МОДУЛЬ 11: «РЕГРЕССИОННОЕ ТЕСТИРОВАНИЕ»

ЦЕЛИ И ЗАДАЧИ РЕГРЕССИОННОГО ТЕСТИРОВАНИЯ

ОПРЕДЕЛЕНИЯ

Тестирование программной системы - не разовое мероприятие, а постоянный процесс, активный в течение всего жизненного цикла разработки системы. В течение этого процесса система неизбежно изменяется - либо в результате исправления ошибок, либо в результате расширения ее функциональности. Задача тестировщика в такой ситуации - подтвердить, что новая или исправленная функциональность не вызвала новые ошибки, а если ошибки все-таки возникли - определить причины их возникновения.

Поэтому при корректировках программы необходимо гарантировать сохранение качества. Для этого используется регрессионное тестирование - выборочное тестирование, позволяющее убедиться: а) изменения не вызвали нежелательных побочных эффектов; б) измененная система по-прежнему соответствует требованиям.

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.

ПЕРВАЯ ЦЕЛЬ регрессионного тестирования - в соответствии с используемым критерием покрытия кода, гарантировать тот же уровень покрытия, что и при полном повторном тестировании программы. Для этого запускаются тесты, относящиеся к измененным областям кода или функциональным возможностям.

ВТОРАЯ ЦЕЛЬ регрессионного тестирования - удостовериться, что программа функционирует в соответствии со своей спецификацией, и что изменения не привели к внесению новых ошибок в ранее протестированный код.

Самый простой способ регрессионного тестирования - полное выполнение всех тестов после каждого существенного изменения системы и сравнение результатов выполнения тестов до и после изменения. Однако 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 Таблице 11.1 [Котляров 2006].

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

ДОПОЛНИТЕЛЬНО ТРЕТЬЕЙ ЦЕЛЬЮ регрессионного тестирования является уменьшение стоимости и сокращение времени выполнения тестов.

Если результаты выполнения тестов до изменений были положительными, то появление неудачных тестов при регрессионном тестировании может означать, что в системе появились новые дефекты, вызванные исправлением старых.

Повторное выполнение тестов может завершиться одним из трех способов:

- ✓ Все тесты пройдены успешно. В этом случае изменения не затрагивают уже протестированные функции, но может потребоваться разработка новых тестовых примеров для новых функций системы.
- ✓ Часть тестов, ранее выполнявшихся успешно, завершается с отрицательным результатом. Причины этого могут быть следующие:
  - o корректное изменение функциональности тестируемой системы, в результате которого тестовый пример перестал соответствовать требованиям;
  - o некорректное изменение функциональности системы, в результате которого тестовый пример выявил расхождение с требованиями;
  - o влияние остаточных данных от предыдущих тестовых примеров, ранее остававшееся незамеченным.
- ✓ Выполнение тестов аварийно завершается в самом начале или при выполнении определенного тестового примера.

В некоторых случаях повторное выполнение всех тестов невозможно из-за большого времени выполнения всех тестов и ограниченным временем, отведенным на тестирование. В этом случае применяется практика выборочного регрессионного тестирования отдельных частей системы, затронутых изменениями. Полное тестирование при таком подходе проводится только после накопления достаточно большого количества изменений или на ключевых стадиях проекта.

Регрессионное тестирование включает в себя следующие стадии:

- ✓ Анализ изменений в системе.
- ✓ Выбор тестовых примеров для проверки системы.
- ✓ Выполнение тестовых примеров.
- ✓ Анализ результатов выполнения.
- ✓ Модификация тестового окружения, тестовых примеров или уведомление разработчиков о дефекте системы.

Таким образом, можно определить следующие основные задачи повторяемости тестирования при внесении изменений:

- ✓ обеспечение возможности полного выполнения всех тестов, проверяющих функциональность системы или проведение анализа, позволяющего выявить тесты, которые должны быть повторно выполнены для тестирования изменившейся функциональности;
- ✓ разработка тестовых примеров и тестового окружения с использованием методик, облегчающих модификацию при изменениях в тестируемой системе;
- ✓ разработка тестовых примеров, структура которых полностью исключает их взаимное влияние по остаточным данным.

Следствием повторяемости тестирования является постоянное обеспечение тестировщиков и разработчиков актуальной информацией о текущем состоянии системы и корректности изменений, внесенных в ходе разработки системы.

ВИДЫ РЕГРЕССИОННОГО ТЕСТИРОВАНИЯ

Поскольку регрессионное тестирование представляет собой повторное проведение цикла обычного тестирования, виды регрессионного тестирования совпадают с видами обычного тестирования. **МОЖНО ГОВОРИТЬ О МОДУЛЬНОМ РЕГРЕССИОННОМ ТЕСТИРОВАНИИ И 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.
- ✓ **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 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.

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 выполняется после адаптивного или усовершенствующего сопровождения, а 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.

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

Практика тестирования измененной версии программы P’ в тех же условиях, в которых тестировалась исходная программа P, называется **УПРАВЛЯЕМЫМ РЕГРЕССИОННЫМ ТЕСТИРОВАНИЕМ**.

Для обеспечения управляемости регрессионного тестирования необходимо выполнение ряда условий:

- ✓ Как при модульном, так и при интеграционном регрессионном тестировании в качестве модулей, вызываемых тестируемым модулем должны использоваться реальные модули системы.
- ✓ 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.

- ✓ 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.
- ✓ Для проведения регрессионного тестирования с использованием существующего набора тестов необходимо хранить информацию о результатах выполнения тестов на предыдущих этапах тестирования.

ПРИМЕР РЕГРЕССИОННОГО ТЕСТИРОВАНИЯ

Получив отчет об ошибке, программист анализирует исходный код, находит ошибку, исправляет ее и модульно или интеграционно тестирует результат.

В свою очередь тестировщик, проверяя внесенные программистом изменения, должен:

1. Проверить и утвердить исправление ошибки. Для этого необходимо выполнить указанный в отчете тест, с помощью которого была найдена ошибка.
2. Попробовать воспроизвести ошибку каким-нибудь другим способом.
3. Протестировать последствия исправлений. Возможно, что внесенные исправления привнесли ошибку (наведенную ошибку) в код, который до этого исправно работал.

ФОРМИРОВАНИЕ РЕГРЕССИОННОГО НАБОРА ТЕСТОВ

КЛАССИФИКАЦИЯ ТЕСТОВ ПРИ ОТБОРЕ

Создание наборов регрессионных тестов начинают с множества исходных тестов. При заданном критерии регрессионного тестирования исходные тесты подразделяются на подмножества:

- ✓ **МНОЖЕСТВО ТЕСТОВ, ПРИГОДНЫХ ДЛЯ ПОВТОРНОГО ИСПОЛЬЗОВАНИЯ**. 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.
- ✓ **МНОЖЕСТВО ТЕСТОВ, ТРЕБУЮЩИХ ПОВТОРНОГО ЗАПУСКА**. Тесты, которые уже запускались, но требуют перезапуска, т.к. затрагивают, по крайней мере, один измененный покрываемый элемент, подлежащий повторному тестированию. При повторном выполнении эти тесты могут давать результат, отличный от результата, показанного на исходной программе. Эти тесты обеспечивают хорошее покрытие структурных элементов даже при наличии новых функциональных возможностей.
- ✓ **МНОЖЕСТВО УСТАРЕВШИХ ТЕСТОВ**. Тесты, не применимые к измененной программе и непригодные для дальнейшего тестирования, поскольку они затрагивают только покрываемые элементы, которые были удалены при изменении программы. Они удаляются из набора регрессионных тестов.
- ✓ **НОВЫЕ ТЕСТЫ, КОТОРЫЕ ЕЩЕ НЕ ЗАПУСКАЛИСЬ** и могут быть использованы для тестирования.

На Рис.11.2 [Котляров 2006] показан жизненный цикл теста. После создания тест вводится в базу 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.

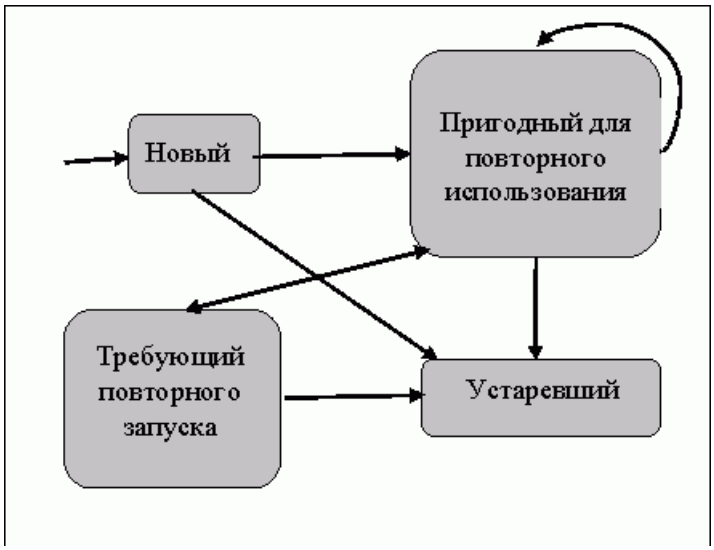

Рис. 11.2. Жизненный цикл теста.

ВОЗМОЖНОСТИ ПОВТОРНОГО ИСПОЛЬЗОВАНИЯ ТЕСТОВ

К изменению существующих тестов могут привести три следующих вида деятельности программистов:

1. Создание новых тестов.
2. Выполнение тестов.
3. Изменение кода.

Существуют четыре уровня **ПОВТОРНОГО ИСПОЛЬЗОВАНИЯ ТЕСТА**:

- ✓ **УРОВЕНЬ 1**. Тест не допускает повторного использования. Требуется создание нового набора тестов.
- ✓ **УРОВЕНЬ 2**. Повторное использование возможно только входных данных теста. Часто 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 новая траектория и выходные данные теста могут отличаться от предыдущих результатов. Поэтому эти тесты должны быть запущены повторно для получения новых выходных данных.
- ✓ **УРОВЕНЬ 3**. Возможно повторное использование как входных, так и выходных данных теста. На этом уровне обычно располагаются функциональные тесты. Если модуль подвергся только изменению кода с сохранением функциональности, возможно повторное использование функциональных тестов для проверки правильности реализации. Так как 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 - повторное использование входных данных, выходных данных и траектории теста. В этом случае на траектории теста не изменяется ни один оператор. Поэтому в повторном запуске этих тестов необходимости нет, так как выходные данные и траектория останутся неизменными.

ЛИТЕРАТУРА

[Котляров 2006] - В.П. Котляров, Т.В. Коликова. Основы тестирования программного обеспечения. Учебное пособие. М.: Интернет-Университет Информационных технологий.

ОПТИМИЗАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТИ ТЕСТОВЫХ ПРИМЕРОВ

Отметим, что входные данные в каждом тестовом примере явно задают начальное состояние тестируемой системы и режимы ее работы при выполнении тестового сценария. Однако неявное влияние на выполнение теста оказывает и состояние тестового окружения - набор параметров, изменение любого из которых может повлиять либо на результат выполнения теста, либо на возможность его корректной работы и завершения. Эта информация обычно отсутствует в тест-планах, однако требуемое для выполнения тестов состояние тестового окружения необходимо учитывать при разработке тестовых примеров.

Например, рассмотрим программную систему, которая может стартовать двумя различными способами - с настройками по умолчанию после включения (режим FACTORY_SETTINGS), и с последними сохраненными настройками после перезагрузки (режим COLD_START) [Синицын 2006]. При этом при старте в режиме FACTORY_SETTINGS значения по умолчанию присваиваются всем настройкам системы, а после перезагрузки (режим COLD_START) все настройки остаются в значениях, установленных непосредственно перед перезагрузкой.

Для проверки следующих требований:

> *Проверить, что после включения системы настройки устанавливаются в значения по умолчанию.*
>
> *Проверить, что после перезагрузки системы настройки устанавливаются в последнее сохраненное значение.*

необходимы как минимум три тестовых примера со следующими сценариями:

> *Тестовый пример 1*
>
> *Включить систему в режиме FACTORY_SETTINGS*
>
> *Проверить, что настройки имеют значения по умолчанию (в реальном тест-плане здесь должны быть проверки конкретных значений переменных).*
>
> *Тестовый пример 2*
>
> *Включить систему в режиме FACTORY_SETTINGS*
>
> *Перезагрузить систему (вызвать ее старт в режиме COLD_START)*
>
> *Проверить, что настройки имеют значения по умолчанию (в реальном тест-плане здесь должны быть проверки конкретных значений переменных)*
>
> *Тестовый пример 3*
>
> *Включить систему в режиме FACTORY_SETTINGS*
>
> *Изменить значения настроек системы (в реальном тест-плане здесь должны быть установлены конкретные значения переменных)*
>
> *Перезагрузить систему (вызвать ее старт в режиме COLD_START)*

> *Проверить, что настройки имеют последние введенные значения (в реальном тест-плане здесь должны быть проверки конкретных значений переменных)*

Первый пункт сценария во всех трех тестах одинаков. Если первоначальный старт системы в режиме FACTORY_SETTINGS занимает значительное время, то общее время выполнения трех тестов будет еще больше. Если общее количество подобных тестов достаточно велико (десятки и сотни), то при выполнении тестов будет нерационально расходоваться время на выполнение тестовых примеров - время на инициализацию системы в каждом тестовом примере будет превышать суммарное время выполнения «полезных» этапов сценариев тестовых примеров.

Для экономии времени можно инициализировать систему в режиме FACTORY_SETTINGS только в первом тестовом примере. Второй и третий тесты начнут свою работу из расчета, что система уже была включена в режиме FACTORY_SETTINGS, и все значения настроек уже установлены в некоторые значения. Сценарии тестов при этом будут выглядеть следующим образом:

> *Тестовый пример 1*
>
> *Включить систему в режиме FACTORY_SETTINGS*
>
> *Проверить, что настройки имеют значения по умолчанию (в реальном тест-плане здесь должны быть проверки конкретных значений переменных)*
>
> *Тестовый пример 2*
>
> *Перезагрузить систему (вызвать ее старт в режиме COLD_START)*
>
> *Проверить, что настройки имеют значения по умолчанию (в реальном тест-плане здесь должны быть проверки конкретных значений переменных)*
>
> *Тестовый пример 3*
>
> *Изменить значения настроек системы*
>
> *Перезагрузить систему (вызвать ее старт в режиме COLD_START)*
>
> *Проверить, что настройки имеют последние введенные значения (в реальном тест-плане здесь должны быть проверки конкретных значений переменных)*

При такой структуре тестовых примеров важна последовательность их выполнения. Первый тестовый пример инициализирует тестируемую систему и приводит ее в необходимое начальное состояние (запускает ее в режиме FACTORY_SETTINGS), второй и третий примеры, считая, что система уже инициализирована, проверяют только ее работу при перезагрузке.

В ходе разработки системы требования и программный код могут измениться таким образом, что при регрессионном тестировании может быть принято решение о выполнении тестов только для режима COLD_START. Если при этом будут выполняться только тестовые примеры 2 и 3, то корректное выполнение сценария станет невозможным - значения настроек системы не получили значений по умолчанию при старте системы, а сама система запускается в нештатном режиме - перезагружается не включившись.

Чтобы диагностировать такие ситуации, в состав предусловий тестовых примеров 2 и 3 необходимо включать проверки того, что к моменту выполнения тестового примера система находится в необходимом состоянии. Первый тестовый пример при этом может выставлять некоторый флаг (переменную в тестовом

окружении), установленное значение которого будет сигнализировать о том, что система корректно стартовала.

При наличии таких проверок тестовые примеры будут выглядеть следующим образом:

> *Первоначальные установки тестового окружения.*
>
> *Установить значение флага Флаг_Система_Стартовала = FALSE*
>
> *Тестовый пример 1.*
>
> *Включить систему в режиме FACTORY_SETTINGS*
>
> *Установить значение флага Флаг_Система_Стартовала = TRUE*
>
> *Проверить, что настройки имеют значения по умолчанию (в реальном тест-плане здесь должны быть проверки конкретных значений переменных).*
>
> *Тестовый пример 2.*
>
> *Проверить, что флаг Флаг_Система_Стартовала = TRUE, иначе прервать тестирование с выдачей диагностического сообщения.*
>
> *Перезагрузить систему (вызвать ее старт в режиме COLD_START)*
>
> *Проверить, что настройки имеют значения по умолчанию (в реальном тест-плане здесь должны быть проверки конкретных значений переменных).*
>
> *Тестовый пример 3.*
>
> *Проверить, что флаг Флаг_Система_Стартовала = TRUE, иначе прервать тестирование с выдачей диагностического сообщения.*
>
> *Изменить значения настроек системы (в реальном тест-плане здесь должны быть установлены конкретные значения переменных).*
>
> *Перезагрузить систему (вызвать ее старт в режиме COLD_START)*
>
> *Проверить, что настройки имеют последние введенные значения (в реальном тест-плане здесь должны быть проверки конкретных значений переменных).*

Пример может показаться несколько надуманным, однако, на практике часто возникает ситуация в которой друг за другом следует несколько десятков тестовых примеров, а при регрессионном тестировании требуется выполнить, например, тестовые примеры с номерами от 25 по 40. Первый тестовый пример при этом инициализирует систему, а остальные работают с уже стартовавшей системой. Если просто выполнять тестовые примеры 25-40, то их выполнение окажется невозможным - они не инициализируют систему. Разумным выходом из этой ситуации является выполнение тестовых примеров 1, 25-40.

Для облегчения проведения регрессионного тестирования (и тестирования вообще) тестовые примеры часто разбивают на группы. Каждая группа содержит набор тестовых примеров, проверяющих отдельную локальную часть функциональности тестируемой системы. При отборе тестовых примеров для частичного

регрессионного тестирования их можно отбирать сразу группами. Тестовые примеры из предыдущего раздела можно разбить на две группы:

> *Тестирование старта системы: тестовый пример 1*
>
> *Тестирование перезагрузки системы: тестовые примеры 2-3*

Разбиение тестов на группы удобно и с точки зрения установки начального состояния тестового окружения для выполнения тестов - так, перед выполнением группы тестов можно инициализировать значения переменных или состояние системы, необходимое для выполнения всей группы. Такие установки называются настройками группы тестов по умолчанию (group defaults, test group defaults).

Перед выполнением каждого теста может потребоваться установка одних и тех же переменных в одни и те же значения. Чтобы не дублировать эти установки в описании каждого тестового примера, в тест-плане можно определить настойки по умолчанию для каждого теста (test case defaults), например следующим образом:

> *Первоначальные установки тестового окружения*
>
> *Установить значение флага Флаг_Система_Стартовала = FALSE*
>
> *Настройки по умолчанию для группы:*
>
> *Установить сервисный режим работы системы*
>
> *Настройки по умолчанию для тестового примера:*
>
> *Обнулить значения выходных переменных тестового окружения, в котором сохраняются настройки системы.*
>
> *Группа 1: Тестирование старта системы (режим FACTORY_SETTINGS)*
>
> *Тестовый пример 1.*
>
> *Включить систему в режиме FACTORY_SETTINGS*
>
> *Установить значение флага Флаг_Система_Стартовала = TRUE*
>
> *Проверить, что настройки имеют значения по умолчанию (в реальном тест-плане здесь должны быть проверки конкретных значений переменных).*
>
> *Группа 2: Тестирование перезагрузки системы (режим COLD_START)*
>
> *Тестовый пример 2.*
>
> *Проверить, что флаг Флаг_Система_Стартовала = TRUE, иначе прервать тестирование с выдачей диагностического сообщения.*
>
> *Перезагрузить систему (вызвать ее старт в режиме COLD_START).*
>
> *Проверить, что настройки имеют значения по умолчанию (в реальном тест-плане здесь должны быть проверки конкретных значений переменных).*
>
> *Тестовый пример 3*

Проверить, что флаг Флаг_Система_Стартовала = TRUE, иначе прервать тестирование с выдачей диагностического сообщения.

Изменить значения настроек системы (в реальном тест-плане здесь должны быть установлены конкретные значения переменных).

Перезагрузить систему (вызвать ее старт в режиме COLD_START).

Проверить, что настройки имеют последние введенные значения (в реальном тест-плане здесь должны быть проверки конкретных значений переменных).

Для облегчения проведения выборочного регрессионного тестирования каждый тестовый пример должен быть полностью автономным - ход его выполнения и результат, не должны зависеть от предыдущих тестовых примеров. Тем самым при выборочном тестировании результат тестирования не зависит от выбранного набора тестовых примеров. Однако, на практике создание автономных тестов зачастую невозможно по различным причинам (как правило - из-за длительного времени выполнения таких тестов).

В случае, когда в наборе тестовых примеров тесты не являются автономными, говорят о **ТЕСТОВОЙ ЗАВИСИМОСТИ**. Тестовая зависимость бывает двух видов - **ПРЕДУСМОТРЕННАЯ СТРУКТУРОЙ ТЕСТОВЫХ ПРИМЕРОВ И ПАРАЗИТНАЯ.**

ПРИМЕР ПРЕДУСМОТРЕННОЙ ТЕСТОВОЙ ЗАВИСИМОСТИ был рассмотрен в предыдущем разделе - корректность выполнения тестов определялась порядком их выполнения. Такая тестовая зависимость требует документирования и сопровождения, как и сами описания тестовых примеров.

ПАРАЗИТНЫЕ ТЕСТОВЫЕ ЗАВИСИМОСТИ ВЫЗВАНЫ НЕКОРРЕКТНЫМ СОСТАВЛЕНИЕМ ТЕСТ-ПЛАНА. Проявляются они в том, что один (или более) тестов корректно работает только в том случае, если до него были выполнены другие тестовые примеры и эта зависимость не предусмотрена тестером. Природа паразитной тестовой зависимости схожа с ошибками использования неинициализированных или остаточных данных в динамической памяти при программировании.

ЛИТЕРАТУРА

[Синицын 2006] - Синицын С.В., Налютин Н.Ю. Верификация программного обеспечения. Курс лекций. Московский инженерно-физический институт. М. 2006.

РЕГРЕССИОННОЕ ТЕСТИРОВАНИЕ ФУНКЦИИ РЕШЕНИЯ КВАДРАТНОГО УРАВНЕНИЯ

```
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;
}
```

Пример 11.1. Функция Equation - исходная версия.

Код этой функции приведен на Примере 11.1 [Котляров 2006]. Входные параметры - коэффициенты квадратного уравнения A, B и C, а также флаг Print, ненулевое значение которого указывает, что решение необходимо вывести на экран. Выходные параметры - X1 и X2, предназначенные для хранения корней уравнения, и возвращаемое значение функции - дискриминант уравнения.

Существующие тесты для функции Equation приведены в Таблице 11.2 [Котляров 2006]. Входные данные тестов представляют собой совокупность значений Print, A, B и C, подаваемых на вход функции. Выходными данными для теста являются значения X1 и X2, возвращаемое значение функции, а также строка, выводимая на экран; в Таблице 11.2 приведены ожидаемые значения выходных данных и для каждого теста вычисляется траектория его прохождения по коду.

Таблица 11.2. Входные и выходные данные тестов.

Тест	**Входные данные(A, B, C, Print)**	**Ожидаемые выходные данные (X1, X2) Возвращаемое значение Выводимая строка**
1	11-612-325	Solution: X1 = 2, X2 = -3
2	2-3510.755.567764-31	Solution: X1 = 0.75+5.567764i, X2 = 0.75-5.567764i
3	12000-24	
4	1210-1-10	
5	1220-12-4	

В исходном виде программа содержит дефект, в результате чего уравнения с отрицательным дискриминантом порождают ошибку времени выполнения. В новой версии программы дефект должен быть исправлен; кроме того, необходимо реализовать запрос пользователя на изменение формата вывода решения. Код новой версии функции Equation приводится на Примере 11.2 [Котляров 2006].

```
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;
}
```

Пример 11.2. Функция Equation - измененная версия.

При изменении функции Equation от Примера 11.1 к Примеру 11.2 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.

ЛИТЕРАТУРА

[Котляров 2006] - В.П. Котляров, Т.В. Коликова. Основы тестирования программного обеспечения. Учебное пособие. М.: Интернет-Университет Информационных технологий.