Лекция 2. Регрессионное тестирование

Цели и задачи регрессионного тестирования

При корректировках программы необходимо гарантировать сохранение качества. Для этого используется *регрессионное тестирование* – дорогостоящая, но необходимая деятельность в рамках этапа сопровождения, направленная на перепроверку корректности измененной программы. В соответствии со стандартным определением, регрессионное тестирование – это выборочное тестирование, позволяющее убедиться, что изменения не вызвали нежелательных побочных эффектов, или что измененная система по-прежнему соответствует требованиям.

Главной задачей этапа сопровождения является реализация систематического процесса обработки изменений в коде. После каждой модификации программы необходимо удостовериться, что на функциональность программы не оказал влияния модифицированный код. Если такое влияние обнаружено, говорят о *регрессионном дефекте.* Для регрессионного тестирования функциональных возможностей, изменение которых не планировалось, используются ранее разработанные тесты. Одна из целей регрессионного тестирования состоит в том, чтобы, в соответствии с используемым критерием покрытия кода (например, критерием покрытия потока операторов или потока данных), гарантировать тот же уровень покрытия, что и при полном повторном тестировании программы. Для этого необходимо запускать тесты, относящиеся к измененным областям кода или функциональным возможностям.

Пусть $T = \{t_1, t_2, \ldots, t_N\}$ – множество из *N* тестов, используемое при первичной разработке программы *P*, а $T' \subseteq T$ - подмножество регрессионных тестов для тестирования новой версии программы *P*'. Информация о покрытии кода, обеспечиваемом *T'*, позволяет указать блоки *P'*, требующие дополнительного тестирования, для чего может потребоваться повторный запуск некоторых тестов из множества $T \supseteq T'$, или даже создание *T''* – набора новых тестов для *P'* – и обновление *T*.

Другая цель регрессионного тестирования состоит в том, чтобы удостовериться, что программа функционирует в соответствии со своей спецификацией, и что изменения не привели к внесению новых ошибок в ранее протестированный код. Эта цель всегда может быть достигнута повторным выполнением всех тестов регрессионного набора, но более перспективно отсеивать тесты, на которых выходные данные модифицированной и старой программы не могут различаться. Результаты сравнения выборочных методов и метода повторного прогона всех тестов приведены в табл. 11.1.

Таблица 11.1. Выборочное регрессионное тестирование и повторный прогон всех тестов

Повторный прогон всех тестов	Выборочное регрессионное тестирование
Прост в реализации	Требует дополнительных расходов при внедрении
Дорогостоящий и неэффективный	Способно уменьшать расходы за счет исключения лишних тестов

Обнаруживает все ошибки, которые были бы найдены при исходном тестировании	Может приводить к пропуску ошибок

Задача отбора тестов из набора T для заданной программы P и измененной версии этой программы P' состоит в выборе подмножества $T'_{идеальное} \subseteq T$ для повторного запуска на измененной программе P', где $T'_{идеальное} = \{t \in T \mid P'(t) \neq P(t)\}$. Так как выходные данные P и P' для тестов из множества T и $T'_{идеальное}$ заведомо одинаковы, нет необходимости выполнять ни один из этих тестов на P'. В общем случае, в отсутствие динамической информации о выполнении P и P' не существует методики вычисления множества $T'_{идеальное}$ для произвольных множеств P, P' и T. Это следует из отсутствия общего решения проблемы останова, состоящей в невозможности создания в общем случае алгоритма, дающего ответ на вопрос, завершается ли когда-либо произвольная программа P для заданных значений входных данных. На практике создание $T'_{идеальное}$ возможно только путем выполнения на инструментированной версии P' каждого регрессионного теста, чего и хочется избежать.

Реалистичный вариант решения задачи выборочного регрессионного тестирования состоит в получении полезной информации по результатам выполнения P и объединения этой информации с данными статического анализа для получения множества $T'_{реальное}$, в виде аппроксимации $T'_{идеальное}$. Этот подход применяется во всех известных выборочных методах регрессионного тестирования, основанных на анализе кода. Множество $T'_{реальное}$ должно включать все тесты из T, активирующие измененный код, и не включать никаких других тестов, то есть тест $t \in T$ входит в $T'_{реальное}$ тогда и только тогда, когда t задействует код P в точке, где в P' код был удален или изменен, или где был добавлен новый код.

Если некоторый тест t задействует в P тот же код, что и в P', выходные данные P и P' для t различаться не будут. Из этого следует, что если $P(t) \neq P'(t)$, t должен задействовать некоторый код, измененный в P' по отношению к P, то есть должно выполняться отношение $t \in T'_{реальное}$. С другой стороны, поскольку не каждое выполнение измененного кода отражается на выходных значениях теста, могут существовать некоторые такие $t \in T'_{реальное}$, что $P(t) \neq P'(t)$. Таким образом, $T'_{реальное}$ содержит $T'_{идеальное}$ целиком и может использоваться в качестве его альтернативы без ущерба для качества тестируемого программного продукта.

Важной задачей регрессионного тестирования является также уменьшение стоимости и сокращение времени выполнения тестов.

Рассмотрим отбор тестов на примере рис. 11.1. Код, покрываемый тестами, выделен цветом и штриховкой. Легко заметить, что код, покрываемый тестом 1, не изменился с предыдущей версии, следовательно, повторное выполнение теста 1 не требуется. Напротив, код, покрываемый тестами 2, 3 и 4, изменился; следовательно, необходим их повторный запуск.

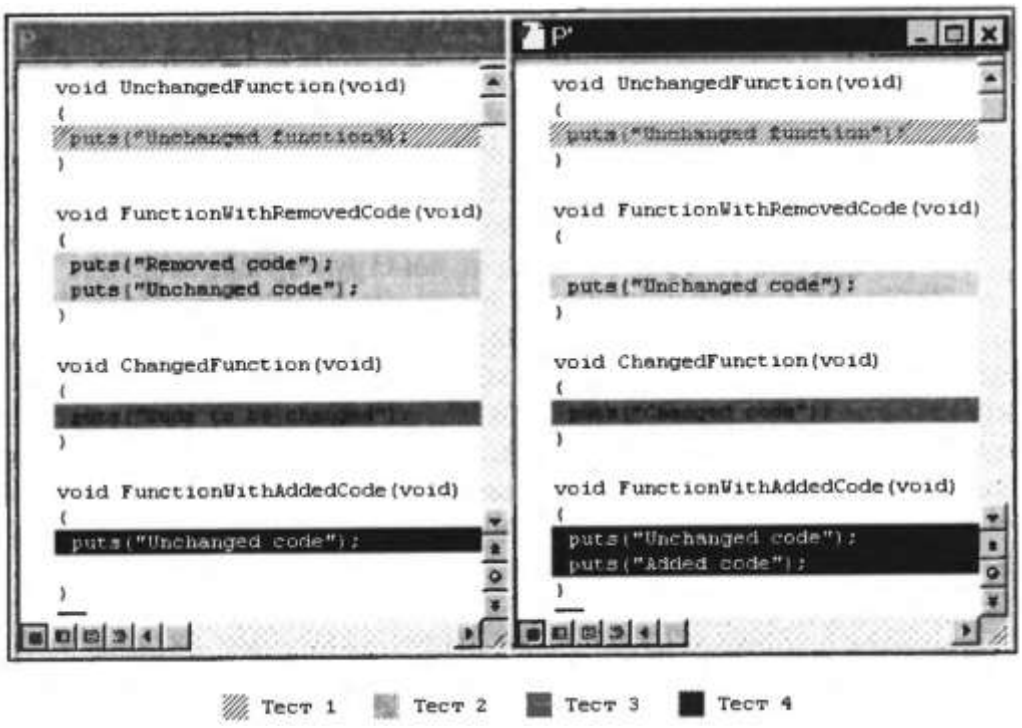

Рис. 11.1. Отбор тестов для множества T'

Виды регрессионного тестирования

Поскольку регрессионное тестирование представляет собой повторное проведение цикла обычного тестирования, виды регрессионного тестирования совпадают с видами обычного тестирования. Можно говорить, например, о модульном регрессионном тестировании или о функциональном регрессионном тестировании.

Другой способ классификации видов регрессионного тестирования связывает их с типами сопровождения, которые, в свою очередь, определяются типами модификаций. Выделяют три типа сопровождения:

- *Корректирующее сопровождение,* называемое обычно исправлением ошибок, выполняется в ответ на обнаружение ошибки, не требующей изменения спецификации требований. При корректирующем сопровождении производится диагностика и корректировка дефектов в программном обеспечении с целью поддержания системы в работоспособном состоянии.
- *Адаптивное сопровождение* осуществляется в ответ на требования изменения данных или среды исполнения. Оно применяется, когда существующая система улучшается или расширяется, а спецификация требований изменяется с целью реализации новых функций.
- *Усовершенствующее (прогрессивное) сопровождение* включает любую обработку с целью повышения эффективности работы системы или эффективности ее сопровождения.

В процессе адаптивного или усовершенствующего сопровождения обычно вводятся новые модули. Чтобы отобразить то или иное усовершенствование или адаптацию, изменяется

спецификация системы. При корректирующем сопровождении, как правило, спецификация не изменяется, и новые модули не вводятся. Модификация программы на фазе разработки подобна модификации при корректирующем сопровождении, так как из-за обнаружения ошибки вряд ли требуется менять спецификацию программы. За исключением редких моментов крупных изменений, на фазе сопровождения изменения системы обычно невелики и производятся с целью устранения проблем или постепенного расширения функциональных возможностей.

Соответственно, определяют два типа регрессионного тестирования: прогрессивное и корректирующее:

- Прогрессивное регрессионное тестирование предполагает модификацию технического задания. В большинстве случаев при этом к системе программного обеспечения добавляются новые модули.
- При корректирующем регрессионном тестировании техническое задание не изменяется. Модифицируются только некоторые операторы программы и, возможно, конструкторские решения.

Прогрессивное регрессионное тестирование обычно выполняется после адаптивного или усовершенствующего сопровождения, тогда как корректирующее регрессионное тестирование выполняется во время тестирования в цикле разработки и после корректирующего сопровождения, то есть после того, как над программным обеспечением были выполнены некоторые корректирующие действия. Вообще говоря, корректирующее регрессионное тестирование должно быть более простым, чем прогрессивное регрессионное тестирование, поскольку допускает повторное использование большего количества тестов.

Подход к отбору регрессионных тестов может быть активным или консервативным. Активный подход во главу угла ставит уменьшение объема регрессионного тестирования и пренебрегает риском пропустить дефекты. Активный подход применяется для тестирования систем с высокой исходной надежностью, а также в случаях, когда эффект изменений невелик. Консервативный подход требует отбора всех тестов, которые с ненулевой вероятностью могут обнаруживать дефекты. Этот подход позволяет обнаруживать большее количество ошибок, но приводит к созданию более обширных наборов регрессионных тестов.

Управляемое регрессионное тестирование

В течение жизненного цикла программы период сопровождения длится долго. Когда измененная программа тестируется набором тестов T, мы сохраняем без изменений по отношению к тестированию исходной программы P все факторы, которые могли бы воздействовать на вывод программы. Поэтому атрибуты конфигурации, в которой программа тестировалась последний раз (например, план тестирования, тесты t_j и покрываемые элементы $MT(P, C, t_j)$, подлежат управлению конфигурацией. Практика тестирования измененной версии программы P' в тех же условиях, в которых тестировалась исходная программа P, называется управляемым регрессионным тестированием. При неуправляемом регрессионном тестировании некоторые свойства методов регрессионного тестирования могут изменяться, например,

безопасный метод отбора тестов может перестать быть безопасным. В свою очередь, для обеспечения управляемости регрессионного тестирования необходимо выполнение ряда условий:

- Как при модульном, так и при интеграционном регрессионном тестировании в качестве модулей, вызываемых тестируемым модулем непосредственно или косвенно, должны использоваться реальные модули системы. Это легко осуществить, поскольку на этапе регрессионного тестирования все модули доступны в завершенном виде.
- Информация об изменениях корректна. Информация об изменениях указывает на измененные модули и разделы спецификации требований, не подразумевая при этом корректность самих изменений. Кроме того, при изменении спецификации требований необходимо усиленное регрессионное тестирование изменившихся функций этой спецификации, а также всех функций, которые могли быть затронуты по неосторожности. Единственным случаем когда мы вынуждены положиться на правильность измененного технического задания, является изменение технического задания для всей системы или для модуля верхнего (в графе вызовов) уровня, при условии, что кроме технического задания, не существует никакой дополнительной документации и/или какой-либо другой информации, по которой можно было бы судить об ошибке в техническом задании.
- В программе нет ошибок, кроме тех, которые могли возникнуть из-за ее изменения.
- Тесты, применявшиеся для тестирования предыдущих версий программного продукта, доступны, при этом протокол прогона тестов состоит из входных данных, выходных данных и траектории. Траектория представляет собой путь в управляющем графе программы, прохождение которого вызывается использованием некоторого набора входных данных. Ее можно применять для оценки структурного покрытия, обеспечиваемого набором тестов.
- Для проведения регрессионного тестирования с использованием существующего набора тестов необходимо хранить информацию о результатах выполнения тестов на предыдущих этапах тестирования.

Предположим, что никакие операторы программы, кроме тех, чье поведение зависит от изменений, не могут неблагоприятно воздействовать на программу. Даже при таком условии существуют некоторые ситуации, требующие особого внимания, например, проблема утечки памяти и ей подобные. Ситуации такого рода в разных системах программирования обрабатываются по-разному. Например, язык Java сам по себе включает систему управления памятью. Если же система не контролирует распределение памяти автоматически, мы должны считать, что все операторы работы с памятью также обладают поведением, зависящим от изменений.

Проблема языков типа C и C++, которые допускают произвольные арифметические операции над указателями, состоит в том, что указатели могут нарушать границы областей памяти, на которые они указывают. Это означает, что переменные могут обрабатываться

способами, которые не поддаются анализу на уровне исходного кода. Чтобы учесть такие нарушения границ памяти, выдвигаются следующие гипотезы:

- *Гипотеза 1* (Четко определенная память). Каждый сегмент памяти, к которому обращается система программного обеспечения, соответствует некоторой символически определенной переменной.
- *Гипотеза 2* (Строго ограниченный указатель). Каждая переменная или выражение, используемое как указатель, должно ссылаться на некоторую базовую переменную и ограничиваться использованием сегмента памяти, определяемого этой переменной.

Чтобы гарантировать покрытие всех зависящих от изменений компонентов, для которых можно показать, что они затрагиваются существующими тестами, достаточно одного теста для каждого из таких компонентов. Множество тестов достаточно большого размера (как правило сценарных), может способствовать обнаружению ошибок, вызванных нарушениями условий управляемого регрессионного тестирования.

Существуют и организационные условия проведения регрессионного тестирования. Это ресурс (время), необходимый тестовому аналитику для ознакомления со спецификацией требований системы, ее архитектурой и, возможно, самим кодом.

Обоснование корректности метода отбора тестов

Перечислим некоторые особенности реализации регрессионного тестирования:

- Некоторые участки кода программы не получают управление при выполнении некоторых тестов.
- Если участок кода реализует требование, но измененный фрагмент кода не получает управления при выполнении теста, то он и не может воздействовать на значения выходных данных программы при выполнении данного теста.
- Даже если участок кода, реализующий требование, получает управление при выполнении теста, это далеко не всегда отражается на выходных данных программы при выполнении данного теста. Действительно, если изменяется первый блок программы, например, путем добавления инициализации переменной, все пути в программе также изменяются, и, как следствие, требуют повторного тестирования. Однако может так случиться, что только на небольшом подмножестве путей действительно используется эта инициализированная переменная.
- Не каждый тест $t_k \in T$, проверяющий код, находящийся на одном пути с измененным кодом, обязательно покрывает этот измененный код.
- Код, находящийся на одном пути с измененным кодом, может не воздействовать на значения выходных данных измененных модулей программы.
- Не всегда каждый оператор программы воздействует на каждый элемент ее выходных данных.

Предположим, что изменения в программе ограничиваются одним оператором. Если при выполнении какого-либо теста на исходной программе этот оператор никогда не получает

управление, можно с уверенностью сказать, что он не получит управление и в ходе выполнения теста на новой программе, а результаты тестирования новой и старой программ будут совпадать. Следовательно, нет необходимости выполнять этот тест на новой программе. Указанный метод легко можно обобщить для случая нескольких изменений: если тест не задействует ни одного измененного оператора, и его входные данные не изменились, код, выполняемый им в измененной программе, будет в точности таким же, как в первоначальной версии. Такой тест не выявляет различий между двумя версиями системы; следовательно, нет необходимости прогонять его повторно. Если тест не затрагивает ни одного оператора вывода, поведение которого зависит от измененных операторов, это означает, что, несмотря на изменения в программе, все операторы, которые получают управление при выполнении этого теста, не изменят вывод системы по отношению к предыдущей версии. Таким образом, нет необходимости повторно прогонять и тесты такого рода.

Следовательно, необходимо ориентироваться на выбор только тех тестов, которые покрывают измененный код, воздействующий, в свою очередь, на вывод программы. Такой подход гарантирует, что будут выбраны только тесты, обнаруживающие изменения, и метод будет, как говорят, точным.

Классификация тестов при отборе

Создание наборов регрессионных тестов рекомендуется начинать с множества исходных тестов. При заданном критерии регрессионного тестирования все исходные тесты $t(t \in T)$ подразделяются на три подмножества:

1. *Множество тестов, пригодных для повторного использования.* Это тесты, которые уже запускались и пригодны к использованию, но затрагивают только покрываемые элементы программы, не претерпевшие изменений. При повторном выполнении выходные данные таких тестов совпадут с выходными данными, полученными на исходной программе. Следовательно, такие тесты не требуют перезапуска.
2. *Множество тестов, требующих повторного запуска.* К ним относятся тесты, которые уже запускались, но требуют перезапуска, поскольку затрагивают, по крайней мере, один измененный покрываемый элемент, подлежащий повторному тестированию. При повторном выполнении такие тесты могут давать результат, отличный от результата, показанного на исходной программе. Множество тестов, требующих повторного запуска, обеспечивает хорошее покрытие структурных элементов даже при наличии новых функциональных возможностей.
3. *Множество устаревших тестов.* Это тесты, более не применимые к измененной программе и непригодные для дальнейшего тестирования, поскольку они затрагивают только покрываемые элементы, которые были удалены при изменении программы. Их можно удалить из набора регрессионных тестов.
4. *Новые тесты,* которые еще не запускались и могут быть использованы для тестирования.

Рис. 11.2 дает представление о жизненном цикле теста. Сразу после создания тест вводится в структуру базы данных как новый. После исполнения новый тест переходит в

категорию тестов, пригодных для повторного использования либо устаревших. Если выполнение теста способствовало увеличению текущей степени покрытия кода, тест помечается как пригодный для повторного использования. В противном случае он помечается как устаревший и отбрасывается. Существующие тесты, повторно запущенные после внесения изменения в код, также классифицируются заново как пригодные для повторного использования или устаревшие в зависимости от тестовых траекторий и используемого критерия тестирования.

Классификация тестов по отношению к изменениям в коде требует анализа последствий изменений. Тесты, активирующие код, затронутый изменениями, могут требовать повторного запуска или оказаться устаревшими. Чтобы тест был включен в класс тестов, требующих повторного запуска, он должен быть затронут изменениями в коде, а также должен способствовать увеличению степени покрытия измененного кода по используемому критерию. Затронутым элементом теста может быть траектория, выходные значения, или и то, и другое. Чтобы тест был включен в класс тестов, пригодных для повторного использования, он должен вносить вклад в увеличение степени покрытия кода и не требовать повторного запуска.

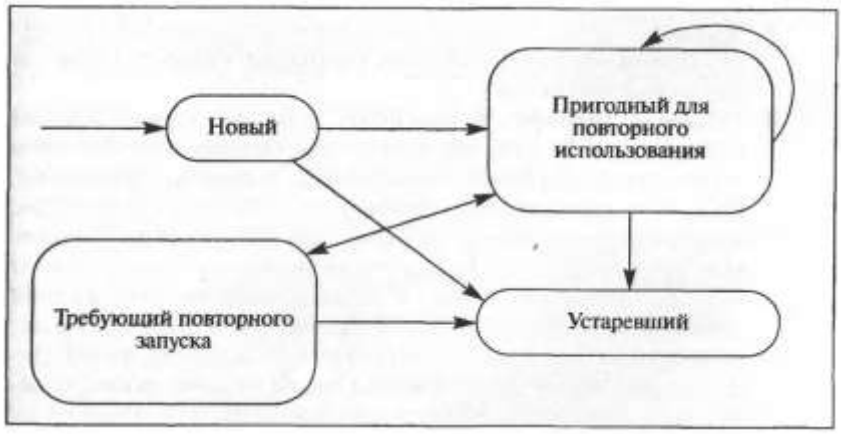

Рис. 11.2. Жизненный цикл теста

Степень покрытия кода определяется для тестов, пригодных для повторного использования, поскольку к этому классу относятся тесты, не требующие повторного запуска и способствующие увеличению степени покрытия до желаемой величины. Если имеется компонент программы, не задействованный пригодными для повторного использования тестами, то вместо них выбираются и выполняются с целью увеличения степени покрытия тесты, требующие повторного запуска. После запуска такой тест становится пригодным для повторного использования или устаревшим. Если тестов, требующих повторного запуска, больше не осталось, а необходимая степень покрытия кода еще не достигнута, порождаются дополнительные тесты и тестирование повторяется.

Окончательный набор тестов собирается из тестов, пригодных для повторного использования, тестов, требующих повторного запуска, и новых тестов. Наконец, устаревшие и избыточные тесты удаляются из набора тестов, поскольку избыточные тесты не проверяют новые функциональные возможности и не увеличивают покрытие.

Возможности повторного использования тестов

К изменению существующих тестов могут привести три вида деятельности программистов:

- Создание новых тестов.
- Выполнение тестов.
- Изменение кода.

Поскольку каждый тест содержит входные данные, выходные данные и траекторию, эти компоненты могут подвергнуться изменению в любой комбинации. При изменении входных данных существующего теста будем считать, что старый тест прекращает существование, и создается новый тест. Таким образом, к числу разрешенных изменений теста относятся всевозможные пертурбации выходных данных или траекторий. Изменение выходных данных без изменения траектории и/или входных данных невозможно. Следовательно, существует только два возможных варианта изменения теста: изменение траектории или изменение траектории и выходных данных.

В соответствии с приведенными выше рассуждениями можно выделить четыре уровня повторного использования теста:

- *Уровень 1:* Тест не допускает повторного использования. Требуется создание нового набора тестов (например, путем удаления или изменения этого теста).
- *Уровень 2:* Повторно использовать возможно только входные данные теста. Во многих случаях цель тестирования состоит в активизации некоторых покрываемых элементов программы. Если из траектории существующего теста видно, что элементы программы, подлежащие покрытию, задействуются до измененных команд, входные данные теста могут быть использованы повторно для покрытия этих элементов. В результате изменений в программе и/или техническом задании новая траектория и выходные данные теста могут отличаться от результатов предыдущего выполнения. Таким образом, тесты первого уровня должны быть запущены повторно для получения новых выходных данных и траекторий.
- *Уровень 3:* Возможно повторное использование как входных, так и выходных данных теста. Очевидно, что на этом уровне обычно располагаются функциональные тесты. Если модуль подвергся только изменению кода с сохранением функциональности, возможно повторное использование существующих функциональных тестов для проверки правильности реализации. Поскольку траектория может измениться, а выходные данные – подвергнуться воздействию со стороны изменений кода, такие тесты должны быть запущены повторно, но ожидается получение идентичных результатов.
- *Уровень 4:* Наивысший уровень повторного использования теста, предусматривающий повторное использование входных данных, выходных данных и траектории теста. В этом случае на траектории теста не изменяется ни один оператор. Следовательно, в повторном запуске этих тестов необходимости нет, так как выходные данные и траектория останутся неизменными.

Пример регрессионного тестирования функции решения квадратного уравнения

Код этой функции приведен на рис. 11.3. Входными параметрами являются коэффициенты квадратного уравнения A, B и C, а также флаг Print, ненулевое значение которого указывает, что полученное решение необходимо вывести на экран. К выходным параметрам относятся X1 и X2, предназначенные для хранения *корней уравнения, и возвращаемое значение функции – дискриминант уравнения. В исходном виде функция содержит дефект, в результате чего уравнения с отрицательным дискриминантом порождают ошибку времени выполнения. В новой версии функции дефект должен быть исправлен; кроме того, необходимо реализовать запрос пользователя на изменение формата вывода решения. Код новой версии функции Equation приводится на рис. 11.4.*

```
double Equation (int Print, float A, float B, float C,
                 float& X1, float& X2)
{
 float D = B * B - 4.0 * A * C;
 if (D >= 0)
 {
 X1 = (-B + sqrt(D)) / 2.0 / A;
 X2 = (-B - sqrt(D)) / 2.0 / A;
 }
 else
 {
 X1 = -B / 2.0 / A;
 X2 = sqrt(D);
 }
 if (Print)
 printf(«Solution: %f, %f\n», X1, X2);
 return D;
}
```

Рис. 11.3. Функция Equation – исходная версия

```
double Equation(int Print, float A, float B, float C, float&
X1, float& X2)
{
 float D = B * B - 4.0 * A * C;

 if (D >= 0)
 {
 X1 = (-B + sqrt(D)) / 2.0 / A;
 X2 = (-B - sqrt(D)) / 2.0 / A;
 }
 else
 {
 X1 = -B / 2.0 / A;
 X2 = sqrt(-D);
 }
 if (Print)
 {
 if (D >= 0)
  printf(«Solution: X1 = %f, X2 = %f\n», X1, X2);
 else
  printf(«Solution: X1 = %f+%fi, X2 = %f-%fi\n», X1, X2,
X1, X2);
 }

 return D;
}
```

Рис. 11.4. Функция Equation – измененная версия

Существующие тесты для функции Equation приведены в табл. 11.2. Входные данные тестов представляют собой совокупность значений Print, A, B и C, подаваемых на вход функции. Выходными данными для теста являются значения XI и X2, возвращаемое значение функции, а также строка, выводимая на экран; в табл. 11.2 приведены ожидаемые значения выходных данных. Кроме того, для каждого теста вычисляется траектория его прохождения по коду.

При изменении функции Equation от рис. 11.3 к рис. 11.4 меняется формат выводимых на экран данных, так что тесты 1 и 2, проверяющие вывод на экран, могут быть повторно использованы только на уровне 2. Тесты 3, 4 и 5 могут быть использованы на уровне 3 или 4 в зависимости от результатов анализа их траектории.

Таблица 11.2. Входные и выходные данные тестов

	Входные данные				**Ожидаемые выходные данные**			
Тест	A	B	C	Print	X1	X2	Возвращаемое значение	Выводимая строка
1	1	1	-6	1	2	-3	25	Solution: XI = 2, X2 = -3
2	2	-3	5	1	0.75	5.567764	-31	Solution: X1 = 0.75+5.567764i, X2 = 0.75-5.567764i
3	1	2	0	0	0	-2	4	
4	1	2	1	0	-1	-1	0	
5	1	2	2	0	-1	2	-4	

Классификация выборочных методов

Для проверки корректности различных подходов к регрессионному тестированию используется модель оценки методов регрессионного тестирования. Основными объектами рассмотрения стали полнота, точность, эффективность и универсальность.

Точность – мера способности метода избегать выбора тестов из T, на которых результат выполнения измененной программы не будет отличаться от результата ее первоначальной версии, то есть тестов, неспособных обнаруживать ошибки в P'. Предположим, что набор T содержит r регрессионных тестов. Из них для n тестов $(n \le r)$ поведение и результаты выполнения старой программы P отличаются от поведения и результатов выполнения новой программы P'. Набор тестов $T' \subseteq T$ содержит $m\,(m \neq 0)$ тестов, полученных с использованием метода отбора регрессионных тестов M. Из этих m тестов для l тестов поведение P' и P различается. Точность T' относительно P, P', T и M, выраженная в процентах, определяется выражением $100 \cdot l / m$, тогда как соответствующий процент выбранных тестов определяется выражением $100 \cdot l / n$, если $m \neq 0$ или равен 100%, если $n \neq 0$.

Исходя из приведенного определения, точность множества тестов – это отношение числа тестов данного множества, на которых результаты выполнения новой и старой программ различаются, к общему числу тестов множества. Точность является важным атрибутом метода регрессионного тестирования. Неточный метод имеет тенденцию отбирать тесты, которые не должны были быть выбраны. Чем менее точен метод, тем ближе объем выбранного набора тестов к объему исходного набора тестов.

Эффективность – оценка вычислительной стоимости стратегии выборочного регрессионного тестирования, то есть стоимости реализации ее требований по времени и памяти, а также возможности автоматизации. Относительной эффективностью называется эффективность метода тестирования при условии наличия не более одной ошибки в тестируемой программе. Абсолютной эффективностью называется эффективность метода в реальных условиях, когда оценка количества ошибок в программе не ограничена.

Универсальность отражает меру способности метода к применению в достаточно широком диапазоне ситуаций, встречающихся на практике.

Для программы P, ее измененной версии P' и набора тестов T для P требуется, чтобы методика выборочного повторного тестирования удовлетворяла следующим критериям оценки:

- Критерий 1 – *Безопасность.* Методика выборочного повторного тестирования должна быть безопасной, то есть должна выбирать все тесты из T, которые потенциально могут обнаруживать ошибки (все тесты, чье поведение на P' и P может быть различным). Безопасная методика должна рассматривать последствия добавления, удаления и изменения кода. При добавлении нового кода в P, в T могут уже содержаться тесты, покрывающие этот новый код. Такие тесты необходимо обнаруживать и учитывать при отборе.
- Критерий 2 – Точность. Стратегия повторного прогона всех тестов является безопасной, но неточной. В дополнение к выбору всех тестов, потенциально способных

обнаруживать ошибки, она также выбирает тесты, которые ни в коем случае не могут демонстрировать измененное поведение. В идеале, методика выборочного повторного тестирования должна быть точной, то есть должна выбирать только тесты с изменившимся поведением. Однако для произвольно взятого теста, не запуская его, невозможно определить, изменится ли его поведение. Следовательно, в лучшем случае мы можем рассчитывать лишь на некоторое увеличение точности.

Всевозможные существующие выборочные методы регрессионного тестирования различаются не в последнюю очередь выбором объекта или объектов, для которых выполняется анализ покрытия и анализ изменений. Например, при анализе на уровне функции при изменении любого оператора функции вся функция считается измененной; при анализе на уровне отдельных операторов мы можем исключить часть тестов, содержащих вызов функции, но не активирующих измененный оператор. Выбор объектов для анализа покрытия отражается на уровне подробности анализа, а значит, и на его точности и эффективности. Абсолютные величины точности и количества выбранных тестов для заданных набора тестов и множества изменений должны рассматриваться только вместе с уменьшением размера набора тестов. Небольшой процент выбранных тестов может быть приемлемым, только если уровень точности остается достаточно высоким.

- Критерий 3 – *Эффективность.* Методика выборочного повторного тестирования должна быть эффективной, то есть должна допускать автоматизацию и выполняться достаточно быстро для практического применения в условиях ограниченного времени регрессионного тестирования. Методика должна также предусматривать хранение информации о ходе исполнения тестов в минимально возможном объеме.
- Критерий 4 – *Универсальность.* Методика выборочного повторного тестирования должна быть универсальной, то есть применимой ко всем языкам и языковым конструкциям, эффективной для реальных программ и способной к обработке сколь угодно сложных изменений кода.

В общем случае существует некоторый компромисс между безопасностью, точностью и эффективностью. При отборе тестов анализ необходимо провести за время, меньшее, чем требуется для выполнения и проверки результатов тестов из T, не вошедших в T'. С учетом этого ограничения решением задачи регрессионного тестирования будет безопасный метод с хорошим балансом дешевизны и высокой точности.

Случайные методы

Когда из-за ограничений по времени использование метода повторного прогона всех тестов невозможно, а программные средства отбора тестов недоступны, инженеры, ответственные за тестирование, могут выбирать тесты случайным образом или на основании «догадок», то есть предположительного соотнесения тестов с функциональными возможностями на основании предшествующих знаний или опыта. Например, если известно, что некоторые тесты задействуют особенно важные функциональные возможности или

обнаруживали ошибки ранее, их было бы неплохо использовать также и для тестирования измененной программы. Один простой метод такого рода предусматривает случайный отбор предопределенного процента тестов из *T*. Подобные случайные методы принято обозначать *random(x)*, где *x* – процент выбираемых тестов.

Случайные методы оказываются на удивление дешевыми и эффективными. Случайно выбранные входные данные могут давать больший разброс по покрытию кода, чем входные данные, которые используются в наборах тестов, основанных на покрытии, в одних случаях дублируя покрытие, а в других не обеспечивая его. При небольших интервалах тестирования их эффективность может быть как очень высокой, так и очень низкой. Это приводит и к большему разбросу статистики отбора тестов для таких наборов. Однако при увеличении интервала тестирования этот разброс становится значительно меньше, и средняя эффективность случайных методов приближается к эффективности метода повторного прогона всех тестов с небольшими отклонениями для разных попыток. Таким образом, в последнем случае пользователь случайных методов может быть более уверен в их эффективности. Вообще, детерминированные методы эффективнее случайных методов, но намного дороже, поскольку выборочные стратегии требуют большого количества времени и ресурсов при отборе тестов.

Если изменения в новой версии затрагивают код, выполняемый относительно часто, при случайных входных данных измененный код может в среднем активироваться даже чаще, чем при выполнении тестов, основанных на покрытии кода. Это приведет к увеличению метрики количества отобранных тестов для случайных наборов. Наоборот, относительно редко выполняемый измененный код активируется случайными тестами реже, и соответствующая метрика снижается. При уменьшении мощности множества отобранных тестов падает эффективность обнаружения ошибок.

Когда выбранное подмножество, хотя и совершенное с точки зрения полноты и точности, все еще слишком дорого для регрессионного тестирования, особенно важна гибкость при отборе тестов. Какие дополнительные процедуры можно применить для дальнейшего уменьшения числа выбранных тестов? Одно из возможных решений – случайное исключение тестов. Однако, поскольку такое решение допускает произвольное удаление тестов, активирующих изменения в коде, существует высокий риск исключения всех тестов, обнаруживающих ошибку в этом коде. Тем не менее, если стоимость пропуска ошибок незначительна, а интервал тестирования велик, целесообразным будет использование случайного метода с небольшим процентом выбираемых тестов (25-30%), например, *random(25)*.

При использовании другого случайного метода – метода экспертных оценок – в данном случае наиболее вероятен выбор всех тестов, так как затраты на прогон невелики. Однако при регрессионном тестировании больших программных систем, когда повторный прогон всех тестов неприемлем, эксперт вынужден отсеивать некоторые тесты, что также может приводить к тому, что часть изменений не будет протестирована полностью.

Безопасные методы

Метод выборочного регрессионного тестирования называется безопасным, если при некоторых четко определенных условиях он не исключает тестов (из доступного набора тестов), которые обнаружили бы ошибки в измененной программе, то есть обеспечивает выбор всех тестов, обнаруживающих изменения. Тест называется обнаруживающим изменения, если его выходные данные при прогоне на P' отличаются от выходных данных при прогоне на P: $P(t) \neq P'(t)$. Тесты, активизирующие измененный код, называются выполняющими изменение.

Выбор всех выполняющих изменение тестов является безопасным, но при этом отбираются некоторые тесты, не обнаруживающие изменений. Безопасный метод может включать в T' подмножество тестов, выходные данные которых для P и P' ни при каких условиях не отличаются. Поскольку не существует методики, кроме собственно выполнения теста, позволяющей для любой P' определить, будут ли выходные данные теста различаться для P и P', ни один метод не может быть безопасным и абсолютно точным одновременно. T' является безопасным подмножеством T тогда и только тогда, когда:

$P(t) \neq P'(t) \Rightarrow t \in T'$.

Если P и P' выполняются в идентичных условиях и T' является безопасным подмножеством T, исполнение T' на P' всегда обнаруживает любые связанные с изменениями ошибки в P, которые могут быть найдены путем исполнения T. Если существует тест, обнаруживающий ошибку, безопасный метод всегда находит ее. Таким образом, ни один случайный метод не обладает такой же эффективностью обнаружения ошибок, как безопасный метод.

При некоторых условиях безопасные методы в силу определения «безопасности» гарантируют, что все «обнаруживаемые» ошибки будут найдены. Поэтому относительная эффективность всех безопасных методов равна эффективности метода повторного прогона всех тестов и составляет 100%. Однако их абсолютная эффективность падает с увеличением интервала тестирования. Отметим, что безопасный метод действительно безопасен только в предположении корректности исходного множества тестов T, то есть когда при выполнении всех $t \in T$ исходная программа P завершилась с корректными значениями выходных данных, а все устаревшие тесты были из T удалены.

Существуют программы, измененные версии и наборы тестов, для которых применение безопасного отбора не дает большого выигрыша в размере набора тестов. Характеристики исходной программы, измененной версии и набора тестов могут совместно или независимо воздействовать на результаты отбора тестов. Например, при усложнении структуры программы вероятность активации произвольным тестом произвольного изменения в программе уменьшается. Безопасный метод предпочтительнее выполнения всех тестов набора тогда и только тогда, когда стоимость анализа меньше, чем стоимость выполнения невыбранных тестов. Для некоторых систем, критичных с точки зрения безопасности, стоимость пропуска ошибки может быть настолько высока, что небезопасные методы выборочного регрессионного тестирования использовать нельзя.

Примером безопасного метода может служить метод, который выбирает из T каждый

тест, выполняющий, по крайней мере, один оператор, добавленный или измененный в *P′* или удаленный из *P*. Применение этого метода для регрессионного тестирования функции решения квадратного уравнения потребует построения матрицы покрытия, пример которой приведен в табл, 12.1. Следует отметить, что матрица покрытия соответствует исходной версии программы, поскольку аналогичная информация для новой версии программы пока не собрана. Звездочка в ячейке таблицы означает, что соответствующий тест покрывает определенную строку кода; если тест не покрывает строку кода, ячейка оставлена пустой. Строки, измененные по отношению к исходной версии, выделены цветом. Легко заметить, что в соответствии с требованиями предложенного безопасного метода для повторного выполнения должны быть отобраны тесты 1, 2 и 5.

Методы минимизации

Процедура минимизации набора тестов ставит целью отбор минимального (в терминах количества тестов) подмножества *T*, необходимого для покрытия каждого элемента программы, зависящего от изменений. Для проверки корректности программы используются только тесты из минимального подмножества.

Таблица 12.1. Матрица покрытия тестируемого кода

		Тест				
№	**Строка кода**	**1**	**2**	**3**	**4**	**5**
1	Double Equation (int Print, float A, float B, float C, float& X1, floats& X2) {	*	*	*	*	*
2	float D=B*B-4.0*A*C;	*	*	*	*	*
3	if (D >= 0) **{**	*	*	*	*	*
4	X1 = (-B + sqrt(D)) / 2.0 / A;	*		*	*	
5	X2 = (-B - sqrt(D)) / 2.0 / A; } else {	*		*	*	
6	X1 = -B / 2.0 / A;		*			*
7	X2 = sqrt (D); }		*			*
8	if (Print)	*	*	*	*	*
9	printf ("Solution: %f, %f\n", X1, X2);	*	*			

10	return D;	*	*	*	*	*
11	}	*	*	*	*	*

Обоснование применения методов минимизации состоит в следующем:

- Корреляция между эффективностью обнаружения ошибок и покрытием кода выше, чем между эффективностью обнаружения ошибок и размером множества тестов. Неэффективное тестирование, например многочасовое выполнение тестов, не увеличивающих покрытие кода, может привести к ошибочному заключению о корректности программы.
- Независимо от способа порождения исходного набора тестов, его минимальные подмножества имеют преимущество в размере и эффективности, так как состоят из меньшего количества тестов, не ослабляя при этом способности к обнаружению ошибок или снижая ее незначительно.
- Вообще говоря, сокращенный набор тестов, отобранный при минимизации, может обнаруживать ошибки, не обнаруживаемые сокращенным набором того же размера, выбранным случайным или каким-либо другим способом. Такое преимущество минимизации перед случайными методами в эффективности является закономерным. Однако из всех детерминированных методов минимизация приводит к созданию наименее эффективных наборов тестов, хотя и самых маленьких. В частности, безопасные методы эффективнее методов минимизации, хотя и намного дороже.

Минимизация набора тестов требует определенных затрат на анализ. Если стоимость этого анализа больше затрат на выполнение некоторого порогового числа тестов, существует более дешевый случайный метод, обеспечивающий такую же эффективность обнаружения ошибок.

Хотя минимальные наборы тестов могут обеспечивать структурное покрытие измененного кода, зачастую они не являются безопасными, поскольку очевидно, что некоторые тесты, потенциально способные обнаруживать ошибки, могут остаться за чертой отбора. Набор функциональных тестов обычно не обладает избыточностью в том смысле, что никакие два теста не покрывают одни и те же функциональные требования. Если тесты исходно создавались по критерию структурного покрытия, минимизация приносит плоды, но когда мы имеем дело с функциональными тестами, предпочтительнее не отбрасывать тесты, потенциально способные обнаруживать ошибки. В существующей практике тестирования инженеры предпочитают не заниматься минимизацией набора тестов.

Многие критерии покрытия кода фактически не требуют выбора минимального множества тестов. В некотором смысле, о безопасных стратегиях и стратегиях минимизации можно думать как о находящихся на двух полюсах множества стратегий. На практике, использование «почти минимальных» наборов тестов может быть удовлетворительным. Стремление к сокращению объема набора тестов основано на интуитивном предположении, что неоднократное повторное выполнение кода в ходе модульного тестирования «расточительно». Однако усилия, требуемые для минимизации набора тестов, могут быть существенны, и,

следовательно, могут не оправдывать затрат. Отметим, что большинство стратегий выборочного регрессионного тестирования, описанных в литературе, в общем-то, не зависит от критерия покрытия, возможно, использовавшегося при создании исходного набора тестов. Инженеры, занимающиеся регрессионным тестированием, часто не имеют информации о том, как разрабатывался исходный набор тестов.

Обнаружение ошибок важно для приложений, где стоимость выполнения тестов очень высока, в то время как стоимость пропуска ошибок считается незначительной. В этих условиях использование методов минимизации целесообразно, поскольку они связаны с отбором небольшого количества тестов. Примером применения методов минимизации служит метод, выбирающий из *T* не менее одного теста для каждого оператора программы, добавленного или измененного при создании *P'*. В табл. 12.1 для случая регрессионного тестирования функции Equation данный метод ограничится отбором одного теста – теста 2, так как этот тест покрывает обе измененные строки.

Методы, основанные на покрытии кода

Значение методов, основанных на покрытии кода, состоит в том, что они гарантируют сохранение выбранным набором тестов требуемой степени покрытия элементов *P'* относительно некоторого критерия структурного покрытия *C*, использовавшегося при создании первоначального набора тестов. Это не означает, что если атрибут программы, определенный *C*, покрывается первоначальным множеством тестов, он будет также покрыт и выбранным множеством; гарантируется только сохранение процента покрываемого кода. Методы, основанные на покрытии, уменьшают разброс по покрытию, требуя отбора тестов, активирующих труднодоступный код, и исключения тестов, которые только дублируют покрытие. Поскольку на практике критерии покрытия кода обычно применяются для отбора единственного теста для каждого покрываемого элемента, подходы, основанные на покрытии кода, можно рассматривать как специфический вид методов минимизации.

Разновидностью методов, основанных на покрытии кода, являются методы, которые базируются на покрытии потока данных. Эти методы эффективнее методов минимизации и почти столь же эффективны, как безопасные методы. В то же время, они могут требовать, по крайней мере, такого же времени на анализ, как и наиболее эффективные безопасные методы, и, следовательно, могут обходиться дороже безопасных методов и намного дороже других методов минимизации. Они имеют тенденцию к включению избыточных тестов в набор регрессионных тестов для покрытия зависящих от изменений пар определения-использования, что, в некоторых случаях, ведет к большому числу отобранных тестов. Этот факт зафиксирован экспериментально.

Методы, основанные на использовании потока данных, могут быть полезны и для других задач регрессионного тестирования, кроме отбора тестов, например, для нахождения элементов *P*, недостаточно тестируемых *T'*.

Метод стопроцентного покрытия измененного кода аналогичен методу минимизации. Так, для примера из табл. 12.1 существует 4 способа отобрать 2 теста в соответствии с этим

критерием. Одного теста недостаточно. Результаты сравнения методов выборочного регрессионного тестирования приведены в табл. 12.2.

Таблица 12.2. Сравнение методов выборочного регрессионного тестирования

Класс методов	Случайные	Безопасные	Минимизации	Покрытия
Полнота	От 0% до 100%	100%	< 100%	< 100%
Размер набора тестов	Настраивается	Большой	Небольшой	Зависит от параметров метода
Время выполнения метода	Пренебрежимо мало	Значительное	Значительное	Значительное
Перспективные свойства методов регрессионного тестирования	Отсутствие средства поддержки регрессионного тестирования	Высокие требования по качеству	Стоимость пропуска ошибки невелика	Набор исходных тестов создается по критерию покрытия

Регрессионное тестирование: методики, не связанные с отбором тестов и методики порождения тестов

Интеграционное регрессионное тестирование

С появлением новых направлений в разработке программного обеспечения (например, объектно-ориентированного программирования), поощряющих использование большого количества маленьких процедур, повышается важность обработки межмодульного влияния изменений кода для методик уменьшения стоимости регрессионного тестирования. Для решения этой задачи необходимо рассматривать зависимости по глобальным переменным, когда переменной в одной или нескольких процедурах присваивается значение, которое затем используется во многих других процедурах. Такую зависимость можно рассматривать как зависимость между процедурами по потоку данных. Также возможны межмодульные зависимости по ресурсам, например по памяти, когда ресурс разделяется между несколькими процедурами. Отметим, что при системном регрессионном тестировании зависимости такого рода можно игнорировать.

Если изменение спецификации требований затрагивает глобальную переменную, могут

потребоваться новые модульные тесты. В противном случае, повторному выполнению подлежат только модульные тесты, затрагивающие как измененный код, так и операторы, содержащие ссылку на глобальную переменную.

Брандмауэр можно определить как подмножество графа вызовов, содержащее измененные и зависящие от изменений процедуры и интерфейсы. Методы отбора тестов, использующие брандмауэр, требуют повторного интеграционного тестирования только тех процедур и интерфейсов, которые непосредственно вызывают или вызываются из измененных процедур.

Регрессионное тестирование объектно-ориентированных программ

Объектно-ориентированный подход стимулирует новые приложения методик выборочного повторного тестирования. Действительно, при изменении класса необходимо обнаружить в наборе тестов класса только тесты, требующие повторного выполнения. Точно так же при порождении нового класса из существующего необходимо определить тесты из множества тестов базового класса, требующие повторного выполнения на классе-потомке. Хотя благодаря инкапсуляции вероятность ошибочного взаимодействия объектно-ориентированных модулей кода уменьшается, тем не менее, возможно, что тестирование прикладных программ выявит ошибки в методах, не найденные при модульном тестировании методов. В этом случае необходимо рассмотреть все прикладные программы, использующие измененный класс, чтобы продемонстрировать, что все существующие тесты, способные обнаруживать ошибки в измененных классах, были запущены повторно, и было выбрано безопасное множество тестов. При повторном тестировании прикладных программ, классов или их наследников применение методик выборочного повторного тестирования к существующим наборам тестов может принести немалую пользу.

В объектно-ориентированной программе вызов метода во время выполнения может быть сопоставлен любому из ряда методов. Для заданного вызова мы не всегда можем статически определить метод, с которым он будет связан. Выборочные методы повторного тестирования, которые полагаются на статический анализ, должны обеспечивать механизмы для разрешения этой неопределенности.

Уменьшение объема тестируемой программы

Еще один путь сокращения затрат на регрессионное тестирование состоит в том, чтобы вместо повторного тестирования (большой) измененной программы с использованием соответственно большого числа тестов доказать, что измененная программа адекватно тестируется с помощью выполнения некоторого (меньшего) числа тестов на остаточной программе. Остаточная программа создается путем использования графа зависимости системы вместо графа потока управления, что позволяет исключить ненужные зависимости между компонентами в пределах одного пути графа потока управления. Так, корректировка какого-либо оператора в идеале должна приводить к необходимости тестировать остаточную программу, состоящую из всех операторов исходной программы, способных повлиять на этот

оператор или оказаться в сфере его влияния. Для получения остаточной программы необходимо знать место корректировки (в терминах операторов), а также информационные и управляющие связи в программе. Этот подход работает лучше всего для малых и средних изменений больших программ, где высокая стоимость регрессионного тестирования может заставить вообще отказаться от его проведения. Наличие дешевого метода остаточных программ, обеспечивающего такую же степень покрытия кода, делает регрессионное тестирование успешным даже в таких случаях.

Метод остаточных программ имеет ряд ограничений. В частности, он не работает при переносе программы на машину с другим процессором или объемом памяти. Более того, он может давать неверные результаты и на той же самой машине, если поведение программы зависит от адреса ее начальной загрузки, или если для остаточной программы требуется меньше памяти, чем для измененной, и, соответственно, на остаточной программе проходит тест, который для измененной программы вызвал бы ошибку нехватки памяти. Исследования метода на программах небольшого объема показали, что выполнение меньшего количества тестов на остаточной программе не оправдывает затрат на отбор тестов и уменьшение объема программы. Однако для программ с большими наборами тестов это не так.

Для теста 1 табл. 12.1 для функции Equation остаточная программа выглядит так, как показано в табл. 13.1. Нумерация строк оставлена такой же, как в исходной программе. Таким образом, можно заметить, что были удалены строки 6 и 7, которые не затрагиваются тестом 1 в ходе его выполнения, а также строки 3 и 8, содержащие вычисление предикатов, которые в ходе выполнения теста всегда истинны. Запуск теста на полной измененной программе и на остаточной программе приводит к активизации одних и тех же операторов, поэтому выигрыша во времени получить не удается, однако за счет сокращения объема программы уменьшается время компиляции. Для нашего примера этот выигрыш незначителен и не оправдывает затрат на анализ, необходимый для уменьшения объема. Таким образом, рассмотренная технология рекомендуется к применению, прежде всего, в случаях, когда стоимость компиляции относительно высока.

Табл. 13.1. Остаточная программа

№	Строка кода
1	double Equation (int Print, float A, float B, float C, float& X1, float& X2) {
2	float D = B*B-4.0*A*C;
4	X1 = (-B+sqrt(D)) / 2.0 / A;
5	X2 = (-B-sqrt(D)) / 2.0 / A;
9	Printa ("Solution: %f, %f\n" X1, X2);
10	return D;
11	}

Сведения о методике уменьшения объема тестируемой программы приведены в табл. 13.2.

Таблица 13.2. Результаты применения методики уменьшения объема

Характеристика	Изменение в результате применения методики
Время компиляции тестируемой программы	Уменьшается
Время выполнения тестируемой программы	Не изменяется
Время работы метода отбора	Увеличивается
Риск пропуска ошибок	Увеличивается
Результаты применения методики на практике	Отрицательные

Методы упорядочения

Методы упорядочения позволяют инженерам-тестировщикам распределить тесты так, что тесты с более высоким приоритетом исполняются раньше, чем тесты с более низким приоритетом, чтобы затем ограничиться выбором первых n тестов для повторного исполнения. Это особенно важно для случаев, когда тестировщики могут позволить себе повторное выполнение только небольшого количества регрессионных тестов.

Одной из проблем упорядочения тестов является отсутствие представлений о том, скольких тестов в конкретном проекте достаточно для отбора. В отличие от подхода минимизации, использующего все тесты минимального набора, оптимальное число тестов в упорядоченном наборе неизвестно. Возникает проблема баланса между тем, что необходимо делать в ходе регрессионного тестирования, и тем, что мы можем себе позволить. В итоге количество запускаемых тестов определяется ограничениями по времени и бюджету и порядком тестов в наборе. С учетом этих факторов следует запускать повторно как можно больше тестов, начиная с верхней строки списка упорядоченных тестов.

Возможное преимущество упорядочения тестов состоит в том, что сложность соответствующего алгоритма, $0(n^2)$ в наихудшем случае, меньше, чем сложность алгоритма минимизации, который в некоторых случаях может требовать экспоненциального времени выполнения.

Проблему упорядочения тестов можно сформулировать следующим образом:

Дано: T – набор тестов, PT – набор перестановок T, f – функция из PT на множество вещественных чисел.

Найти: набор $T' \in PT$ такой, что:

$$(\forall T'')(T'' \in PT)(T'' \neq T')\left[f(T') \geq f(T'')\right].$$

В приведенном определении PT представляет собой множество всех возможных вариантов упорядочения – T, а f – функция, которая, будучи применена к любому такому упорядочению, выдает его вес. (Предположим, что большие значения весов предпочтительнее малых.)

Упорядочение может преследовать различные цели:

– Увеличение частоты обнаружения ошибок наборами тестов, то есть увеличение вероятности обнаружить ошибку раньше при выполнении регрессионных тестов из этих наборов.

– Ускорение процесса покрытия кода тестируемой системы и достижение требуемой степени покрытия кода на более ранних этапах процесса тестирования.
– Быстрейший рост вероятности того, что тестируемая система надежна.
– Увеличение вероятности обнаружения ошибок, связанных с конкретными изменениями кода, на ранних этапах процесса тестирования и т.п.

Методы упорядочения планируют выполнение тестов в процессе регрессионного тестирования в порядке, увеличивающем их эффективность в терминах достижения заданной меры производительности. Обоснование использования упорядочивающего метода состоит в том, что упорядоченный набор тестов имеет большую вероятность достижения цели, чем тесты, расположенные по какому-либо другому правилу или в случайном порядке.

Различают два типа упорядочения тестов: общее и в зависимости от версии. Общее упорядочение тестов при данных программе *P* и наборе тестов *T* подразумевает нахождение порядка тестов *T*, который окажется полезным для тестирования нескольких последовательных измененных версий. Считается, что для этих версий итоговый упорядоченный набор тестов позволяет в среднем быстрее достигнуть цели, ради которой производилось упорядочение, чем исходный набор тестов. Однако имеется значительный объем статистических свидетельств в пользу наличия связи между частотой обнаружения ошибок и конкретной версией тестируемой программы: разные версии программы предоставляют различные возможности по упорядочению тестов. При регрессионном тестировании мы имеем дело с конкретной версией программного продукта и хотим упорядочивать тесты наиболее эффективно по отношению именно к этой версии.

Например, упорядочивать тесты можно по количеству покрываемых ими изменений кода. При безопасном отборе тестов из табл. 12.1 будут выбраны тесты 1, 2 и 5, из которых наиболее приоритетным является тест 2, так как он затрагивает оба изменения, тогда как тесты 1 и 5 – только одно.

Сведения о методике упорядочения тестов суммированы в табл. 13.3.

Таблица 13.3. Результаты применения методики упорядочения тестов

Характеристика	**Изменение в результате применения методики**
Время работы метода отбора	Увеличивается незначительно
Частота обнаружения ошибок	Увеличивается
Скорость покрытия кода	Увеличивается
Результаты применения методики на практике	Положительные

Целесообразность отбора тестов

Поскольку в общем случае оптимальный отбор тестов (то есть выбор в точности тех тестов, которые обнаруживают ошибку) невозможен, соотношение между затратами на

применение методов выборочного регрессионного тестирования и выигрышем от их использования является основным вопросом практического применения выборочного регрессионного тестирования. На основании оценки этого соотношения делается вывод о целесообразности отбора тестов.

Эффективное регрессионное тестирование представляет собой компромисс между качеством тестируемой программы и затратами на тестирование. Чем больше регрессионных тестов, тем полнее проверка правильности программы. Однако большее количество выполняемых тестов обычно означает увеличение финансовых затрат и времени на тестирование, что на практике не всегда приемлемо. Выполнение меньшего количества регрессионных тестов может оказаться дешевле, но не позволяет гарантировать сохранение качества.

Когда отдельные модули невелики и несложны, а связанные с ними наборы тестов также небольшие, простой повторный запуск всех тестов достаточно эффективен. При интеграционном тестировании это менее вероятно. В то время как тесты для отдельных модулей могут быть небольшими, тесты для групп модулей и подсистем достаточно велики, что создает предпосылки для уменьшения издержек тестирования. С другой стороны, с ростом размера приложений стоимость применения выборочной стратегии повторного тестирования может возрасти до неприемлемой величины. Затраты на необходимый для отбора анализ могут перевешивать экономию от прогона сокращенного набора тестов и анализа результатов прогона. Однако в области тестирования достаточно больших программ положительный баланс затрат и выгод вполне достижим.

Модель затрат и выгод при использовании выборочных стратегий регрессионного тестирования должна учитывать прямые и косвенные затраты. Прямые затраты включают отбор и выполнение тестов и анализ результатов. Косвенные затраты включают затраты на управление, сопровождение баз данных и разработку программных средств. Выгоды – это затраты, которых удалось избежать, не выполняя часть тестов. Чтобы метод выборочного регрессионного тестирования был эффективнее метода повторного прогона всех тестов, стоимость анализа при отборе подмножества тестов вкупе со стоимостью их выполнения и проверки результатов должна быть меньше, чем стоимость выполнения и проверки результатов исходного набора тестов.

Пусть T' – подмножество T, отобранное некоторой стратегией выборочного регрессионного тестирования M для программы P, $|T'|$ – обозначает мощность T', s – средняя стоимость отбора одного теста в результате применения M к P для создания T', а r – средняя стоимость выполнения одного теста из T на P и проверки его результата. Тогда для того, чтобы выборочное регрессионное тестирование было целесообразным, требуется выполнение неравенства:

$$s|T'| < r(|T| - |T'|).$$

Применяя вышеупомянутую модель стоимости с целью анализа затрат, полезно условно разделять регрессионное тестирование на две фазы – предварительную и критическую. *Предварительная фаза регрессионного тестирования* начинается после выпуска очередной

версии программного продукта; во время этой фазы разработчики расширяют функциональность программы и исправляют ошибки, готовясь к выпуску следующей версии. Одновременно тестировщики могут планировать будущее тестирование или выполнять задачи, требующие наличия только предыдущей версии программы, такие как сбор тестовых траекторий и анализ покрытия. Как только в программу внесены исправления, начинается *критическая фаза регрессионного тестирования.* В течение этой фазы регрессионное тестирование новой версии программы является доминирующим процессом, время которого обычно ограничено моментом поставки заказчику. Именно на критической фазе регрессионного тестирования наиболее важна минимизация затрат. При использовании выборочного метода регрессионного тестирования важно использовать факт наличия этих двух фаз, уделяя как можно больше внимания выполнению задач, связанных с анализом, в течение предварительной фазы, чтобы на критической фазе заниматься только прогоном тестов и уменьшить вероятность срыва сроков поставки. Тем не менее, важно понимать, что до внесения последнего изменения в код анализ может быть выполнен только частично.

Если не учитывать не очень больших затрат на анализ при использовании детерминированных методов, решение о применении конкретного метода отбора тестов будет зависеть от отношения стоимости выполнения большего количества тестов к цене пропуска ошибки, что зависит от множества факторов, специфических для каждого конкретного случая. При отсутствии ошибок сбережения пропорциональны уменьшению размера набора тестов и могут быть измерены в терминах процента выбранных тестов, |T'|/|T|.

Модели стоимости могут использоваться как при выборе наилучшей стратегии, так и для оценки пригодности конкретной стратегии. При анализе учитываются такие факторы как размер программы (в строках кода), мощность множества регрессионных тестов и количество покрываемых элементов, задействованных исходным множеством тестов.

Общий метод исследования проблемы целесообразности отбора тестов состоит в нахождении или создании исходной и измененной версий некоторой системы и соответствующего набора тестов. В этих условиях применяется методика отбора тестов, и размер и эффективность выбранного набора тестов сравнивается с размером и эффективностью первоначального набора тестов. Результаты показывают, что применение методов отбора регрессионных тестов, в том числе и безопасных, не всегда целесообразно, поскольку затраты и выгоды от их использования изменяются в широком диапазоне в зависимости от многих факторов. На практике наборы, основанные на покрытии, обеспечивают лучшие результаты отбора тестов.

Разумеется, отношение покрытия – не единственный фактор, который может отразиться на целесообразности применения выборочного регрессионного тестирования. Для некоторых приложений создание условий для тестирования (в том числе компиляция и загрузка модулей и ввод данных) может обходиться намного дороже, чем вычислительные ресурсы для непосредственного исполнения тестируемой системы. Например, в телекоммуникационной промышленности стоимость создания тестовой лаборатории для моделирования реальной сети связи может достигать нескольких миллионов долларов.

Подсчет порога целесообразности помогает определить, может ли отбор тестов вообще быть целесообразен для данного программного изделия и набора тестов. Однако даже в случаях, когда значение порога целесообразности указывает, что отбор тестов может быть целесообразен, он не обязательно будет таковым; результат зависит от параметров набора тестов, таких как размер набора, характеристики покрытия кода, уровень подробности и время выполнения тестов, а также от местоположения изменений. Существенно повлиять на общую оценку могут затраты на оплату труда тестового персонала, доступность свободного машинного времени для регрессионного тестирования, доступность стенда, на котором развернуто программное обеспечение приложения и т.п. Отметим, что стоимость прогона тестов связана не столько с размером программы, сколько с ограничениями на допустимое время прогона.

В некоторых случаях, когда число тестов, отброшенных выборочным методом регрессионного тестирования незначительно, но его применение тем не менее заслуживает внимания. Дело в том, что любое сокращение высокозатратного времени использования тестовой лаборатории особенно важно, а для отбора тестов используются другие ресурсы. Подобные обстоятельства необходимо включать в оценку стоимости анализа путем учета не только стоимости эксплуатации ресурса, но и таких факторов как время суток, день недели, время, оставшееся до выпуска очередной версии продукта и т.п. В этом случае модель стоимости должна соблюдать баланс между высокой стоимостью прогона тестов в тестовой лаборатории и относительно небольшой стоимостью проведения анализа на незанятых компьютерах.

Для некоторых программ и наборов тестов выборочное тестирование неэффективно, так как порог целесообразности превышает число тестов в наборе. В таких случаях методы отбора тестов, независимо от того, насколько успешно они уменьшают число тестов, требующих повторного выполнения, не могут давать экономию. Этот результат отражает тот факт, что целесообразность отбора зависит как от стоимости анализа, так и от стоимости выполнения тестов. Возможность достижения экономии при отборе регрессионных тестов для конкретной системы программного обеспечения и конкретного набора тестов должна оцениваться комплексно с учетом всех влияющих на решение факторов.

Стоит заметить, что целесообразность применения выборочного метода регрессионного тестирования нельзя воспринимать как нечто само собой разумеющееся. Следует очень осторожно подходить к оценке целесообразности отбора повторно прогоняемых тестов. В ряде случаев, когда или получаемое число остаточных тестов близко к первоначальному их количеству, или накладные расходы на повторное тестирование незначительны, выгоднее прогонять заново все тесты, особенно если прогон тестов полностью автоматизирован.

Функции предсказания целесообразности

На практике не существует способа в точности предсказать, сколько тестов будет выбрано при минимизации (или по результатам применения любой другой методики отбора тестов). Когда количество тестов, отсеянных по результатам отбора, незначительно, ресурсы,

потраченные на отбор тестов, пропадают впустую. В таких случаях говорят, что выборочное регрессионное тестирование нецелесообразно. Чтобы выяснить, заслуживает ли внимания попытка применения выборочного метода регрессионного тестирования, необходимо использовать прогнозирующую функцию.

Прогнозирующие функции основаны на покрытии кода. Для их вычисления используется информация о доле тестов, активирующих покрываемые сущности – операторы, ветви или функции, – что позволяет предсказать количество тестов, которые будут выбраны при изменении этих сущностей. Существует как минимум одна прогнозирующая функция, которая может быть использована для предсказания целесообразности применения безопасной стратегии выборочного регрессионного тестирования.

Пусть P – тестируемая система, S – ее спецификация, T – набор регрессионных тестов для P, а $|T|$ означает число отдельных тестов в T. Пусть M – выборочный метод регрессионного тестирования, используемый для отбора подмножества T при тестировании измененной версии P; M может зависеть от P, S, T, информации об исполнении T на P и других факторов. Через E обозначим набор рассматриваемых M сущностей тестируемой системы. Предполагается, что T и E - непустые, и что каждый синтаксический элемент P принадлежит, по крайней мере, одной сущности из E. Отношение $covers_M(t, E)$ определяется как отношение покрытия, достигаемого методом M для P. Это отношение определено над $T \times E$ и справедливо тогда и только тогда, когда выполнение теста t на P приводит к выполнению сущности e как минимум один раз. Значение термина «выполнение» определено для всех типов сущностей P. Например, если e – функция или модуль P, e выполняется при вызове этой функции или модуля. Если e – простой оператор, условный оператор, пара определения-использования или другой вид элемента пути в графе выполнения P, e выполняется при выполнении этого элемента пути. Если e – переменная P, e выполняется при чтении или записи этой переменной. Если e – тип P, e выполняется при выполнении любой переменной типа e. Если e – макроопределение P, e выполняется при выполнении расширения этого макроопределения. Если e - сектор P, е выполняется при выполнении всех составляющих его операторов. Соответствующие значения термина «выполнение» могут быть определены по аналогии для других типов сущностей P.

Для данной тестируемой системы P, набора регрессионных тестов T и выборочного метода регрессионного тестирования M можно предсказать, стоит ли задействовать M для регрессионного тестирования будущих версий P, используя информацию об отношении покрытия $covers_M$, достигаемого при использовании M для T и P. Прогноз основан на метрике стоимости, соответствующей P и T. Относительно издержек принимаются некоторые упрощающие предположения.

Пусть E^C обозначает множество покрытых сущностей:

$$E^C = \{e \in E \mid (\exists t \in T)(convers_M(t, E))\}.$$

Обозначение $|E^C|$ используется для числа покрытых сущностей. Иногда удобно представить зависимость $covers_M(t, E)$ в виде бинарной матрицы C, строки которой представляют элементы Т, а столбцы – элементы E. При этом элемент $C_{i,j}$ матрицы C

определяется следующим образом:

$$C_{i,j} = \begin{cases} 1, \textit{если } \mathrm{cov}\,ers_M(i, j) \\ 0, \textit{иначе} \end{cases}$$

Степень накопленного покрытия, обеспечиваемого T, то есть общее число единиц в матрице C, обозначается CC:

$$CC = \sum_{i=1}^{|T|} \sum_{j=1}^{|E|} C_{i,j}$$

Отметим, что если ограничиться включением в C только столбцов, соответствующих покрытым сущностям E^C, накопленное покрытие CC останется неизменным. В частности, для всех непокрытых сущностей и $C_{i,u}$ равно нулю для всех тестов i (так как $covers_M(i, u)$ ложно для всех таких случаев). Следовательно, ограничение на E^C при вычислении суммы, определяющей СС, приводит только к исключению слагаемых, равных нулю.

Пусть T_M подмножество T, выбранное M для P, и пусть $|T_M|$ обозначает его мощность, тогда $T_M = \{t \in T \mid M \textit{ выбирает } t\}$. Пусть S_M удельная стоимость отбора одного теста для T_M при применении M к P, и пусть r – удельная стоимость выполнения одного теста из T на P и проверки его результата. M целесообразно использовать в качестве метода отбора тестов тогда и только тогда, когда:

$s_M |T_M| < r(|T| - |T_M|)$, то есть стоимость анализа, необходимого для отбора T_M, должна быть меньше стоимости прогона невыбранных тестов, $T \supseteq T_M$.

Оценка ожидаемого числа тестов, требующих повторного запуска, обозначается N_M и вычисляется следующим образом:

$$N_M = \frac{CC}{|E|}.$$

Использование этой прогнозирующей функции предполагается только в случаях, когда цель выборочной стратегии регрессионного тестирования состоит в повторном выполнении всех тестов, затронутых изменениями, то есть используется безопасный метод отбора тестов. Несколько усовершенствованный вариант оценки N_M, использующий в качестве пространства сущностей E^C вместо E:

$$N_M^C = \frac{CC}{|E^C|}.$$

Прогнозирующая функция для доли набора тестов, требующей повторного выполнения, то есть для $|T_M|/|T|$ обозначается π_M:

$$\pi_M = \frac{N_M^C}{|T|} = \frac{CC}{|E^C \| T|}.$$

Прогнозирующая функция π_M полагается непосредственно на информацию о покрытии. Главные предпосылки, лежащие в основе применения прогнозирующей функции, таковы:

- Целесообразность применения выборочного метода регрессионного тестирования и, как следствие, наша способность к предсказанию целесообразности непосредственно зависит от доли тестового набора, выбираемой для выполнения методом регрессионного тестирования.
- Эта доля в свою очередь непосредственно зависит от отношения покрытия.

Точность прогнозирующей функции на практике может значительно меняться от версии к версии. Проблема точности может оказаться достаточно серьезной, тем не менее, поскольку прогнозирующая функция используется для долговременного предсказания поведения метода на протяжении нескольких версий, применение средних значений считается допустимым. Отношение $convers_M(t, e)$ в ходе сопровождения изменяется очень слабо. По этой причине информация, полученная в результате анализа единственной версии, может оказаться достаточной для управления отбором тестов на протяжении нескольких последовательных новых версий.

Существуют факторы, влияющие на целесообразность отбора тестов, но не учитываемые прогнозирующей функцией. Один из подходов улучшения качества прогноза состоит в использовании информации об истории изменений программы, которую зачастую можно получить из системы управления конфигурацией.

Например, в табл. 12.1 прогнозирующая функция может быть подсчитана как отношение общего количества звездочек в таблице к количеству строк таблицы, т.е. числу покрываемых сущностей. Эта величина составляет 42 / 11 « 3.8, т.е. безопасный метод будет отбирать в среднем около 4 тестов. Сведения о методике предсказания суммированы в табл. 13.4.

Таблица 13.4. Результаты применения методики предсказания

Характеристика	**Изменение в результате применения методики**
Время работы метода отбора в случае, если (выборочное тестирование целесообразно	Увеличивается незначительно
Время работы метода отбора в случае, если выборочное тестирование нецелесообразно	Уменьшается до пренебрежимо малой величины
Снижение точности предсказания от версии к версии	Зависит от объема изменений
Результаты применения методики на практике	Положительные (ошибка в 0,8%)

Порождение новых тестов

Порождение новых тестов при структурном регрессионном тестировании обычно обусловлено недостаточным уровнем покрытия. Новые тесты разрабатываются так, чтобы задействовать еще не покрытые участки исходного кода. Процесс прекращается, когда уровень покрытия кода достигает требуемой величины (например, 80%). Разработка новых тестов при функциональном регрессионном тестировании является менее тривиальной задачей и обычно связана с вводом новых требований либо с желанием проверить некоторые сценарии работы системы дополнительно.

Основой большинства программных продуктов для управляющих применений, находящихся в промышленном использовании, является цикл обработки событий. Сценарий работы с системой, построенной по такой архитектуре, состоит из последовательности транзакций, управление после обработки каждой транзакции вновь передается циклу обработки событий. Выполнение транзакции приводит к изменению состояния программы; в результате некоторых транзакций происходит выход из цикла и завершение работы программы. Тесты для таких программ представляют собой последовательность транзакций.

Развитие программного продукта от версии к версии влечет за собой появление новых состояний. Поскольку большинство тестов легко может быть расширено путем добавления дополнительных транзакций в список, новые тесты можно создавать путем суперпозиции уже имеющихся, с учетом информации об изменении состояния тестируемой системы в результате прогона теста. Этот подход позволяет указать, какого рода новые тесты с наибольшей вероятностью обнаружат ошибки.

Обозначим тестируемую программу *P*, а множество ее тестов $T = \{t_1, t_2, \ldots, t_n\}$. Будем считать, что состояние тестируемой программы s определяется совокупностью значений некоторого подмножества глобальных и локальных переменных. При создании новых тестов будем рассматривать состояния программы перед запуском теста (s_0) и после его окончания (s_j). Информацию об этих состояниях необходимо собирать для каждого теста по результатам запуска на предыдущей версии продукта. Методика порождения новых тестов на основе анализа «подозрительных» состояний сводится к описанной ниже последовательности действий.

1. Вычисление списка глобальных и локальных переменных, определяющих состояние программы *s*.
2. Сбор информации (на основе анализа профиля программы, полученного на предыдущей версии продукта *i-1*, для каждого существующего теста t_j о состояниях программы перед запуском теста и после его окончания $(m.e. s_0 \; u \; s_j)$. Множество таких состояний обозначается $s_{i-1} : s_{i-1} = s_0 \cup \{s_j \mid \forall j\}$.
3. Выполнение на текущей версии продукта *i* новых и выбранных регрессионных тестов из множества *T'*. По аналогии с s_{i-1} вычисляется множество s_i, которое сохраняется под управлением системы контроля версий.
4. Оценка «подозрительных» с точки зрения наличия ошибок множества новых по сравнению с предыдущими версиями состояний N_i в соответствии со следующей формулой: $N_i = S_i \setminus S_{i-1}$.
5. Анализ состояний множества N_i, в которых дальнейшая работа продукта невозможна в соответствии со спецификацией. Предмет анализа – определить, создаются ли эти состояния в результате выполнения тестов, проверяющих нештатные режимы работы продукта, или каких-либо других тестов. В последнем случае фиксируется ошибка.
6. Исключение нештатных состояний из множества N_i.

7. Переход к шагу 10, если новых состояний, допускающих продолжение выполнения программы, не обнаружено, т.е. $N_i = \otimes$.
8. Для каждого состояния множества N_i вычисление вектора отличия от исходного состояния s_0, т.е. множества переменных, измененных по сравнению с s_0.
9. Модификация множества измененных строк исходного кода *DP* на основе информации об измененных переменных и использование какой-либо методики отбора тестов для выборочного регрессионного тестирования.
10. Повторное выполнение шагов 3-9 до достижения состояния $N_i = \otimes$ либо до истечения времени, отведенного на регрессионное тестирование. Использование методов разбиения на классы эквивалентности для досрочного принятия решения о прекращении цикла тестирования, если ни один из тестов, созданных на очередном этапе, не принадлежит к новому классу эквивалентности.

Для приведенной методики 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 помогает избежать избыточности при многократном слиянии тестов, когда искомые «подозрительные» ситуации возникают в ходе работы теста, но не анализируются, так как не повторяются при его завершении.

Использование описанной методики позволяет в программном комплексе находить ошибки, не обнаруживаемые исходным набором тестов. Отметим, что применение предложенного подхода невозможно для программ, понятие состояния для которых не определено.

Регрессионное тестирование: алгоритм и программная система поддержки

Методика регрессионного тестирования

Методика предназначена для эффективного решения задачи выборочного повторного тестирования. Ее исходными данными являются: программа *P* и ее модифицированная версия *P'*, критерий тестирования *C*, множество (набор) тестов *T*, ранее использовавшихся для тестирования *P*, информация о покрытии элементов *P (M(P,C))* тестами из *T*. Необходимо реализовать эффективный способ, гарантирующий достаточную степень уверенности в правильности P', используя тесты из *T*.

Методика строится на основе сочетания процедур обычного и регрессионного тестирования.

Рассмотрим процедуру обычного тестирования. В ней для получения информации о тестируемых объектах в ходе тестирования необходимо установить соответствие между покрываемыми элементами и тестами для их проверки. Соответственно, процедура тестирования должна включать приведенную ниже последовательность действий:

1. Определить требуемые функциональные возможности программы с использованием, например, метода разбиения на классы эквивалентности.

2. Создать тесты для требуемых функциональных возможностей.
3. Выполнить тесты.
4. В случае необходимости – создать и выполнить дополнительные тесты для покрытия оставшихся (еще не покрытых) структурных элементов (предварительно установив их соответствие функциональным).
5. Создать базу данных тестов программы.

По аналогии с обычным тестированием, процедура регрессионного тестирования в процессе сопровождения состоит из следующих этапов:

1. Использование функции предсказания целесообразности. Если прогнозируемое количество выбранных тестов больше, чем порог целесообразности, провести повторный прогон всех тестов. В противном случае перейти к шагу 2.
2. Идентификация изменений ΔP в программе P' (и множества ΔM измененных покрываемых элементов) и установление взаимно однозначного соответствия между покрываемыми элементами $M(P, C)$ и $M(P', C)$ в соответствии с изменениями:
   $\Delta M = (M(P, C) \setminus M(P', C)) \bigcup (M(P', C) \setminus M(P, C))$.
3. Выбор $T' \subseteq T$ – подмножества исходных тестов, потенциально способных выявить связанные с изменениями ошибки в P', для повторного выполнения на P', с использованием результатов, полученных в пункте 2. Это подмножество можно упорядочить, а также указать число тестов, выполнения которых достаточно для соответствия какому-либо критерию минимизации. Для безопасных методов отбора тестов множество T' удовлетворяет следующим ограничениям:
   $t_i \in T, t_i \notin T' \Rightarrow P(t_i) \equiv P'(t_i)$.
4. Применение подмножества T' для регрессионного тестирования измененной программы P' с целью проверки результатов и установления факта корректности P' по отношению к T' (в соответствии с измененным техническим заданием), а также обновления информации о прохождении тестов из T' на P'.
5. В случае необходимости – создание дополнительных тестов для дополнения набора регрессионных тестов. Это могут быть новые функциональные тесты, необходимые для тестирования изменений в техническом задании или новых функциональных возможностей измененной программы; новые структурные тесты для активизации оставшихся (непокрытых) структурных элементов (предварительно установив их соответствие проверяемым функциональным требованиям).
6. Создание T'' – нового набора тестов для P', применение его для тестирования измененной программы, проверка результатов и установление факта корректности P' по отношению к T'', обновление информации о ходе исполнения теста и создание базы данных тестов измененной программы для хранения этой информации и выходных данных тестов. Удаление устаревших тестов. T'' формируется по следующему правилу:

$$T'' = (T \cup T_{новые}) \setminus T_{устаревшие}.$$

Система поддержки регрессионного тестирования

Структура системы поддержки регрессионного тестирования представлена на рис. 14.1. Исходный код обеих версий тестируемой программы хранится под управлением системы контроля версий. Для отбора тестов по методу покрытия точек использования неисполняемых определений средствами системы контроля версий создается файл различий, на основании которого вычисляется список добавленных, измененных и удаленных строк исходного кода, который является удобной формой представления множества ΔP. Затем производится перебор этого списка. Если какая-либо строка списка представляет собой макроопределение, осуществляется поиск строк кода, содержащих использование этого макроопределения по всему тексту тестируемой программы; найденные строки присоединяются к множеству ΔP. Расширенное множество ΔP сопоставляется с результатами прогона тестов из множества T на предыдущей версии программы. Если в ходе выполнения какого-либо теста t_i получала управление хотя бы одна строка, входящая в множество ΔP тест t_i отбирается для повторного запуска.

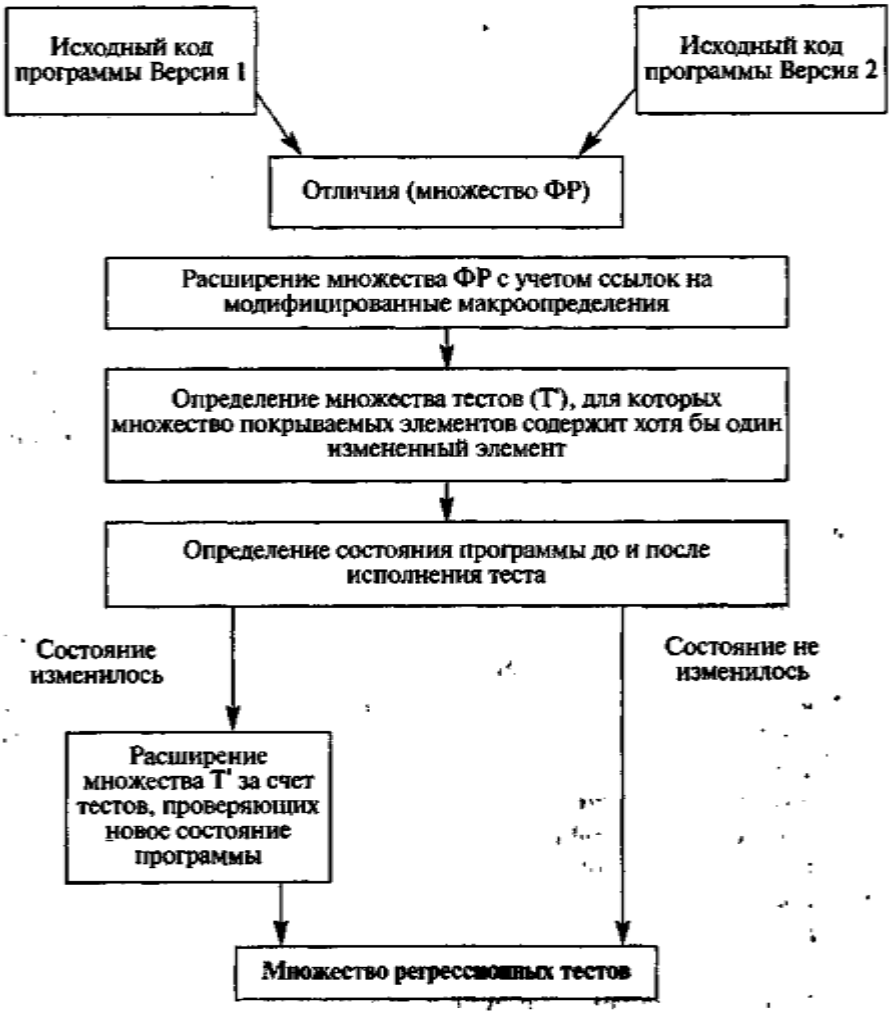

Рис. 14.1. Структура системы поддержки регрессионного тестирования

При создании новых тестов по методу «подозрительных» состояний функция тестируемой программы, содержащая цикл обработки событий, дополняется операторами вывода значений глобальных и видимых локальных переменных. Запуск тестов из множества T' на профилированной версии программы позволяет получить список ее состояний. Этот список анализируется, и для каждого ранее не наблюдавшегося состояния вычисляется список переменных, изменившихся по сравнению с каким-либо известным состоянием. Множество ΔP дополняется строками кода, где используются переменные из этого списка. Для каждого состояния указываются тесты, запуск которых необходим. Наконец, создается список рекомендованных новых тестов в форме, удобной для восприятия человеком.

Выходные данные каждой программы-обработчика доступны пользователю, что позволяет контролировать промежуточные результаты работы системы. К примеру, можно исключить из рассмотрения переменные, которые хотя и изменяются в ходе выполнения программы, на ее состояние не влияют. Архитектура системы позволяет легко расширять функциональность: например, для поддержания какой-либо новой системы контроля версий достаточно создать один новый модуль объемом около 100 строк кода. Остальные модули можно использовать без изменений.

Типовой сценарий проведения регрессионного тестирования программ, написанных на языке C, с применением описанной выше системы состоит из следующих этапов:

1. Вычисляется множество ΔP строк исходного кода, добавленных, удаленных или измененных по сравнению с предыдущей версией.
2. Множество ΔP дополняется строками, непосредственно не изменявшимися, но содержащими ссылки на измененные макроопределения.
3. Вычисляется упорядоченное множество регрессионных тестов T', для которых $\forall i \in T', T_{i,\,j-1} \bigcap \Delta P \neq 0$, где $T_{i,\,j-1}$ – множество строк исходного кода продукта, получающих управление в ходе выполнения теста i на версии системы $j-1$. Тесты упорядочиваются по убыванию количества измененных строк в пути их исполнения.
4. Вычисляется список глобальных и локальных переменных, определяющих состояние программы s. Исходный код тестируемой программы модифицируется так, что информация о состоянии s (значения глобальных, статических и локальных переменных) выводится во внешний файл перед запуском теста и после его окончания.
5. Новые тесты и тесты из множества T' (регрессионные тесты) исполняются на текущей версии продукта j.
6. Тесты, проверяющие нештатные режимы работы продукта, т.е. создающие состояния, в которых дальнейшая работа продукта невозможна в соответствии со спецификацией требований, исключаются из рассмотрения. Если известно, что ни один тест не приводит к возникновению нештатных состояний, данный этап может быть опущен.
7. Для каждого теста i вычисляется множество T_{ij}.
8. Обрабатываются результаты выполнения тестов, и создается множество S_j, состоящее из начального состояния s_0 и всех наблюдавшихся конечных состояний. Вычисляется

множество $N_j = S_j \setminus S_{j-1}$ всех новых по сравнению с предыдущими версиями состояний, которое является «подозрительным» с точки зрения наличия ошибок, и вектора их отличий от исходного состояния s_0, т.е. переменные, измененные по сравнению с s_0.

9. Множество измененных строк исходного кода ΔP дополняется номерами строк, где используются заданные переменные.
10. Если $N_i = \otimes$, цикл работы завершается. Если $N_j \neq \otimes$ следует перейти к шагу 4 или, если счетчик количества итераций работы системы превышает некоторое заданное предельное значение, известить об этом пользователя. Пользователь может принять решение о прекращении тестирования или пропуске некоторого числа циклов.

Если этап 6 выполняется автоматически, а этап 7 можно опустить, возможна полная автоматизация регрессионного тестирования.