Оптимизация тестового набора с сохранением качества автоматизированного тестирования

Филимонов Артём Александрович

Студент

Факультет компьютерных наук НИУ ВШЭ, Москва, Россия

E-mail: aafilimonov_5@edu.hse.ru

Научный руководитель *— Хританков Антон Сергеевич*

В текущих условиях, когда качество разрабатываемых программ имеет ключевое значение, на тестирование затрачивается большое количество ресурсов [1]. Однако качество самих тестов для нахождения ошибок редко оказывается достаточным, чтобы быть уверенным в качестве кода программы. Для оценки тестового набора может использоваться мутационное тестирование, которое тоже достаточно ресурсоемкое. Чтобы сократить расходы и поднять качество тестирования, можно оптимизировать мутационное тестирование.

Мутационное тестирование состоит из ряда этапов. Наименее изученным является этап сокращения или приоритизации тестового набора, так как небольшое количество работ посвящено данной теме [2]. Задача оптимизации мутационного тестирования ещё не решена, поэтому применение новых методов актуально.

Метод предлагаемый в данной работе основан на выборе тестов исходя из покрытия тестами мутантов. Он отличается от альтернатив тем, что для каждого мутанта применяется оптимизированный тестовый набор, и для каждого теста измеряется покрытие. Другие методы, например, используют жадный алгоритм [3] или алгоритм Флойда–Уоршелла для сокращения тестов.

Исходя из того, что тест-кейс покрывает только часть строк программного кода, то предполагается, что найдется мутант, который не будет находится среди этих строк. Значит такой тест бесполезен для выбранного мутанта. Основываясь на покрытии кода для каждого теста, исключаются те тесты, которые точно не смогут найти мутанта. Таким образом получается оптимальный набор тестов под каждого мутанта. Для достижения лучших результатов метод используется в комбинации с другими известными решениями.

Метод состоит из 6 этапов. Начинается с определения набора тестов для проекта. Далее для каждого теста записывается покрытие кода программы. Создаются мутанты и проверяются в соответствии

с покрытием кода тестами. Если оператор кода (мутант) находится в строках покрытия, то тест добавляется к мутанту. Определив наборы тестов для всех мутантов, они исполняются. Результаты выводятся в консоль или файл. Этапы чтение тестов, определение покрытия, создание мутантов разработаны с использование сторонних библиотек, так как такой функционал уже реализован в программах.

Часть алгоритма, в которой для мутанта определяется набор тестов, реализована полностью на основе предложенного метода. Когда для всех тестов известны результаты покрытия, и мутанты созданы, то начинается выборка для каждого из мутантов. В цикле для каждого тест сравниваются строки, входящие в покрытие и строка, в которой находится мутант. Если мутант входит в покрытие, то тест добавляется в набор тестов для мутанта. Таким образом для всех мутантов определяется оптимальный набор тестов. Далее, на каждом мутанте исполняется собственный набор тестов.

Для оценки результатов проводится эксперимент, который проверяет гипотезу о том, что внедрение нового метода оптимизации тестового набора сокращает время мутационного тестирования и снижает его стоимость при прочих равных условиях. Для подтверждения гипотезы используются метрики: количество отброшенных тестов при сохранении множества обнаруженных мутантов, точность выбора тестов, выявляющих мутантов. В среднем ожидается, что тестовый набор сократится на 20–40%, а точность выбора тестов увеличится на 15–30%. Эксперимент проводится на проектах написанных на Python. Для ряда выбранных проектов запускается мутационное тестирование в обычной форме и с использованием нового алгоритма. Затем результаты сравниваются и оцениваются.

Таким образом, разработанный метод позволяет использовать меньшие мощности для проведение тестирования, и как результат мутационное тестирование станет более доступным.

Литература

1. Papadakis M., Kintis M., Zhang J., Jia Y., Traon Y. and Harman M. Mutation Testing Advances: An Analysis and Survey // Advances in Computers, 2019, P. 275–378.
2. Jia Y. and Harman M. An Analysis and Survey of the Development of Mutation Testing // IEEE Transactions on Software Engineering, 2011, 37(5), P. 649–678.
3. Polo Usaola M., Reales Mateo P. and Pérez Lamancha B. Reduction of Test Suites Using Mutation // Fundamental Approaches to Software Engineering, 2012, P. 425–438.