XXX Юбилейная Неделя науки СПбГТУ. Материалы межвузовской научной конференции. Ч. VII: С.18-19, 2002.

УДК 681.324.

В.В. Швецов (6 курс, каф. ИУС), Н.А. Епифанов, асп., В.П. Котляров, проф.

СИСТЕМА ОПТИМИЗАЦИИ РЕГРЕССИОННОГО ТЕСТИРОВАНИЯ ПО ЧАСТИЧНЫМ ТЕСТОВЫМ НАБОРАМ

Средний срок эксплуатации современного программного продукта – несколько лет. За это время система морально устаревает, и команда разработчиков вынуждена «с нуля» создавать что-то новое. Продукт, эксплуатирующийся 5 и более лет – большой успех. Однако и у таких проектов есть свои сложности: для поддержания их жизнеспособности требуется постоянный выпуск новых версий. Изменения в коде могут быть незначительны, составляя 0.001%, а иногда и 0.0001%, но частота выпуска новых версий может составлять от раза в месяц до раза в неделю и чаще.

Одновременно с развитием продукта растёт и парк тестов для него. За годы эксплуатации число тестов увеличивается в несколько раз, так как для проверки новой функциональности каждый раз требуются всё новые тесты. Возникает противоречие между объёмом тестов и, соответственно, временем, необходимым для их прогона, и временем, отведённым для выпуска новой версии (как указано выше, иногда не более недели). На практике для каждой новой версии прогоняются все новые тесты для проверки новой функциональности и избранное подмножество регрессионных тестов для проверки сохранения старой функциональности. Выбор подмножества регрессионных тестов обычно осуществляется методом экспертных оценок, что не является оптимальным ни по критерию качества продукта (могут оставаться ошибки, которые были бы выявлены более полным набором тестов), ни по критерию затраченного времени («лишние» тесты, затрагивающие участки кода, не изменявшиеся с момента выпуска предыдущей версии). Таким образом, возникает задача минимизации частичных тестовых наборов, необходимых для тестирования.

Предлагаемый метод решения этой задачи состоит в следующем. С помощью средств профилирования для каждого «старого» теста i собирается информация о том, какие строки исходного кода продукта получают управление в ходе выполнения этого теста (множество T_i). Эта информация хранится и обновляется для каждой версии продукта. Когда очередная версия продукта передаётся на тестирование, с помощью системы контроля версий создаётся множество добавленных, удалённых и изменённых строк исходного кода (множество Δ). Будем считать, что тестирование программы P по некоторому критерию C заключается в покрытии множества M требуемых элементов m_j, соответствующих элементам и/или взаимосвязям между элементами некоторой модели программы: $M(P, C) = \{m_1, m_2, \dots, m_k\}$. 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 требуемых элементов $MT(P, C, t_j)$. В таком случае с точки зрения критерия C и соответствующего ему множества M(P, C) повторно следует прогонять только такие тесты t_j, для которых во множестве покрываемых ими элементов $MT(P, C, t_j)$ найдётся затронутый изменением элемент m_k. Такие тесты t_j образуют множество I. Это множество можно упорядочить – наибольшую вероятность обнаружить ошибку будут иметь те тесты, у которых количество изменённых строк в пути исполнения наибольшее.

Следует учитывать, что к множеству Δ относятся также и строки, собственно не изменявшиеся, но содержащие ссылки на модифицированные макроопределения и переменные.

Предлагаемый подход позволяет не только уменьшить количество регрессионных тестов, но и указать, какого рода новые тесты с наибольшей вероятностью обнаружат ошибки. Для этого рассмотрим состояния системы перед запуском теста и после его окончания. Эти

состояния характеризуются значением некоторого подмножества глобальных и локальных переменных. Предположим, что мы располагаем информацией об этих состояниях для каждого теста по результатам запуска на предыдущей версии продукта j-1. Множество таких состояний обозначим S_{j-1}. Теперь прогоним новые и регрессионные тесты на текущей версии продукта j и получим множество S_j. Тогда множество $N_j = S_j \backslash S_{j-1}$, т.е. множество новых по сравнению с предыдущими версиями состояний, является «подозрительным» с точки зрения наличия ошибок. Из множества N_j необходимо исключить состояния, в которых дальнейшая работа продукта невозможна по спецификации, т.е. не рассматривать тесты, проверяющие нештатные режимы работы продукта. Для каждого состояния урезанного таким образом множества N_j выявляется вектор его отличия от исходного состояния s_0, т.е. переменные, изменённые по сравнению с s_0. С учётом этой информации модифицируется множество изменённых строк исходного кода Δ, и цикл работы системы повторяется. В ситуациях, когда за заданное число циклов состояние $N_j = \varnothing$ не достигнуто, необходима некоторая эвристическая оценка, которая должна быть сделана человеком. По результатам этой оценки возможно прекращение тестирования или пропуск какого-то числа циклов.

Следует отметить, что при такой организации тестирования, когда новые тесты получаются в результате суперпозиции уже имеющихся, целесообразно в качестве исходных тестов, т.е. тех «кирпичиков», из которых будут строиться тесты в дальнейшем, брать тесты, заключающие всего одну элементарную проверку. Это поможет избежать избыточности при многократном слиянии тестов, когда искомые «подозрительные» ситуации возникают в ходе работы теста, но не анализируются, т.к. не повторяются при его завершении.

Можно предвидеть несколько проблем, которые возникнут при практической реализации предложенного подхода. Одна из них состоит в том, что тесты прогоняются на изменённой версии исходного кода, т.е. тестируется фактически не продукт, а некоторая его модификация. Путей решения этой проблемы может быть несколько. Отсутствие влияния вставленных контрольных печатей на функциональность продукта может быть доказано путём проведения обзора исходного кода, правда, такой подход неприменим для встроенных приложений и приложений реального времени. Проведение предложенной процедуры можно совместить с измерением покрытия исходного кода тестами в том случае, если эта операция проводится. Наконец, операции по сбору необходимых данных могут выполняться на фазах жизненного цикла продукта, когда эти операции не задерживают выход очередной версии, например, на этапе сбора требований заказчика.

Выводы. На данный момент система полностью разработана и проверена на небольшом множестве тестовых примеров. Предполагается использовать данную систему для более сложных проектов.