Программная поддержка оптимизации тестовых наборов для проведения регрессионного тестирования

Цыркунович Е. В.
Кафедра интеллектуальных информационных технологий
Белорусский государственный университет информатики и радиоэлектроники
Минск, Республика Беларусь
e-mail: katya.tsirkunovich@gmail.com

***Аннотация*—Рассматривается проблема выбора тестовых наборов для проведения оптимального регрессионного тестирования. Предложен метод, определяющий минимальное число тестов, необходимых для повторной проверки правильности изменённой программы на фазе сопровождения.**

Ключевые слова: тест, множество тестов, набор тестов, тестовое покрытие, функция целесообразности.

I. ПОНЯТИЕ И МЕТОДЫ РЕГРЕССИОННОГО ТЕСТИРОВАНИЯ

Программы подвержены неизбежным изменениям, как бы хорошо разработаны они ни были первоначально. Их выполнение выявляет дефекты, которые должны быть исправлены. Контроль над изменениями - критический фактор для сохранения полезности программ. Для повторной проверки корректности функциональных возможностей в процессе разработки необходимо использовать регрессионное тестирование, то есть повторное тестирование части программы, зависящей от внесённых изменений.

Регрессионное тестирование гарантирует, что внесённые в код изменения корректны и не воздействуют неблагоприятно на другие блоки программы; в противном случае считается, что в системе появились регрессионные ошибки. Одна из стратегий регрессионного тестирования допускает повторный запуск всех этих тестов, но такой метод потребляет чрезмерно много времени и ресурсов.

Альтернативный метод - выборочное повторное тестирование: выбирает из старого набора тесты, которые считает необходимыми для тестирования изменённой программы. Методы отбора регрессионных тестов основаны на субъективном выборе подмножества из существующего набора тестов. Если программа не тестируется адекватно существующим набором тестов, то маловероятно, что подмножество этого набора будет адекватным для тестирования изменённой версии программы.

II. ПРЕДЛАГАЕМЫЙ МЕТОД ОТБОРА ТЕСТОВ

Для исследования предлагается простой метод выборочного регрессионного тестирования. При выборе подмножества тестов планируется использовать информацию о покрытии тестами исходного кода тестируемой системы и исключать тесты, затрагивающие только участки кода, не изменённые по отношению к предыдущей версии. Для этого необходимо:

С помощью анализа профиля программы для каждого «старого» теста собирать информацию о том, какие элементы (строки, функции и т.п.) исходного кода продукта получают управление в ходе выполнения этого теста; хранить и обновлять эту информацию для каждой версии продукта.

При передаче очередной версии продукта на тестирование с помощью системы контроля версий создавать множество добавленных, удалённых и изменённых строк исходного кода ДР.

Повторно прогонять только такие тесты, для которых во множестве покрываемых ими элементов найдётся элемент, затронутый изменением.

Упорядочить созданное множество. Наибольшую вероятность обнаружить ошибку будут иметь те тесты, у которых количество изменённых строк в пути исполнения наибольшее.

Чтобы учесть последствия изменения неисполняемых операторов, предлагается использовать критерий покрытия точек использования неисполняемых определений. Если какое-либо неисполняемое определение изменилось по сравнению с предыдущей версией программы, следует считать изменившимися все элементы программы, содержащие его использование.

Будем считать, что тестирование программы P по некоторому критерию C заключается в покрытии множества M покрываемых элементов m_j, соответствующих элементам и/или взаимосвязям между элементами некоторой модели программы:

$M(P, C) = \{m_1, m_2 \dots m_к\}$.

Пусть выполняемые тесты t_j составляют упорядоченное множество тестов $T = \{t_1, t_2, \dots, t_n\}$. Каждому тесту t_j соответствует некоторое подмножество покрываемых им элементов $MT(P, C, t_j)$. В этих условиях методика решения задачи выборочного регрессионного тестирования путём покрытия точек использования неисполняемых определений включает следующие этапы:

1) С помощью анализа профиля программы для каждого существующего теста t_j собирается информация о том, какие строки исходного кода продукта покрываются этим тестом, то есть создаётся множество MT(P, C, tj).

2) Множество $MT(P, C, t_j)$ хранится под управлением системы контроля версий (например, CVS) и обновляется для каждой версии продукта.

3) Когда очередная версия продукта передаётся на тестирование, с помощью системы контроля версий создаётся множество добавленных, удалённых и изменённых строк исходного кода (множество AP):

$$P = (P' \setminus P) \quad (P \setminus P')$$

4) Множество Δ P расширяется за счёт раскрытия макроопределений и других подобных им неисполняемых операторов.

5) Для повторного прогона выбираются только такие тесты t_j, для которых во множестве покрываемых ими элементов множества $MT(P, C, t_j)$ найдётся затронутый изменением элемент m_k. Такие тесты t_j образуют множество T':

$$T' = \{t_j \mid MT(P, C, t_j)\ P\ \}$$

6) Множество T' можно упорядочить - наибольшую вероятность обнаружить ошибку будут иметь те тесты, у которых количество изменённых строк в пути исполнения наибольшее.

Рассмотрим пример применения предложенной методики для тестирования большой промышленной программы. После проведения 3 циклов тестирования в программе было найдено некоторое количество ошибок. Изменения ДР, связанные с исправлением этих ошибок, локализованы в 3 покрываемых сущностях (модулях) программы, обозначаемых A, B и C; общее количество модулей в программе значительно больше трёх. Тестовый набор программы состоит из N тестов, где N — большое число. Матрица покрытия приводится в таблице 2,1. Покрытие тестами сущностей обозначено знаком '*'; так, тест 1 покрывает сущности A и B, но не покрывает сущностей C, D и Z. Столбцы, соответствующие изменившимся по сравнению с предыдущей версией сущностям, выделены цветом. Из таблицы видно, что сущности A, B и C изменились, а сущности D и Z остались неизменными.

	Сущ-ность A	Сущ-ность B	Сущ-ность C	Сущ-ность D	…	Сущ-ность Z
Тест 1	*	*			…	
Тест 2			*	*	…	
Тест 3	*			*	…	
Тест 4				*	…	*
…	…	…	…	…	…	…
Тест N					…	*

Рис.1 Пример применения метода расширенного регрессионного тестирования

На первом этапе вычисляется функция предсказания целесообразности по правилу:

$$N_M = N^C_M = \frac{\text{общее число знаков '*' в таблице}}{\text{общее число сущностей (ZZ)}} = 2$$

Согласно прогнозу, для исполнения будет отобрано 2 теста. Предположим, что эта величина ниже порога целесообразности, и применение выборочного метода имеет смысл. Для повторного исполнения будут выбраны все тесты, покрывающие изменённые сущности, а именно тесты 1, 2 и 3. Кроме того, они будут упорядочены: тест 1, покрывающий две изменённые сущности, будет расположен выше теста 3, который покрывает одну изменённую сущность, несмотря на то, что общее количество сущностей, покрываемых тестом 3, больше (три против двух). Отметим также, что тест 1 и тест 2 вместе обеспечивают однократное покрытие всех изменённых сущностей; следовательно, они могут составить набор для тестирования по методу минимизации. В таблице на рисунке 1 наборы для безопасного тестирования и для тестирования по методу минимизации выделены цветом.

В результате выполнения выбранных тестов, анализа их заключительных состояний и порождения и выполнения новых тестов в программе были дополнительно найдены 3 ошибки.

III. ОЦЕНКА МЕТОДА

Для работы вышеописанного метода отбора тестов для проведения регрессионного тестирования требуется сохранять в памяти массив, содержащий данные о прохождении всех тестов на предыдущей версии программы. Размер этого массива пропорционален размеру программы и количеству тестов в наборе и составляет 0(|T| * n).

Таковы требования метода по памяти. Время выполнения метода пропорционально времени просмотра вышеуказанного массива (это самая трудоёмкая операция метода) и также составляет 0(|T| * n). Эта оценка позволяет утверждать, что данный метод пригоден для регрессионного тестирования промышленных программ.

[1] Канер С., Фолк Д., Нгуен Е.К. Тестирование программного обеспечения // Киев, ДиаСофт, 2000.

[2] Котляров В. П., Епифанов Н. А., Некрасов А. О., Коликова Т. В. Основы современного тестирования программного обеспечения, разработанного на C# // СПб., СПбГТУ, Нестор, 2003,86 с.

[3] Котляров В. П., Пинаев Д. В. Методы и средства автоматизации тестирования программного проекта // СПб., СПбГТУ, 1998.

[4] Кречетов Н. Автоматизированное тестирование приложений клиент/сервер // Сети, 1996, №33.

[5] Майерс Г. Д. Надёжность программного обеспечения // М., Мир, 1980.

[6] Пучкин Р. Тестирование прикладного ПО с графическим интерфейсом // Корпоративные системы, 1999, №11.