# **1. Общие понятия тестирования программного обеспечения**

Тестирование ПО – это процесс выполнения его программ на некотором наборе данных, для которых заранее известен результат, а также правило поведения этих программ.

Тестирование проводится уже с программой, не имеющих ошибок. Целью тестирования является получение результатов по конкретным данным, а также контроль качества программы и убедиться в правильности работы ПО.

Тестирование должно включать проверку всех ветвей программы и должно включать минимальный набор примеров. При тестировании нельзя предусмотреть всевозможные ситуации, но надо добиваться определенных результатов.

Необходимо тщательно подбирать тест не только для правильных (предусмотренных) входных данных, но и для неправильных (непредусмотренных);

Следует сохранять использованные тесты (для повышения эффективности повторного тестирования программы после ее модификации или установки у заказчика).

Относительно тестовых данных можно выделить следующие виды тестовых проверок:

1. Проверка в нормальных условиях

Данная проверка производится на основе данных, которые характерны для реальных условий функционирования программы.

Результат такого теста – показ правильных результатов для характерного набора данных.

2. Проверка в экстремальных условиях

Тестовые данные при этой проверке должны включать граничные значения области изменения входных переменных, которые должны восприниматься программой как правильные данные.

Здесь проводятся нулевые примеры: ввод цифр, имеющих нулевые значения, для символов - цепочка пробелов или нулей.

3. Проверка в исключительных ситуациях

Для данной проверки значения данных подбирают так, чтобы они лежали за пределами допустимой области изменения. Данные должны содержать пробелы, цифры и буквы в разной последовательности и сочетании.

## **Виды тестирования**

Существует два основных вида тестирования: функциональное и структурное. При функциональном тестировании программа рассматривается как "черный ящик" (то есть ее текст не используется). Происходит проверка соответствия поведения программы ее внешней спецификации. Возможно ли при этом полное тестирование программы? Очевидно, что критерием полноты тестирования в этом случае являлся бы перебор всех возможных значений входных данных, что невыполнимо.

Поскольку исчерпывающее функциональное тестирование невозможно, речь может идти о разработки методов, позволяющих подбирать тесты не "вслепую", а с большой вероятностью обнаружения ошибок в программе.

При структурном тестировании программа рассматривается как "белый ящик" (т.е. ее текст открыт для пользования). Происходит проверка логики программы. Полным тестированием в этом случае будет такое, которое приведет к перебору всех возможных путей на графе передач управления программы (ее управляющем графе). Даже для средних по сложности программ числом таких путей может достигать десятков тысяч. Если ограничиться перебором только линейных не зависимых путей, то и в этом случае исчерпывающее структурное тестирование практически невозможно, т. к. неясно, как подбирать тесты, чтобы обеспечить "покрытие" всех таких путей. Поэтому при структурном тестировании необходимо использовать другие критерии его полноты, позволяющие достаточно просто контролировать их выполнение, но не дающие гарантии полной проверки логики программы.

Но лаже если предположить, что удалось достичь полного структурного тестирования некоторой программы, в ней тем не менее, могут содержаться ошибки, т.к.:

1) программа может не соответствовать своей внешней спецификации, что в частности, может привести к тому, что в ее управляющем графе окажутся пропущенными некоторые необходимые пути;

2) не будут обнаружены ошибки, появление которых зависит от обрабатываемых данных (т.е. на одних исходных данных программа работает правильно; а на других – с ошибкой).

Таким образом, ни структурное, ни функциональное тестирование не может быть исчерпывающим.

Рассмотрим подробнее основные этапы тестирования программных комплексов.

В тестировании многомодульных программных комплексов можно выделить 4 этапа:

тестирование отдельных модулей;

совместное тестирование модулей;

тестирование функций программного комплекса (т.е. поиск различий между разработанной программой и ее внешней спецификацией);

тестирование всего комплекса в целом (т.е. поиск несоответствия созданного программного продукта, сформулированным ранее целям проектирования, отраженным обычно в техническом задании).

## **3. Виды тестирования**

## **3.1 Совместное тестирование модулей**

Известны два подхода к совместному тестированию модулей: пошаговое и монолитное тестирование.

При монолитном тестировании сначала по отдельности тестируются все модули программного комплекса, а затем все они объединяются в рабочую программу для комплексного тестирования.

При пошаговом тестировании каждый модуль для своего тестирования подключается к набору уже проверенных модулей.

В первом случае для автономного тестирования каждого модуля требуется модуль-драйвер (то есть вспомогательный модуль, имитирующий вызов тестируемого модуля) и один или несколько модулей - заглушек (то есть вспомогательных модулей, имитирующих работу модулей, вызываемых из тестируемого). При пошаговом тестировании модули проверяются не изолированно друг от друга, поэтому требуются либо только драйверы, либо только заглушки.

При сравнении пошагового и монолитного тестирования можно отметить следующие преимущества первого подхода:

1) меньшая трудоемкость (при монолитном тестировании требуются больше драйверов и заглушек, чем при пошаговом тестировании, когда требуются или только драйверы - если модули подключаются "снизу-вверх ", - или только заглушки - если модули подключаются "сверху-вниз");

2) более раннее обнаружение ошибок в интерфейсах между модулями (их сборка начинается раньше, чем при монолитном тестировании);

Есть преимущества и у монолитного тестирования:

– меньше расход машинного времени (хотя из-за большей сложности может потребоваться дополнительная проверка программы, и это преимущество будет сведено па нет);

– предоставляется больше возможностей для организации параллельной работы на начальном этапе тестирования.

В целом более целесообразным является выбор пошагового тестирования. При его использовании возможны две стратегии подключения модулей: нисходящая и восходящая.

## **3.2 Сверху-вниз (нисходящее тестирование)**

Нисходящее тестирование начинается с главного (или верхнего) модуля программы. Одна из основных проблем, возникающих при нисходящем тестировании – создание заглушек. Это тривиальная задача, т. к. как правило, недостаточно, чтобы в заглушке выполнялся вывод соответствующего информационного сообщения и возврат всегда одних и тех же значений выходных данных.

Другая проблема, которую необходимо решать при нисходящем тестировании. Форма представления тестов в программе, так как, как правило, главный модуль получает входные данные не непосредственно, а через специальные модули ввода, которые при тестировании в начале заменяются заглушками. Для передачи в главный модуль разных тестов нужно или иметь несколько разных заглушек, или записать эти тесты в файл во внешней памяти и с помощью заглушки считывать их.

Поскольку после тестирования главного модуля процесс проверки может продолжаться по-разному, следует придерживаться следующих правил:

а) модули, содержащие операции ввода-вывода, должны подключаться к тестированию как можно раньше;

б) критические (т.е. наиболее важные) для программы в целом модули также должны тестироваться в первую очередь.

Основные достоинства нисходящего тестирования:

– уже на ранней стадии тестирования есть возможность получить работающий вариант разрабатываемой программы;

– быстро могут быть выявлены ошибки, связанные с организацией взаимодействие с пользователем.

Кроме достоинств есть и проблемы, которые могут возникать при нисходящем тестировании:

– может оказаться трудным или даже невозможным построить один и тот же тест на входах разных модулей;

– не всегда окажется возможным легко оценить соответствие значений данных на входах разных модулей;

– результаты выполнения программы на построенном для проверки одного модуля тесте выводятся не им, а другими модулями, может оказаться трудным восстановление действительных результатов работы данного модуля; появляется соблазн совмещения нисходящего проектирования с тестированием, что, как правило, неразумно, т.к. проектирование - процесс итеративный и в нем неизбежен возврат на верхние уровни и исправление принятых ранее решений, что обесценивает результаты уже проведенного тестирования;

может возникнуть желание перейти к тестированию модуля следующего уровня до завершения тестирования предыдущего по объективным причинам (необходимости создания нескольких версий заглушек, использования модулями верхнего уровня ресурсов модулей нижних уровней).

## **3.3 Снизу-вверх (восходящее тестирование)**

При восходящем тестировании проверка программы начинается с терминальных модулей (т. е. тех, которые не вызывают не каких других модулей программы). Эти стратегия во многом противоположна нисходящему тестированию (в частности, преимущества становятся недостатками и наоборот).

Нет проблемы выбора следующего подключаемого модуля - учитывается лишь то, чтобы он вызывал только уже протестированные модули. В отличие от заглушек драйверы не должны иметь несколько версий, поэтому их разработка в большинстве случаев проще (кроме того, использование средств автоматизации и отладки облегчает создание как раз драйверов, а не заглушек).

Другие достоинства восходящего тестирования:

– поскольку нет промежуточных модулей (тестируемый модуль является для рабочего варианта программы модулем самого верхнего уровня), нет проблем, связанных с возможностью или трудностью задания тестов, нет возможности совмещения проектирования с тестированием; нет трудностей, вызывающих желание перейти к тестированию следующего модуля, не завершив проверки предыдущего. Основным недостатком восходящего тестирования является то, что проверка всей структуры разрабатываемого программного комплекса возможна только на завершающей стадии тестирования.

Хотя однозначного вывода о преимущества той или иной стратегии пошагового тестирования сделать нельзя (нужно учитывать конкретные характеристики тестируемой программы), в большинстве случаев более предпочтительным является восходящее тестирование.

## **4. Инструменты тестирования**

Под средствами тестирования понимают программы, которые помогают автоматизировать процесс испытаний. Пакет подпрограмм, входящих в некоторую программу.

А именно:

– генератор тестовых данных. Он формирует данные для использования в проверяемой программе.

– программы-утилиты.

– компаратор файлов - это программа, которая считывает два файла и выводит ка печать их различающие результаты.

– программы-профилировщики. Они предназначены для отладки в тех случаях, когда надо выявить те участки программы, которые не были использованы

– тестовый монитор - это программа, которая пересылает нужные данные т ими тестируемого модуля и накапливает выходные данные, выдаваемые на печать или записывающие в файл.