статичні методи використовуються при проведенні інспекцій і розгляді специфікацій компонентів без їхнього виконання.

Техніка статичного аналізу полягає в методичному перегляді (або огляді) і аналізі структури програм, а також у доведенні їхньої правильності вручну. Статичний аналіз направлений на аналіз документів, розроблених на всіх процесах життєвого циклу (ЖЦ) і полягає в інспекції вхідного коду і наскрізного контролю програми.

Інспекція ПЗ – це статична перевірка відповідності програми заданим специфікаціями, проводиться шляхом аналізу різних представлень результатів проектування (документації, вимог, специфікацій, схем або коду програм) на процесах ЖЦ. Перегляди й інспекції результатів проектування і відповідності їх вимогам замовника забезпечують більш високу якість розроблюваного ПЗ.

При інспекції програм розглядаються документи робочого проектування на процесах ЖЦ разом з незалежними експертами й учасниками розробки ПЗ. На початковому процесі проектування інспекція припускає перевірку повноти, цілісності, однозначності, несуперечності і сумісності документів з вимогами до програмної системи. На процесі реалізації системи під інспекцією розуміють аналіз текстів програм на дотримання вимог стандартів і прийнятих керівних документів технології програмування.

Ефективність такої перевірки полягає в тому, що залучені експерти намагаються подивитися на проблему «з боку» і піддають її всебічному критичному аналізу.

Ці прийоми дозволяють на більш ранніх процесах проектування знайти помилки або недоробки шляхом багаторазового перегляду вхідного опису програми. Символьне тестування застосовується для перевірки окремих ділянок програми на вхідних символьних значеннях.

Крім того, розробляється безліч нових засобів автоматизації символьного виконання програм. Наприклад, автоматизований засіб статичного контролю для

мовно-орієнтованої розробки, інструменти автоматизації доведення коректності й автоматизований апарат мереж Петрі.

Масове виробництво і повсюдна доступність 64-бітових процесорів привели розробників додатків до необхідності розробки 64-бітових версій своїх програм. Адже для того, щоб користувачі могли отримати реальні переваги від використання нових процесорів, додатки повинні бути перекомпілювати для підтримки 64-бітної архітектури. Основні проблеми при переносі коду виявляються в додатках, розроблених з використанням низькорівневих мов програмування типу C або C++. Усі високорівневі конструкції та бібліотеки C++ в кінцевому підсумку реалізовані з використанням низькорівневих типів даних, таких як покажчик, машинне слово і т.п. Оскільки при зміні архітектури ці типи даних змінюються, то і поведінка програм також може змінитися.

Для того щоб б впевненим у коректності програми на новій платформі, необхідно вручну виконати перегляд коду і переконатися в його коректності. Тому виникає завдання пошуку в вихідному коді програми тих місць, які при перенесенні з 32-бітної на 64-бітову архітектуру можуть працювати неправильно. Тестують команди за послідовністю їх розташування в тексті програми. Під час статичного тестування тестують не всі можливі шляхи в графі програми, тому частина помилок може залишитися невиявленою; програмний код взагалі не виконується — його тестують тільки шляхом логічного аналізу.

Для статичного тестування використовують такий інструментальний засіб, як компілятор. Виявивши, наприклад, синтаксичну помилку або неправильну операцію, він видає відповідне повідомлення. Компонувальник теж може видавати корисні повідомлення, зокрема імена змінних та інших об'єктів, що повторюються, посилання на неоголошені змінні та функції.

Статичний аналіз програми спеціалісти-тестувальники виконують неавтоматизовано. Вони читають вихідний код програми, обговорюють його і знаходять, як правило, досить багато помилок. Робота вичитування програмного коду досить рутинна, але необхідна.

Існують статичні аналізатори коду (як, наприклад, Gimpel Software PC-lint і Parasoft C++ test) призначені для комплексного забезпечення якості коду і містять кілька сотень аналізованих правил. У них також є деякі з правил, які аналізують коректність 64-бітних додатків. Однак, оскільки це аналізатори коду загального призначення, то їх використання для забезпечення якості 64-бітних додатків не завжди зручно. Це пояснюється, насамперед, тим, що вони не призначені саме для цієї мети. Іншим серйозним недоліком є їх зорієнтованість на модель даних, яка використовується в Unix-системах (LP64). У той час як модель даних, яка використовується в Windows-системах (LLP64), істотно відрізняється від неї. Тому застосування цих статичних аналізаторів для перевірки 64-бітних Windows-додатків можливо тільки після неочевидного додаткового налаштування.

Список літератури

1. Андон Ф.И., Коваль Г.И., Коротун Т.М., Лаврищева Е.М. Суслов В.Ю. Основы инженерии качества программных систем. Киев: Академпериодика.– Второе изд., 2007. – 680 с.
2. Weyuker E.J., Ostrand T.J. Theories of program testing and the application of revealing subdomains. IEEE Trans.Soft.Eng. – 1980. – V. 6, – №. 3, – P. 236–246.
3. Лаврищева Е.М., Коротун Т.М. Построение процесса тестирования программных систем. Проблемы программирования.–2002.– №1-2.– С. 272-281.
4. Бабенко Л.П., Лавришева Е.М. Основы программной инженерии. Киев: Знание, 2001. – 269 с.