Практическое занятие № 5

Организация тестирования ПС

1. Тестирование модулей

Важным аспектом тестирования после проектирования тестов является последовательность слияния всех модулей в систему или программу. Эта сторона вопроса обычно не получает достаточного внимания и часто рассматривается слишком поздно. Выбор этой последовательности, однако, является одним из самых жизненно важных решений, принимаемых на этапе тестирования, поскольку он определяет форму, в которой записываются тесты, типы необходимых инструментов тестирования, последовательность программирования модулей, а также тщательность и экономичность всего этапа тестирования. По этой причине такое решение должно приниматься на уровне проекта в целом и на достаточно ранней его стадии.

Тестирование модулей (или блоков) представляет собой процесс тестирования отдельных подпрограмм или процедур программы. Здесь подразумевается, что, прежде чем начинать тестирование программы в целом, следует протестировать отдельные небольшие модули, образующие эту программу. Такой подход мотивируется тремя причинами. *Во-первых,* появляется возможность управлять комбинаторикой тестирования, поскольку первоначально внимание концентрируется на небольших модулях программы. *Во-вторых*, облегчается задача отладки программы, т.е. обнаружение места ошибки и исправление текста программы. *В-третьих,* допускается параллелизм, что позволяет одновременно тестировать несколько модулей.

Цель тестирования модулей — сравнение функций, реализуемых модулем, со спецификациями его функций или интерфейса.

Тестирование модулей в основном ориентировано на принцип «белого ящика». Это объясняется прежде всего тем, что принцип «белого ящика»

труднее реализовать при переходе в последующем к тестированию более крупных единиц, например, программ в целом. Кроме того, последующие этапы тестирования ориентированы на обнаружение ошибок различного типа, т. е. ошибок, не обязательно связанных с логикой программы, а возникающих, например, из-за несоответствия программы требованиям пользователя.

Имеется большой выбор возможных подходов, которые могут быть использованы для слияния модулей в более крупные единицы. В большинстве своем они могут рассматриваться как варианты шести основных подходов, описанных ниже.

1.1 Пошаговое тестирование

Реализация процесса тестирования модулей опирается на два ключевых положения: построение эффективного набора тестов и выбор способа, посредством которого модули комбинируются при построении из них рабочей программы. Второе положение является важным, так как оно задает форму написания тестов модуля, типы средств, используемых при тестировании, порядок кодирования и тестирования модулей, стоимость генерации тестов и стоимость отладки. Рассмотрим два подхода к комбинированию модулей: пошаговое и монолитное тестирование.

Возникает вопрос: «Что лучше — выполнить по отдельности тестирование каждого модуля, а затем, комбинируя их, сформировать рабочую программу или же каждый модуль для тестирования подключать к набору ранее оттестированных модулей?».

Первый подход обычно называют *монолитным методом,* или *методом «большого удара»,* при тестировании и сборке программы;

Второй подход известен как *пошаговый метод* тестирования или сборки.

Метод пошагового тестирования предполагает, что модули тестируются не изолированно друг от друга, а подключаются поочередно для выполнения теста к набору уже ранее оттестированных модулей.

Пошаговый процесс продолжается до тех пор, пока к набору оттестированных модулей не будет подключен последний модуль.

Детального разбора обоих методов мы делать не будем, приведем лишь некоторые общие выводы.

Монолитное тестирование требует больших затрат труда. При пошаговом же тестировании «снизу-вверх» затраты труда сокращаются.

Расход машинного времени при монолитном тестировании меньше.

1. Использование монолитного метода предоставляет большие возможности для параллельной организации работы на начальной фазе тестирования (тестирования всех модулей одновременно). Это положение может иметь важное значение при выполнении больших проектов, в которых много модулей и много исполнителей, поскольку численность персонала, участвующего в проекте, максимальна на начальной фазе.

2. При пошаговом тестировании раньше обнаруживаются ошибки в интерфейсах между модулями, поскольку раньше начинается сборка программы. В противоположность этому при монолитном тестировании модули «не видят друг друга» до последней фазы процесса тестирования.

5. Отладка программ при пошаговом тестировании легче. Если есть ошибки в межмодульных интерфейсах, а обычно так и бывает, то при монолитном тестировании они могут быть обнаружены лишь тогда, когда собрана вся программа. В этот момент локализовать ошибку довольно трудно, поскольку она может находиться в любом месте программы. Напротив, при пошаговом тестировании ошибки такого типа в основном связаны с тем модулем, который подключается последним.

6. Результаты пошагового тестирования более совершенны.

В заключение отметим, что п. 1, 4, 5, 6 демонстрируют преимущества пошагового тестирования, а п. 2 и 3 — его недостатки. Поскольку для

современного этапа развития вычислительной техники характерны тенденции к уменьшению стоимости аппаратуры и увеличению стоимости труда, последствия ошибок в математическом обеспечении весьма серьезны, а стоимость устранения ошибки тем меньше, чем раньше она обнаружена; преимущества, указанные в п. 1, 4, 5, 6, выступают на первый план. В то же время ущерб, наносимый недостатками (п. 2 и 3), невелик. Все это позволяет нам сделать вывод, что *пошаговое тестирование является предпочтительным.*

Убедившись в преимуществах пошагового тестирования перед монолитным, исследуем две возможные стратегии тестирования: *нисходящее* и *восходящее.* Прежде всего внесем ясность в терминологию.

Во-первых, термины «нисходящее тестирование», «нисходящая разработка», «нисходящее проектирование» часто используются как синонимы. Действительно, термины «нисходящее тестирование» и «нисходящая разработка» являются синонимами (в том смысле, что они подразумевают определенную стратегию при тестировании и создании текстов модулей), но нисходящее проектирование — это совершенно иной и независимый процесс. Программа, спроектированная нисходящим методом, может тестироваться и нисходящим, и восходящим методами.

Во-вторых, восходящая разработка, или тестирование, часто отождествляется с монолитным тестированием. Это недоразумение возникает из-за того, что начало восходящего тестирования идентично монолитному при тестировании нижних или терминальных модулей. Но выше мы показали, что восходящее тестирование на самом деле представляет собой пошаговую стратегию.

1.2 Восходящее тестирование

При восходящем подходе программа собирается и тестируется «снизу вверх». Только модули самого нижнего уровня («терминальные» модули;

модули, не вызывающие других модулей) тестируются изолированно, автономно. После того как тестирование этих модулей завершено, вызов их должен быть так же надежен, как вызов встроенной функции языка или оператор присваивания. Затем тестируются модули, непосредственно вызывающие уже проверенные. Эти модули более высокого уровня тестируются не автономно, а вместе с уже проверенными модулями более низкого уровня. Процесс повторяется до тех пор, пока не будет достигнута вершина. Здесь завершается и тестирование модулей, и тестирование сопряжений программы.

При восходящем тестировании для каждого модуля необходим *драйвер:* нужно подавать тесты в соответствии с сопряжением тестируемого модуля. Одно из возможных решений — написать для каждого модуля небольшую ведущую программу. Тестовые данные представляются как «встроенные» непосредственно в эту программу переменные и структуры данных, и она многократно вызывает тестируемый модуль, с каждым вызовом передавая ему новые тестовые данные. Имеется и лучшее решение: воспользоваться программой тестирования модулей — это инструмент тестирования, позволяющий описывать тесты на специальном языке и избавляющий от необходимости писать драйверы.

Здесь отсутствуют проблемы, связанные с невозможностью или трудностью создания всех тестовых ситуаций, характерные для нисходящего тестирования. Драйвер как средство тестирования применяется непосредственно к тому модулю, который тестируется, где нет промежуточных модулей, которые следует принимать во внимание. Анализируя другие проблемы, возникающие при нисходящем тестировании, можно заметить, что при восходящем тестировании невозможно принять неразумное решение о совмещении тестирования с проектированием программы, поскольку нельзя начать тестирование до тех пор, пока не спроектированы модули нижнего уровня. Не существует также и трудностей

с незавершенностью тестирования одного модуля при переходе к тестированию другого, потому что при восходящем тестировании с применением нескольких версий заглушки нет сложностей с представлением тестовых данных.

1.3 Нисходящее тестирование

Нисходящее тестирование (называемое также нисходящей разработкой) не является полной противоположностью восходящему, но в первом приближении может рассматриваться как таковое. При нисходящем подходе программа собирается и тестируется «сверху вниз». Изолированно тестируется только головной модуль. После того как тестирование этого модуля завершено, с ним соединяются (например, редактором связей) один за другим модули, непосредственно вызываемые им, и тестируется полученная комбинация. Процесс повторяется до тех пор, пока не будут собраны и проверены все модули.

При этом подходе немедленно возникают два вопроса: 1. «Что делать, когда тестируемый модуль вызывает модуль более низкого уровня (которого в данный момент еще не существует)?» и 2. «Как подаются тестовые данные?»

Ответ на первый вопрос состоит в том, что для имитации функций недостающих модулей программируются модули - *заглушки,* которые моделируют функции отсутствующих модулей.

Интересен и второй вопрос: в какой форме готовятся тестовые данные и как они передаются программе? Если бы головной модуль содержал все нужные операции ввода и вывода, ответ был бы прост: тесты пишутся в виде обычных для пользователей внешних данных и передаются программе через выделенные ей устройства ввода. Так, однако, случается редко. В хорошо спроектированной программе физические операции ввода-вывода выполняются на нижних уровнях структуры, поскольку физический

ввод-вывод — абстракция довольно низкого уровня. Поэтому для того, чтобы решить проблему экономически эффективно, модули добавляются не в строго нисходящей последовательности (все модули одного горизонтального уровня, затем модули следующего уровня), а таким образом, чтобы *обеспечить функционирование операций физического ввода-вывода как можно быстрее.* Когда эта цель достигнута, нисходящее тестирование получает значительное преимущество: все дальнейшие тесты готовятся в той же форме, которая рассчитана на пользователя.

Нисходящий метод имеет как достоинства, так и недостатки по сравнению с восходящим. Самое значительное достоинство — то, что этот метод совмещает тестирование модуля, тестирование сопряжений и частично тестирование внешних функций. С этим же связано другое его достоинство: когда модули ввода-вывода уже подключены, тесты можно готовить в удобном виде. Нисходящий подход выгоден также в том случае, когда есть сомнения относительно осуществимости программы в целом или, когда в проекте программы могут оказаться серьезные дефекты.

Преимуществом нисходящего подхода очень часто считают отсутствие необходимости в драйверах; вместо драйверов вам просто следует написать «заглушки».

Нисходящий метод тестирования имеет, к сожалению, некоторые недостатки. Основным из них является то, что модуль редко тестируется досконально сразу после его подключения. Дело в том, что основательное тестирование некоторых модулей может потребовать крайне изощренных заглушек. Программист часто решает не тратить массу времени на их программирование, а вместо этого пишет простые заглушки и проверяет лишь часть условий в модуле. Он, конечно, собирается вернуться и закончить тестирование рассматриваемого модуля позже, когда уберет заглушки. Такой план тестирования — определенно не лучшее решение, поскольку об отложенных условиях часто забывают.

Второй тонкий недостаток нисходящего подхода состоит в том, что он может породить веру в возможность начать программирование и тестирование верхнего уровня программы до того, как вся программа будет полностью спроектирована. Эта идея на первый взгляд кажется экономичной, но обычно дело обстоит совсем наоборот. Большинство опытных проектировщиков признает, что проектирование программы — процесс итеративный. Редко первый проект оказывается совершенным. Нормальный стиль проектирования структуры программы предполагает по окончании проектирования нижних уровней вернуться назад и подправить верхний уровень, внеся в него некоторые усовершенствования или исправляя ошибки, либо иногда даже выбросить проект и начать все сначала, потому что разработчик внезапно увидел лучший подход. Если же головная часть программы уже запрограммирована и оттестирована, то возникает серьезное сопротивление любым улучшениям ее структуры. В конечном итоге за счет таких улучшений обычно можно сэкономить больше, чем те несколько дней или недель, которые рассчитывает выиграть проектировщик, приступая к программированию слишком рано.

1.4 Метод «большого скачка»

Вероятно, самый распространенный подход к интеграции модулей — метод «большого скачка». В соответствии с этим методом каждый модуль тестируется автономно. По окончании тестирования модулей они интегрируются в систему все сразу.

Метод «большого скачка» по сравнению с другими подходами имеет много недостатков и мало достоинств. Заглушки и драйверы необходимы для каждого модуля. Модули не интегрируются до самого последнего момента, а это означает, что в течение долгого времени серьезные ошибки в сопряжениях могут остаться необнаруженными.

Если программа мала (как, например, программа загрузчика) и хорошо

спроектирована, метод «большого скачка» может оказаться приемлемым. Однако для крупных программ метод «большого скачка» обычно губителен.

1.5 Метод сандвича

Тестирование методом сандвича представляет собой компромисс между восходящим и нисходящим подходами. Здесь делается попытка воспользоваться достоинствами обоих методов, избежав их недостатков.

При использовании этого метода одновременно начинают восходящее и нисходящее тестирование, собирая программу как снизу, так и сверху и встречаясь в конце концов где-то в середине. Точка встречи зависит от конкретной тестируемой программы и должна быть заранее определена при изучении ее структуры. Например, если разработчик может представить свою систему в виде уровня прикладных модулей, затем уровня модулей обработки запросов, затем уровня примитивных функций, то он может решить применять нисходящий метод на уровне прикладных модулей (программируя заглушки вместо модулей обработки запросов), а на остальных уровнях применить восходящий метод.

Метод сандвича сохраняет такое достоинство нисходящего и восходящего подходов, как начало интеграции системы на самом раннем этапе. Поскольку вершина программы вступает в строй рано, мы, как в нисходящем методе, уже на раннем этапе получаем работающий каркас программы. Так как нижние уровни программы создаются восходящим методом, снимаются проблемы нисходящего метода, которые были связаны с невозможностью тестировать некоторые условия в глубине программы.

1.6 Модифицированный метод сандвича

При тестировании методом сандвича возникает та же проблема, что и при нисходящем подходе, хотя здесь она стоит не так остро. Проблема эта в том, что невозможно досконально тестировать отдельные модули.

Восходящий этап тестирования по методу сандвича решает эту проблему для модулей нижних уровней, но она может по-прежнему оставаться открытой для нижней половины верхней части программы. В модифицированном методе сандвича нижние уровни также тестируются строго снизу-вверх. А модули верхних уровней сначала тестируются изолированно, а затем собираются нисходящим методом. Таким образом,

модифицированный метод сандвича также представляет собой компромисс между восходящим и нисходящим подходами.

2. Комплексное тестирование

Комплексное тестирование, вероятно, самая непонятная форма тестирования. Во всяком случае комплексное тестирование не является тестированием всех функций полностью собранной системы; тестирование такого типа называется *тестированием внешних функций.* Комплексное тестирование — процесс поисков несоответствия системы ее исходным целям. Элементами, участвующими в комплексном тестировании, служат сама система, описание целей продукта и вся документация, которая будет поставляться вместе с системой. Внешние спецификации, которые были ключевым элементом тестирования внешних функций, играют лишь незначительную роль в комплексном тестировании.

Часть аргументов в пользу этого должна быть уже очевидной: измеримые цели необходимы, чтобы определить правила для процессов проектирования. Остальные соображения должны проясниться сейчас. Если цели сформулированы, например, в виде требования, чтобы система была достаточно быстрой, вполне надежной, и чтобы в разумных пределах обеспечивалась безопасность, тогда нет способа определить при тестировании, в какой степени система достигает своих целей.

Если вы не сформулировали цели вашего продукта или если эти цели неизмеримы, вы не можете выполнить комплексное тестирование.

Комплексное тестирование может быть процессом и контроля, и испытаний. Процессом испытаний оно является тогда, когда выполняется в реальной среде пользователя или в обстановке, которая специально создана так, чтобы напоминать среду пользователя. Однако такая роскошь часто недоступна по ряду причин, и в подобных случаях комплексное тестирование системы является процессом контроля (т.е. выполняется в имитируемой, или тестовой, среде). Например, в случае бортовой вычислительной системы космического корабля или системы противоракетной защиты вопрос о реальной среде (запуск настоящего космического корабля или выстрел настоящей ракетой) обычно не стоит. Кроме того, как мы увидим дальше, некоторые типы комплексных тестов не осуществимы в реальной обстановке по экономическим соображениям, и лучше всего выполнять их в моделируемой среде.

2.1 Проектирование комплексного теста

Комплексное тестирование — наиболее творческий из всех обсуждавшихся до сих пор видов тестирования. Разработка хороших комплексных тестов требует часто даже больше изобретательности, чем само проектирование системы. Здесь нет простых рекомендаций типа тестирования всех ветвей или построения функциональных диаграмм. Однако следующие 15 пунктов дают некоторое представление о том, какие виды тестов могут понадобиться (рис. 1).

1. *Тестирование стрессов.* Распространенный недостаток больших систем состоит в том, что они функционируют как будто бы нормально при слабой или умеренной нагрузке, но выходят из строя при большой нагрузке и в стрессовых ситуациях реальной среды. Тестирование стрессов представляет собой попытки подвергнуть систему крайнему «давлению», например, попытку одновременно подключить к системе разделения времени 100 терминалов, насытить банковскую систему мощным потоком

входных сообщений или систему управления — процессами аварийных сигналов от всех ее процессов.

Одна из причин, по которой тестирование стрессов опускают (кроме очевидных логических проблем, рассматриваемых в следующем разделе), состоит в том, что персонал, занимающийся тестированием, хотя и признает потенциальную пользу таких тестов, считает, что столь жесткие стрессовые ситуации никогда не возникнут в реальной среде. Это предположение редко оправдывается. Например, бывают случаи, когда все пользователи системы разделения времени пытаются подключиться в одно и то же время

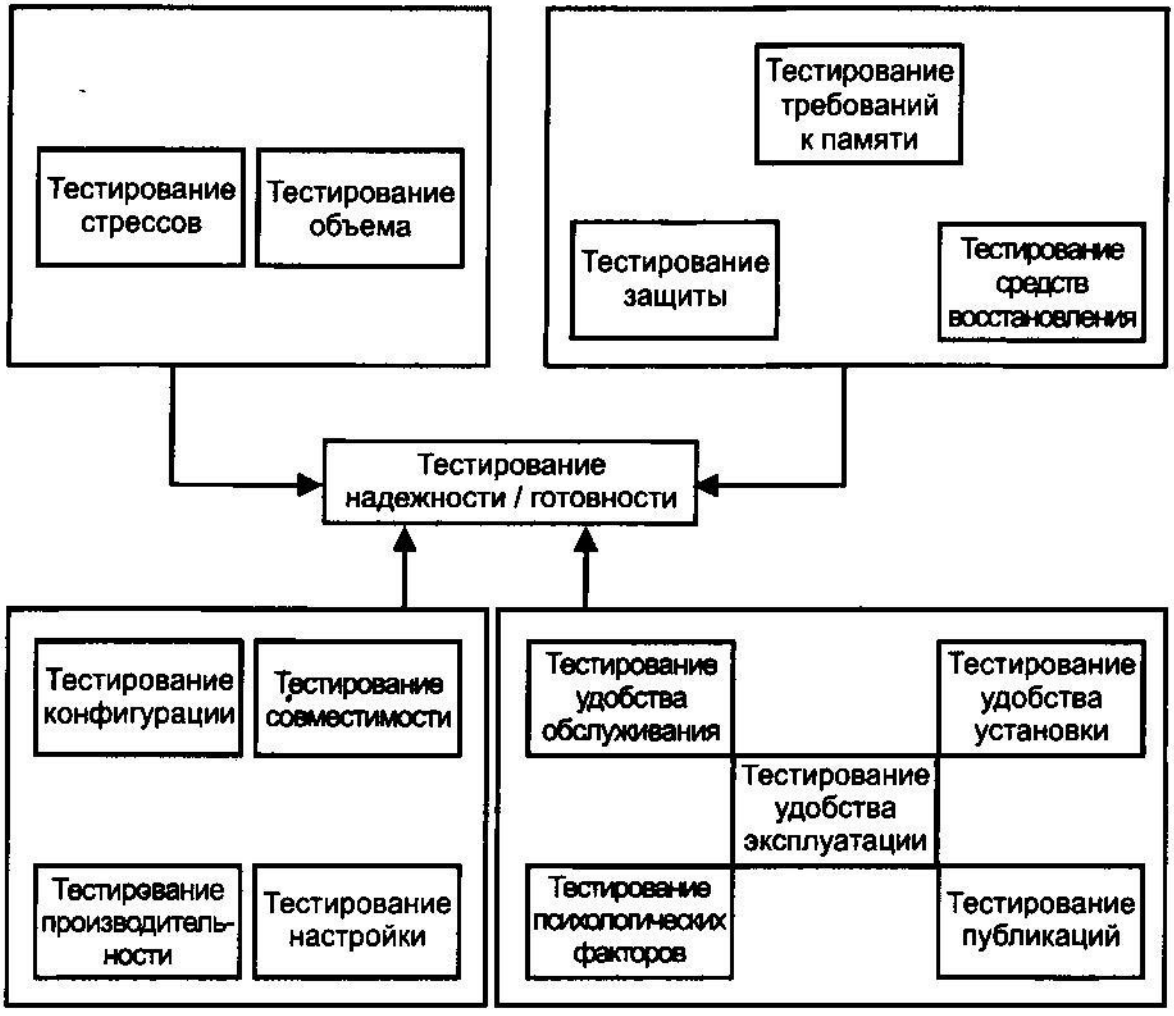

Рисунок 1 – Схема проектирования комплексного теста

(например, когда произошел отказ системы на 1-2 минуты и система только

что восстановлена). У банковских систем, обслуживающих терминалы покупателей, бывают пиковые нагрузки в первые часы работы магазинов и в час обеденного перерыва.

2. *Тестирование объема*. В то время как при тестировании стрессов делается попытка подвергнуть систему серьезным нагрузкам в короткий интервал времени, тестирование объема представляет собой попытку предъявить системе большие объемы данных в течение более длительного времени. На вход компилятора следует подать до нелепости громадную программу. Программа обработки текстов — огромный документ. Очередь заданий операционной системы следует заполнить до предела. Цель тестирования объема — показать, что система или программа не может обрабатывать данные в количествах, указанных в их спецификациях.

3. *Тестирование конфигурации.* Многие системы, например, операционные системы или системы управления файлами, обеспечивают работу различных конфигураций аппаратуры и программного обеспечения. Число таких конфигураций часто слишком велико, чтобы можно было проверить все варианты. Однако следует тестировать по крайней мере максимальную и минимальную конфигурации. Система должна быть проверена со всяким аппаратным устройством, которое она обслуживает, или со всякой программой, с которой она должна взаимодействовать. Если сама программная система допускает несколько конфигураций (т.е. покупатель может выбрать только определенные части или варианты системы), должна быть тестирована каждая из них.

4. *Тестирование совместимости.* В большинстве своем разрабатываемые системы не являются совершенно новыми; они представляют собой улучшение прежних версий или замену устаревших систем. В таких случаях на систему, вероятно, накладывается дополнительное требование совместимости, в соответствии с которым взаимодействие пользователя с прежней версией должно полностью

сохраниться и в новой системе. Например, возможно, потребуется, чтобы в новом выпуске операционной системы язык управления заданиями, язык общения с терминалом и скомпилированные прикладные программы, использовавшиеся раньше, могли применяться без изменений. Такие требования совместимости следует тестировать. Как периодически подчеркивалось при разговоре обо всех других формах тестирования, цель при тестировании совместимости должна состоять в том, чтобы показать наличие несовместимости.

5. *Тестирование защиты.* Так как внимание к вопросам сохранения секретности в обществе возрастает, к большинству систем предъявляются определенные требования по обеспечению защиты от несанкционированного доступа. Например, операционная система должна устранить всякую возможность для программы пользователя увидеть данные или программу другого пользователя. Административная информационная система не должна позволять подчиненному получить сведения о зарплате тех, кто стоит выше его по служебной лестнице. Цель тестирования защиты — нарушить секретность в системе. Один из методов — нанять профессиональную группу «взломщиков», т. е. людей с опытом разрушения средств обеспечения защиты в системах. Для тестирования защиты важно построить такие тесты, которые нарушат программные средства защиты, например, механизм защиты памяти операционной системы или механизмы защиты данных системы управления базой данных.

Одним из путей разработки подобных тестов является изучение известных проблем защиты в этих системах и генерация тестов, которые позволяют проверить, как решаются аналогичные проблемы в тестируемой системе.

6. *Тестирование требований к памяти.* При проектировании многих систем ставятся цели, определяющие объем основной и вторичной памяти, которую системе разрешено использовать в различных условиях. С помощью специальных тестов нужно попытаться показать, что система этих

целей не достигает.

7. *Тестирование производительности.* При разработке многих программ ставится задача — обеспечить их производительность, или эффективность. Определяются такие характеристики, как время отклика и уровень пропускной способности при определенной нагрузке и конфигурации оборудования. Проверка системы в этих случаях сводится к демонстрации того, что данная программа не удовлетворяет поставленным целям. Поэтому необходимо разработать тесты, с помощью которых можно попытаться показать, что система не обладает требуемой производительностью.

8. *Тестирование настройки.* К сожалению, процедуры настройки многих систем сложны. Тестирование процесса настройки системы очень важно, поскольку одна из наиболее обескураживающих ситуаций, с которыми сталкивается покупатель, заключается в том, что он оказывается не в состоянии даже настроить новую систему.

9. *Тестирование надежности/готовности.* Конечно, назначение всех видов тестирования — повысить надежность тестируемой программы, но если в исходном документе, отражающем цели программы, есть особые указания, касающиеся надежности, можно разработать специальные тесты для ее проверки. Ключевой момент в комплексном тестировании заключается в попытке доказать, что система не удовлетворяет исходным требованиям к надежности (среднее время между отказами, количество ошибок, способность к обнаружению, исправлению ошибок и/или устойчивость к ошибкам и т. д.). Тестирование надежности крайне сложно, и
все же следует постараться тестировать как можно больше этих требований. Например, в систему можно намеренно внести ошибки (как аппаратные, так и программные), чтобы тестировать средства обнаружения, исправления и обеспечения устойчивости. Другие требования к надежности тестировать почти невозможно.

10. *Тестирование средств восстановления.* Важная составная часть требований к операционным системам, системам управления базами данных и системам передачи данных — обеспечение способности к восстановлению, например, восстановлению утраченных данных в базе данных или восстановлению после отказа в телекоммуникационной линии. Во многих проектах совершенно забывают о тестировании соответствующих средств. Лучше всего попытаться показать, что эти средства работают неправильно, при комплексном тестировании системы. Можно намеренно ввести в операционную систему программные ошибки, чтобы проверить, восстановится ли она после их устранения. Неисправности аппаратуры, например, ошибки устройств ввода-вывода и контроля на четность ячеек памяти, можно промоделировать. Ошибки в данных (помехи в линиях связи и неправильные значения указателей в базе данных) можно намеренно создать или промоделировать для анализа реакции на них системы.

11. *Тестирование удобства обслуживания.* Либо в требованиях к продукту, либо в требованиях к проекту должны быть перечислены задачи, определяющие удобство обслуживания (сопровождения) системы. Все сервисные средства системы нужно проверять при комплексном тестировании. Все документы, описывающие внутреннюю логику, следует проанализировать глазами обслуживающего персонала, чтобы понять, как быстро и точно можно указать причину ошибки, если известны только некоторые се симптомы. Все средства, обеспечивающие сопровождение и поставляемые вместе с системой, также должны быть проверены.

12. *Тестирование публикаций.* Проверка точности всей документации для пользователя является важной частью комплексного тестирования. Все комплексные тесты следует строить только на основе документации для пользователя. Ее проверяют при определении правильности представления предшествующих тестов системы.

Пользовательская документация должна быть и предметом инспекции при проверке ее на точность и ясность. Любые примеры, приведенные в документации, следует оформить как тест и подать на вход программы.

13. *Тестирование психологических факторов.* Хотя во время тестирования системы следует проверить и психологические факторы, эта сторона не так важна, как другие, потому что обычно при тестировании уже слишком поздно исправлять серьезные просчеты в таких вопросах. Однако мелкие недостатки могут быть обнаружены и устранены при тестировании системы. Например, может оказаться, что ответы или сообщения системы плохо сформулированы или ввод команды пользователя требует постоянных переключений верхнего и нижнего регистров.

14. *Тестирование удобства установки.* Процедуры установки (настройки) некоторых типов систем программного обеспечения весьма сложны. Тестирование подобных процедур является частью процесса тестирования системы.

15. *Тестирование удобства эксплуатации.* Не менее важным видом тестирования системы является попытка выявления психологических (пользовательских) проблем, или проблем удобства эксплуатации. Анализ психологических факторов является, к сожалению, до сих пор весьма субъективным, так как при производстве вычислительной техники уделялось недостаточное внимание изучению и определению психологических аспектов программных систем. Большинство систем обработки данных либо является компонентами более крупных систем, предполагающих деятельность человека, либо сами регламентируют такую деятельность во время своей работы. Нужно проверить, что вся эта деятельность (например, поведение оператора или пользователя за терминалом) удовлетворяет определенным условиям.

Не все из перечисленных 15 пунктов применимы к тестированию всякой системы (например, когда тестируется отдельная прикладная

программа), но тем не менее это перечень вопросов, которые разумно иметь в виду.

Основное правило при комплексном тестировании — все годится. Пишите разрушительные тесты, проверяйте все функциональные границы системы, пишите тесты, представляющие ошибки пользователя (например, оператора системы). Нужно, чтобы тестовик думал так же, как пользователь или покупатель, а это предполагает доскональное понимание того, для чего система будет применяться. Поэтому возникает вопрос: «Кто же должен выполнять комплексное тестирование и, в частности, кто должен проектировать тесты?» Во всяком случае этого не должны делать программисты или организации, ответственные за разработку системы.

Группа тестирования системы должна быть независимой организацией и должна включать профессиональных специалистов по комплексному тестированию систем; несколько пользователей, для которых система разрабатывалась; основных аналитиков и проектировщиков системы и, возможно, одного или двух психологов. Очень важно включить самих проектировщиков системы. Это не противоречит аксиоме, согласно которой невозможно тестировать свою собственную систему, поскольку система прошла через много рук после того, как ее описали архитектор или проектировщик. В действительности проектировщик и не пытается тестировать собственную систему; он ищет расхождения между окончательной версией и тем, что он первоначально, возможно год или два назад, имел в виду. Для комплексного тестирования желательно также иметь информацию о рынке или пользователях, уточняющую предположения о конфигурации и характере применения системы.

Комплексное тестирование системы — такая особая и такая важная работа, что в будущем возможно появление компаний, специализирующихся в основном на комплексном тестировании систем, разработанных другими.

Как уже упоминалось, компонентами комплексного теста являются исходные цели, документация, публикации для пользователей и сама система. Все комплексные тесты должны быть подготовлены на основе публикаций для пользователя (а не внешних спецификаций). Ко внешним спецификациям обращаться следует только для того, чтобы разбираться в противоречиях между системой и публикациями о ней.

По своей природе комплексные тесты никогда не сводятся к проверке отдельных функций системы. Они часто пишутся в форме сценариев, представляющих ряд последовательных действий пользователя. Например, один комплексный тест может представлять подключение терминала к системе, выдачу последовательно 10—20 команд и затем отключение от системы. Вследствие их особой сложности тесты системы состоят из нескольких компонентов: сценария, входных данных и ожидаемых выходных данных. В сценарии точно указываются действия, которые должны быть совершены во время выполнения теста.

2.2 Выполнение комплексного теста

При комплексном тестировании вся система рассматривается как «черный ящик»; в частности, несущественно, какие участки программы затрагивает комплексный тест.

Один из методов, позволяющих вовлечь в тестирование пользователей, — *опытная эксплуатация*. Для проведения опытной эксплуатации с одной или несколькими организациями пользователей заключаются контракты на установку у них созданной системы. Это часто выгодно обеим сторонам: организация-разработчик оповещается об ошибках в программном обеспечении, которые она не заметила сама, а организация-пользователь получает возможность изучить систему и экспериментировать с ней до того, как она станет доступной официально.

Второй полезный метод — использовать систему в

организации-изготовителе для внутренних нужд. Это можно сделать часто, но далеко не всегда (например, компании по разработке программного обеспечения негде использовать систему управления процессом очистки нефти). Так, производитель ЭВМ может установить новую операционную систему для опытной эксплуатации во всех своих внутренних вычислительных центрах, прежде чем начинать поставлять ее пользователям, а компания по разработке программного обеспечения может использовать свои собственные компиляторы, программы обработки текстов, начисления зарплаты и т. д.

Воспользуйтесь собственным продуктом, прежде чем передавать его другим.

При комплексном тестировании часто начинают с простых тестов, приберегая более сложные тесты к концу. Так делать не нужно, потому что комплексное тестирование приходится на самый конец цикла разработки, так что на отладку и исправление найденных ошибок остается мало времени. Поскольку сложные тесты часто обнаруживают более сложные для исправления ошибки, измените последовательность: начните с самых трудных тестов, а затем переходите к более простым.

Контрольные вопросы

1. Определение, необходимость проведения и цель тестирования модулей.
2. Сущность пошагового тестирования.
3. Преимущества пошагового тестирования.
4. Сущность восходящего тестирования.
5. Особенность проведения восходящего тестирования(драйвера)
6. Сущность нисходящего тестирования.
7. Особенность проведения нисходящего тестирования (заглушки)
8. Преимущества и недостатки нисходящего подхода к

тестированию.

9. Сущность метода «большого скачка»
10. Недостатки метода «большого скачка»
11. Сущность метода сандвича.
12. Достоинства и недостатки метода сандвича.
13. Сущность модифицированного метода сандвича.
14. Определение комплексного тестирования.
15. Сущность тестирования стрессов, тестирования объема, тестирования конфигурации, тестирования совместимости, тестирования защиты, тестирования требований к памяти, тестирования производительности, тестирования настройки, тестирования надежности/готовности, тестирования средств восстановления, тестирования удобства обслуживания, тестирования публикаций, тестирования психологических факторов, тестирования удобства установки, тестирования удобства эксплуатации
16. Способы выполнения комплексного теста?