УДК 004.415

Студ. Н.Ф. Черняк
Науч. рук. проф. П.П. Урбанович
(кафедра информационных систем и технологий, БГТУ)

МУТАЦИОННЫЙ АНАЛИЗ КАК СРЕДСТВО ПРОВЕРКИ НАДЕЖНОСТИ ТЕСТОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Процесс разработки программного обеспечения (ПО) неотрывно связан с контролем качества как уже существующего, так и нового кода. Важнейшая характеристика ПО – его надежность [1]. Основным средством тестирования надежности ПО является юнит-тестирование. Однако, оно предполагает написание нового кода для контроля качества другого кода. Отсюда вытекает необходимость наличия механизма, который может подтвердить эффективность кода юнит-тестов.

Подобным средством может выступать мутационный анализ [2, 3]. Он основан на изменении тестируемого участка кода путем внедрения в него изменений (мутантов) и повторного запуска соответствующих тестов для него. В случае, если тесты никак не «реагируют» на мутантов (т.е. тестирование проходит успешно), то можно сделать вывод о неэффективности кода юнит-теста. Важно понимать, что мутационное тестирование – это не хаотичное преобразование кода, а абсолютно предсказуемый и понятный процесс, который, при наличии одинаковых входных мутационных операторов, всегда выдает один и тот же список мутаций и результирующие метрики на одинаковом тестируемом исходном коде.

Основной метрикой в теории мутационного тестировании является коэффициент *MSI* (MutationScoreIndicator), вычисляемый по формуле [2]:

$$MSI = M_y/M, \quad (1)$$

где M – общее количество мутантов, M_y – количество «убитых» мутантов. Под общим количеством мутантов подразумевается число мутантов, которые были внесены в тестируемый участок кода. Убитым считается мутант, на внедрение которого тест среагировал, т. е. тест не прошел; в противном случае – он признается выжившим, так как не был обнаружен.

Алгоритм мутационного анализа в общем случае является достаточно устоявшимся и качественно не меняется с момента его формализации. Отличия появляются исключительно в деталях конкретных реализаций.

Традиционный алгоритм мутационного тестирования включает два «ручных» шага: “Ввод тестовой программы” и “Анализ и отметка

одинаковых мутантов”. Если первый является классическим взаимодействием пользователя с системой, то мануальная природа второго обусловлена исключительной высокой стоимостью автоматизации.

Шаг “Анализ и отметка одинаковых мутантов” является также достаточно неоднозначной проверкой, которую лучше переложить на человека. Основная часть алгоритма включает в себя внедрение мутантов в исходный код через построение абстрактного синтаксического дерева (АСД). АСД-модель отлично подходит для выявления специфических участков кода, которые могут быть подвергнуты мутации. Имея АСД, это достаточно легко сделать, оперируя объектами, которые представляют исходный код (например: Node, Expr, BinaryOp и т.д.). Такой подход используется в ведущем средстве мутационного анализа для языка PHP – *Infection* [4].

Таким образом, мутационный анализ предоставляет возможность оценить некоторую часть кода (в данном случае – юнит-тесты), исключая человеческий фактор, так как он подразумевает под собой исключительно автоматизированный процесс по внедрению мутантов. Это дает возможность гарантировать эффективность и того кода, который покрыт этими юнит-тестами.

Необходимо также отметить, что метод мутационного тестирования требует больших вычислительных затрат и до недавнего времени не пользовался популярностью. Это связано с тем, что для каждого внедренного мутанта должна быть собрана вся программа целиком и запущен тест [2]. В настоящее время, рост вычислительных мощностей, а также увеличение потребности в заведомо надежном и эффективном коде, повышают интерес к мутационному анализу.

ЛИТЕРАТУРА

1. Урбанович П. П. Защита информации методами криптографии, стеганографии и обфускации : учеб.-метод. пособие для студ/ П.П. Урбанович. – Минск: БГТУ, 2016. – 220 с.

2. Информационный ресурс Habr.com [Электронный ресурс]. – Режим доступа: https://habr.com/en/post/334394/. – Дата доступа: 28.03.2020.

3. Информационный ресурс Habr.com [Электронный ресурс]. – Режим доступа: https://habr.com/en/company/badoo/blog/462709/. – Дата доступа: 30.03.2020.

4. Jefferson A. Offutt. A Practical System for Mutation Testing: Help for the Common Programmer// A. Jefferson Offutt [et al] // ITC'94: Proceedings of the 1994 international conference on Test. – 1994. – P. 824–830.