Лишь та - ошибка, что не исправляется.
Конфуций

Лекция 10.
ТЕСТИРОВАНИЕ И ОТЛАДКА ПРОГРАММНОГО СРЕДСТВА

Основные понятия. Стратегия проектирования тестов. Заповеди отладки. Автономная отладка и тестирование программного модуля. Комплексная отладка и тестирование программного средства.

10.1. Основные понятия.

Отладка ПС - это деятельность, направленная на обнаружение и исправление ошибок в ПС с использованием процессов выполнения его программ. *Тестирование* ПС - это процесс выполнения его программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется *тестовым* или просто *тестом*. Таким образом, отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть констатировано наличие в ПС ошибки, поиска места ошибки в программах и документации ПС и редактирования программ и документации с целью устранения обнаруженной ошибки. Другими словами:

Отладка = Тестирование + Поиск ошибок + Редактирование.

В зарубежной литературе отладку часто понимают [10.1-10.3] только как процесс поиска и исправления ошибок (без тестирования), факт наличия которых устанавливается при тестировании. Иногда тестирование и отладку считают синонимами [10.4,10.5]. В нашей стране в понятие отладки обычно включают и тестирование [10.6 -10.8], поэтому мы будем следовать сложившейся традиции. Впрочем, совместное рассмотрение в данной лекции этих процессов делает указанное разночтение не столь существенным. Следует, однако, отметить, что тестирование используется и как часть процесса аттестации ПС (см. лекцию 14).

10.2. Принципы и виды отладки программного средства.

Успех отладки ПС в значительной степени предопределяет рациональная организация тестирования. При отладке ПС отыскиваются и устраняются, в основном, те ошибки, наличие которых в ПС устанавливается при тестировании. Как было уже отмечено, тестирование не может доказать правильность ПС [10.9],

в лучшем случае оно может продемонстрировать наличие в нем ошибки. Другими словами, нельзя гарантировать, что тестированием ПС практически выполнимым набором тестов можно установить наличие каждой имеющейся в ПС ошибки. Поэтому возникает две задачи. Первая задача: подготовить такой набор тестов и применить к ним ПС, чтобы обнаружить в нем по возможности большее число ошибок. Однако чем дольше продолжается процесс тестирования (и отладки в целом), тем большей становится стоимость ПС. Отсюда вторая задача: определить момент окончания отладки ПС (или отдельной его компоненты). Признаком возможности окончания отладки является полнота охвата пропущенными через ПС тестами (т.е. тестами, к которым применено ПС) множества различных ситуаций, возникающих при выполнении программ ПС, и относительно редкое проявление ошибок в ПС на последнем отрезке процесса тестирования. Последнее определяется в соответствии с требуемой степенью надежности ПС, указанной в спецификации его качества.

Для оптимизации набора тестов, т.е. для подготовки такого набора тестов, который позволял бы при заданном их числе (или при заданном интервале времени, отведенном на тестирование) обнаруживать большее число ошибок в ПС, необходимо, во-первых, заранее планировать этот набор и, во-вторых, использовать рациональную стратегию планирования (проектирования [10.1]) тестов. Проектирование тестов можно начинать сразу же после завершения этапа внешнего описания ПС. Возможны разные подходы к выработке стратегии проектирования тестов, которые можно условно графически разместить (см. рис. 9.1) между следующими двумя крайними подходами [10.1]. Левый крайний подход заключается в том, что тесты проектируются только на основании изучения спецификаций ПС (внешнего описания, описания архитектуры и спецификации модулей). Строение модулей при этом никак не учитывается, т.е. они рассматриваются как черные ящики. Фактически такой подход требует полного перебора всех наборов входных данных, так как в противном случае некоторые участки программ ПС могут не работать при пропуске любого теста, а это значит, что содержащиеся в них ошибки не будут проявляться. Однако тестирование ПС полным множеством наборов входных данных практически неосуществимо. Правый крайний подход заключается в том, что тесты проектируются на основании изучения текстов программ с целью протестировать все пути выполнения каждой программ ПС. Если принять во внимание наличие в программах циклов с переменным числом повторений, то различных путей выполнения программ ПС может оказаться также

чрезвычайно много, так что их тестирование также будет практически неосуществимо.

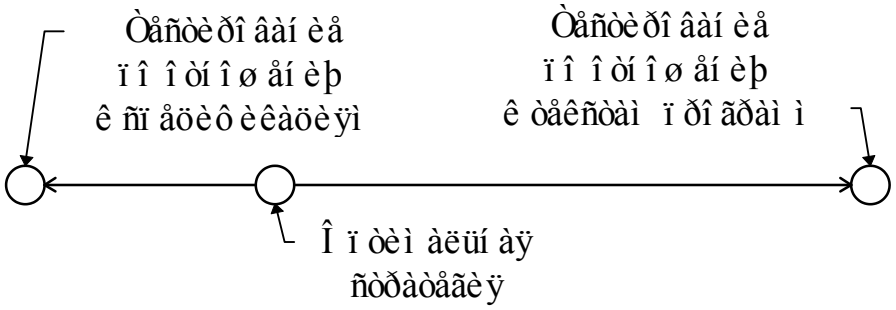

Рис. 10.1. Спектр подходов к проектированию тестов.

Оптимальная стратегия проектирования тестов расположена внутри интервала между этими крайними подходами, но ближе к левому краю. Она включает проектирование значительной части тестов по спецификациям, но она требует также проектирования некоторых тестов и по текстам программ. При этом в первом случае эта стратегия базируется на принципах:

- на каждую используемую функцию или возможность - хотя бы один тест,
- на каждую область и на каждую границу изменения какой-либо входной величины - хотя бы один тест,
- на каждую особую (исключительную) ситуацию, указанную в спецификациях, - хотя бы один тест.

Во втором случае эта стратегия базируется на принципе: каждая команда каждой программы ПС должна проработать хотя бы на одном тесте.

Оптимальную стратегию проектирования тестов можно конкретизировать на основании следующего принципа: для каждого программного документа (включая тексты программ), входящего в состав ПС, должны проектироваться свои тесты с целью выявления в нем ошибок. Во всяком случае, этот принцип необходимо соблюдать в соответствии с определением ПС и содержанием понятия технологии программирования как технологии разработки надежных ПС (см. лекцию 1). В связи с этим Майерс даже определяет разные виды тестирования [10.1] в зависимости от вида программного документа, на основании которого строятся тесты. В нашей стране различаются [10.8] два основных вида отладки (включая тестирование): автономную и комплексную отладку ПС. *Автономная* отладка ПС означает последовательное раздельное тестирование различных частей программ, входящих в ПС, с

поиском и исправлением в них фиксируемых при тестировании ошибок. Она фактически включает отладку каждого программного модуля и отладку сопряжения модулей. *Комплексная* отладка означает тестирование ПС в целом с поиском и исправлением фиксируемых при тестировании ошибок во всех документах (включая тексты программ ПС), относящихся к ПС в целом. К таким документам относятся определение требований к ПС, спецификация качества ПС, функциональная спецификация ПС, описание архитектуры ПС и тексты программ ПС.

10.3. Заповеди отладки программного средства.

В этом разделе даются общие рекомендации по организации отладки ПС. Но сначала следует отметить некоторый феномен [10.1], который подтверждает важность предупреждения ошибок на предыдущих этапах разработки: по мере роста числа обнаруженных и исправленных ошибок в ПС *растет* также относительная вероятность существования в нем необнаруженных ошибок. Это объясняется тем, что при росте числа ошибок, обнаруженных в ПС, уточняется и наше представление об общем числе допущенных в нем ошибок, а значит, в какой-то мере, и о числе необнаруженных еще ошибок.

Ниже приводятся рекомендации по организации отладки в форме заповедей [10.1, 10.8].

Заповедь 1. Считайте тестирование ключевой задачей разработки ПС, поручайте его самым квалифицированным и одаренным программистам; нежелательно тестировать свою собственную программу.

Заповедь 2. Хорош тот тест, для которого высока вероятность обнаружить ошибку, а не тот, который демонстрирует правильную работу программы.

Заповедь 3. Готовьте тесты как для правильных, так и для неправильных данных.

Заповедь 4. Документируйте пропуск тестов через компьютер; детально изучайте результаты каждого теста; избегайте тестов, пропуск которых нельзя повторить.

Заповедь 5. Каждый модуль подключайте к программе только один раз; никогда не изменяйте программу, чтобы облегчить ее тестирование.

Заповедь 6. Пропускайте заново все тесты, связанные с проверкой работы какой-либо программы ПС или ее взаимодействия с другими программами, если в нее были внесены изменения (например, в результате устранения ошибки).

10.4. Автономная отладка программного средства.

При автономной отладке ПС каждый модуль на самом деле тестируется в некотором программном окружении, кроме случая, когда отлаживаемая программа состоит только из одного модуля. Это окружение состоит [10.8] из других модулей, часть которых является модулями отлаживаемой программы, которые уже отлажены, а часть - модулями, управляющими отладкой (*отладочными* модулями, см. ниже). Таким образом, при автономной отладке тестируется всегда некоторая программа (*тестируемая программа*), построенная специально для тестирования отлаживаемого модуля. Эта программа лишь частично совпадает с отлаживаемой программой, кроме случая, когда отлаживается последний модуль отлаживаемой программы. В процессе автономной отладки ПС производится наращивание тестируемой программы отлаженными модулями: при переходе к отладке следующего модуля в его программное окружение добавляется последний отлаженный модуль. Такой процесс наращивания программного окружения отлаженными модулями называется *интеграцией* программы [10.1]. Отладочные модули, входящие в окружение отлаживаемого модуля, зависят от порядка, в каком отлаживаются модули этой программы, от того, какой модуль отлаживается и, возможно, от того, какой тест будет пропускаться.

При восходящем тестировании (см. лекцию 7) это окружение будет содержать только один отладочный модуль (кроме случая, когда отлаживается последний модуль отлаживаемой программы), который будет головным в тестируемой программе. Такой отладочный модуль называют *ведущим* (или драйвером [10.1]). Ведущий отладочный модуль подготавливает информационную среду для тестирования отлаживаемого модуля (т. е. формирует ее состояние, требуемое для тестирования этого модуля, в частности, путем ввода некоторых тестовых данных), осуществляет обращение к отлаживаемому модулю и после окончания его работы выдает необходимые сообщения. При отладке одного модуля для разных тестов могут составляться разные ведущие отладочные модули.

При нисходящем тестировании (см. лекцию 7) окружение отлаживаемого модуля в качестве отладочных модулей содержит *отладочные имитаторы* (заглушки) некоторых еще не отлаженных модулей. К таким модулям относятся, прежде всего, все модули, к которым может обращаться отлаживаемый модуль, а также еще не отлаженные модули, к которым могут обращаться уже отлаженные модули (включенные в это окружение). Некоторые из этих

имитаторов при отладке одного модуля могут изменяться для разных тестов.

На практике в окружении отлаживаемого модуля могут содержаться отладочные модули обоих типов, если используется смешанная стратегия тестирования. Это связано с тем, что как восходящее, так и нисходящее тестирование имеет свои достоинства и свои недостатки.

К *достоинствам восходящего тестирования* относятся:

- простота подготовки тестов,
- возможность полной реализации плана тестирования модуля.

Это связано с тем, что тестовое состояние информационной среды готовится непосредственно перед обращением к отлаживаемому модулю (ведущим отладочным модулем).

Недостатками восходящего тестирования являются следующие его особенности:

- тестовые данные готовятся, как правило, не в той форме, которая рассчитана на пользователя (кроме случая, когда отлаживается последний, головной, модуль отлаживаемой программ);
- большой объем отладочного программирования (при отладке одного модуля приходится составлять много ведущих отладочных модулей, формирующих подходящее состояние информационной среды для разных тестов);
- необходимость специального тестирования сопряжения модулей.

К *достоинствам нисходящего тестирования* относятся следующие его особенности:

- большинство тестов готовится в форме, рассчитанной на пользователя;
- во многих случаях относительно небольшой объем отладочного программирования (имитаторы модулей, как правило, весьма просты и каждый пригоден для большого числа, нередко - для всех, тестов);
- отпадает необходимость тестирования сопряжения модулей.

Недостатком нисходящего тестирования является то, что тестовое состояние информационной среды перед обращением к отлаживаемому модулю готовится косвенно - оно является результатом применения уже отлаженных модулей к тестовым данным или данным, выдаваемым имитаторами. Это, во-первых, затрудняет подготовку тестов и требует высокой квалификации тестовика (разработчика тестов), а во-вторых, делает затруднительным или даже невозможным реализацию полного плана тестирования отлаживаемого модуля. Указанный недостаток иногда

вынуждает разработчиков применять восходящее тестирование даже в случае нисходящей разработки. Однако чаще применяют некоторые модификации нисходящего тестирования, либо некоторую комбинацию нисходящего и восходящего тестирования. Исходя из того, что нисходящее тестирование, в принципе, является предпочтительным, остановимся на приемах, позволяющих в какой-то мере преодолеть указанные трудности.

Прежде всего, необходимо организовать отладку программы таким образом, чтобы как можно раньше были отлажены модули, осуществляющие ввод данных, - тогда тестовые данные можно готовить в форме, рассчитанной на пользователя, что существенно упростит подготовку последующих тестов. Далеко не всегда этот ввод осуществляется в головном модуле, поэтому приходится в первую очередь отлаживать цепочки модулей, ведущие к модулям, осуществляющим указанный ввод (ср. с методом целенаправленной конструктивной реализации в лекции 7). Пока модули, осуществляющие ввод данных, не отлажены, тестовые данные поставляются некоторыми имитаторами: они либо включаются в имитатор как его часть, либо вводятся этим имитатором.

При нисходящем тестировании некоторые состояния информационной среды, при которых требуется тестировать отлаживаемый модуль, могут не возникать при выполнении отлаживаемой программы ни при каких входных данных. В этих случаях можно было бы вообще не тестировать отлаживаемый модуль, так как обнаруживаемые при этом ошибки не будут проявляться при выполнении отлаживаемой программы ни при каких входных данных. Однако так поступать не рекомендуется, так как при изменениях отлаживаемой программы (например, при сопровождении ПС) не использованные для тестирования отлаживаемого модуля состояния информационной среды могут уже возникать, что требует дополнительного тестирования этого модуля (а этого при рациональной организации отладки можно было бы не делать, если сам данный модуль не изменялся). Для осуществления тестирования отлаживаемого модуля в указанных ситуациях иногда используют подходящие имитаторы, чтобы создать требуемое состояние информационной среды. Чаще же пользуются модифицированным вариантом нисходящего тестирования, при котором отлаживаемые модули перед их интеграцией предварительно тестируются отдельно (в этом случае в окружении отлаживаемого модуля появляется ведущий отладочный модуль, наряду с имитаторами модулей, к которым может обращаться отлаживаемый модуль). Однако, представляется более целесообразной другая модификация нисходящего тестирования:

после завершения нисходящего тестирования отлаживаемого модуля для достижимых тестовых состояний информационной среды следует его отдельно протестировать для остальных требуемых состояний информационной среды.

Часто применяют также комбинацию восходящего и нисходящего тестирования, которую называют методом *сандвича* [10.1]. Сущность этого метода заключается в одновременном осуществлении как восходящего, так и нисходящего тестирования, пока эти два процесса тестирования не встретятся на каком-либо модуле где-то в середине структуры отлаживаемой программы. Этот метод при разумном порядке тестирования позволяет воспользоваться достоинствами как восходящего, так и нисходящего тестирования, а также в значительной степени нейтрализовать их недостатки.

Весьма важным при автономной отладке является тестирование сопряжения модулей. Дело в том, что спецификация каждого модуля программы, кроме головного, используется в этой программы в двух ситуациях: во-первых, при разработке текста (иногда говорят: тела) этого модуля и, во-вторых, при написании обращения к этому модулю в других модулях программы. И в том, и в другом случае в результате ошибки может быть нарушено требуемое соответствие заданной спецификации модуля. Такие ошибки требуется обнаруживать и устранять. Для этого и предназначено тестирование сопряжения модулей. При нисходящем тестировании тестирование сопряжения осуществляется попутно каждым пропускаемым тестом, что считают достоинством нисходящего тестирования. При восходящем тестировании обращение к отлаживаемому модулю производится не из модулей отлаживаемой программы, а из ведущего отладочного модуля. В связи с этим существует опасность, что последний модуль может приспособиться к некоторым "заблуждениям" отлаживаемого модуля. Поэтому, приступая (в процессе интеграции программы) к отладке нового модуля, приходится тестировать каждое обращение к ранее отлаженному модулю с целью обнаружения несогласованности этого обращения с телом соответствующего модуля (и не исключено, что виноват в этом ранее отлаженный модуль). Таким образом, приходится частично повторять в новых условиях тестирование ранее отлаженного модуля, при этом возникают те же трудности, что и при нисходящем тестировании.

Автономное тестирование модуля целесообразно осуществлять в четыре последовательно выполняемых шага [10.1].

Шаг 1. На основании спецификации отлаживаемого модуля подготовьте тесты для каждой возможности и каждой ситуации,

для каждой границы областей допустимых значений всех входных данных, для каждой области изменения данных, для каждой области недопустимых значений всех входных данных и каждого недопустимого условия.

Шаг 2. Проверьте текст модуля, чтобы убедиться, что каждое направление любого разветвления будет пройдено хотя бы на одном тесте. Добавьте недостающие тесты.

Шаг 3. Проверьте текст модуля, чтобы убедиться, что для каждого цикла существуют тесты, обеспечивающие, по крайней мере, три следующие ситуации: тело цикла не выполняется ни разу, тело цикла выполняется один раз и тело цикла выполняется максимальное число раз. Добавьте недостающие тесты.

Шаг 4. Проверьте текст модуля, чтобы убедиться, что существуют тесты, проверяющие чувствительность к отдельным особым значениям входных данных. Добавьте недостающие тесты.

10.5. Комплексная отладка программного средства.

Как уже было сказано выше, при комплексной отладке тестируется ПС в целом, причем тесты готовятся по каждому из документов ПС [10.8]. Тестирование этих документов производится, как правило, в порядке, обратном их разработке. Исключение составляет лишь тестирование документации по применению, которая разрабатывается по внешнему описанию параллельно с разработкой текстов программ - это тестирование лучше производить после завершения тестирования внешнего описания. Тестирование при комплексной отладке представляет собой применение ПС к конкретным данным, которые в принципе могут возникнуть у пользователя (в частности, все тесты готовятся в форме, рассчитанной на пользователя), но, возможно, в моделируемой (а не в реальной) среде. Например, некоторые недоступные при комплексной отладке устройства ввода и вывода могут быть заменены их программными имитаторами.

Тестирование архитектуры ПС. Целью тестирования является поиск несоответствия между описанием архитектуры и совокупностью программ ПС. К моменту начала тестирования архитектуры ПС должна быть уже закончена автономная отладка каждой подсистемы. Ошибки реализации архитектуры могут быть связаны, прежде всего, с взаимодействием этих подсистем, в частности, с реализацией архитектурных функций (если они есть). Поэтому хотелось бы проверить все пути взаимодействия между подсистемами ПС. При этом желательно хотя бы протестировать все цепочки выполнения подсистем без повторного вхождения последних. Если заданная архитектура представляет ПС в качестве

малой системы из выделенных подсистем, то число таких цепочек будет вполне обозримо.

Тестирование внешних функций. Целью тестирования является поиск расхождений между функциональной спецификацией и совокупностью программ ПС. Несмотря на то, что все эти программы автономно уже отлажены, указанные расхождения могут быть, например, из-за несоответствия внутренних спецификаций программ и их модулей (на основании которых производилось автономное тестирование) функциональной спецификации ПС. Как правило, тестирование внешних функций производится так же, как и тестирование модулей на первом шаге, т.е. как черного ящика.

Тестирование качества ПС. Целью тестирования является поиск нарушений требований качества, сформулированных в спецификации качества ПС. Это наиболее трудный и наименее изученный вид тестирования. Ясно лишь, что далеко не каждый примитив качества ПС может быть испытан тестированием (об оценке качества ПС см. лекцию 14). Завершенность ПС проверяется уже при тестировании внешних функций. На данном этапе тестирование этого примитива качества может быть продолжено, если требуется получить какую-либо вероятностную оценку степени надежности ПС. Однако, методика такого тестирования еще требует своей разработки. Могут тестироваться такие примитивы качества, как точность, устойчивость, защищенность, временная эффективность, в какой-то мере - эффективность по памяти, эффективность по устройствам, расширяемость и, частично, независимость от устройств. Каждый из этих видов тестирования имеет свою специфику и заслуживает отдельного рассмотрения. Мы здесь ограничимся лишь их перечислением. Легкость применения ПС (критерий качества, включающий несколько примитивов качества, см. лекцию 4) оценивается при тестировании документации по применению ПС.

Тестирование документации по применению ПС. Целью тестирования является поиск несогласованности документации по применению и совокупностью программ ПС, а также выявление неудобств, возникающих при применении ПС. Этот этап непосредственно предшествует подключению пользователя к завершению разработки ПС (тестированию определения требований к ПС и аттестации ПС), поэтому весьма важно разработчикам сначала самим воспользоваться ПС так, как это будет делать пользователь [10.1]. Все тесты на этом этапе готовятся исключительно на основании только документации по применению ПС. Прежде всего, должны тестироваться возможности ПС как это делалось при тестировании внешних функций, но только на

основании документации по применению. Должны быть протестированы все неясные места в документации, а также все примеры, использованные в документации. Далее тестируются наиболее трудные случаи применения ПС с целью обнаружить нарушение требований относительности легкости применения ПС.

Тестирование определения требований к ПС. Целью тестирования является выяснение, в какой мере ПС не соответствует предъявленному определению требований к нему. Особенность этого вида тестирования заключается в том, что его осуществляет организация-покупатель или организация-пользователь ПС [10.1] как один из путей преодоления барьера между разработчиком и пользователем (см. лекцию 3). Обычно это тестирование производится с помощью контрольных задач - типовых задач, для которых известен результат решения. В тех случаях, когда разрабатываемое ПС должно придти на смену другой версии ПС, которая решает хотя бы часть задач разрабатываемого ПС, тестирование производится путем решения общих задач с помощью как старого, так и нового ПС (с последующим сопоставлением полученных результатов). Иногда в качестве формы такого тестирования используют *опытную* эксплуатацию ПС - ограниченное применение нового ПС с анализом использования результатов в практической деятельности. По существу, этот вид тестирования во многом перекликается с испытанием ПС при его аттестации (см. лекцию 14), но выполняется до аттестации, а иногда и вместо аттестации.

Упражнения к лекции 10.

10.1. Что такое отладка программного средства?
10.2. Что такое тестирование программного средства?
10.3. Что такое автономная отладка программного средства?
10.4. Что такое комплексная отладка программного средства?
10.5. Что такое ведущий отладочный модуль?
10.6. Что такое отладочный имитатор программного модуля?

Литература к лекции 10.

10.1. Г. Майерс. Надежность программного обеспечения. - М.: Мир, 1980. - С. 171-262.
10.2. Д. Ван Тассел. Стиль, разработка, эффективность, отладка и испытание программ. - М.: Мир, 1985. - С. 179-295.
10.3. Дж. Хьюз, Дж. Мичтом. Структурный подход к программированию. - М.: Мир, 1980. - С. 254-268.
10.4. Дж. Фокс. Программное обеспечение и его разработка. - М.: Мир, 1985. - С. 227-241.

10.5. М. Зелковиц, А. Шоу, Дж. Гэннон. Принципы разработки программного обеспечения. - М.: Мир, 1982. - С. 105-116.
10.6. Ю.М. Безбородов. Индивидуальная отладка программ. - М.: Наука, 1982. - С. 9-79.
10.7. В.В. Липаев. Тестирование программ. - М.: Радио и связь, 1986. - С. 15-47.
10.8. Е.А. Жоголев. Введение в технологию программирования (конспект лекций). - М.: "ДИАЛОГ-МГУ", 1994.
10.9. Э. Дейкстра. Заметки по структурному программированию / У. Дал, Э. Дейкстра, К. Хоор. Структурное программирование. - М.: Мир, 1975. - С. 7-13.