**Методы и средства организации тестирования.**

Тестирование программного кода - процесс выполнения программного кода, направленный на выявление существующих в нем дефектов. Под дефектом здесь понимается участок программного кода, выполнение которого при определенных условиях приводит к неожиданному поведению системы (т.е. поведению, не соответствующему требованиям). Неожиданное поведение системы может приводить к сбоям в ее работе и отказам, в этом случае говорят о существенных дефектах программного кода. Некоторые дефекты вызывают незначительные проблемы, не нарушающие процесс функционирования системы, но несколько затрудняющие работу с ней. В этом случае говорят о средних или малозначительных дефектах.

Задача тестирования при таком подходе - *определение* условий, при которых проявляются дефекты системы, и *протоколирование* этих условий. В задачи тестирования обычно не входит выявление конкретных дефектных участков программного кода и никогда не входит исправление дефектов - это задача отладки, которая выполняется по результатам тестирования системы.

Цель применения процедуры тестирования программного кода - *минимизация* количества дефектов (в особенности существенных) в конечном продукте. Тестирование само по себе не может гарантировать полного отсутствия дефектов в программном коде системы. Однако, в сочетании с процессами верификации и валидации, направленными на устранение противоречивости и неполноты проектной документации (в частности - требований на систему), грамотно организованное тестирование дает гарантию того, что система удовлетворяет требованиям и ведет себя в соответствии с ними во всех предусмотренных ситуациях.

При разработке систем повышенной надежности, например, авиационных, гарантии надежности достигаются с помощью четкой организации процесса тестирования, определения его связи с остальными процессами жизненного *цикла*, введения количественных характеристик, позволяющих оценивать успешность тестирования. При этом чем выше требования к надежности системы (ее уровень критичности), тем более жесткие требования предъявляются.

Таким образом, в первую *очередь* мы рассматриваем не конкретные результаты тестирования конкретной системы, а общую организацию процесса тестирования, используя подход "хорошо организованный процесс дает качественный результат". Такой подход является общим для многих международных и отраслевых стандартов качества, о которых более подробно будет рассказано в конце данного курса. Качество разрабатываемой системы при таком подходе является следствием организованного процесса разработки и тестирования, а не самостоятельным неуправляемым результатом.

Поскольку современные программные системы имеют весьма значительные размеры, при тестировании их программного кода используется метод функциональной декомпозиции. Система разбивается на отдельные модули (классы, пространства имен и т.п.), имеющие определенную требованиями функциональность и интерфейсы. После этого по отдельности тестируется каждый *модуль* - выполняется *модульное тестирование*. Затем происходит *сборка* отдельных модулей в более крупные конфигурации - выполняется *интеграционное тестирование*, и наконец, тестируется система в целом - выполняется *системное тестирование*.

С точки зрения программного кода, модульное, интеграционное и *системное тестирование* имеют много общего, поэтому пока основное внимание будет уделено модульному тестированию, особенности интеграционного и системного тестирования будут рассмотрены позднее.

В ходе модульного тестирования каждый *модуль* тестируется как на соответствие требованиям, так и на отсутствие проблемных участков программного кода, которые могут вызвать отказы и сбои в работе системы. Как правило, модули не работают вне системы - они принимают данные от других модулей, перерабатывают их и передают дальше. Для того, чтобы с одной стороны, изолировать *модуль* от системы и исключить влияние потенциальных ошибок системы, а с другой стороны - обеспечить *модуль* всеми необходимыми данными, используется тестовое окружение.

Задача тестового окружения - создать среду выполнения для модуля, эмулировать все внешние интерфейсы, к которым обращается *модуль*. Об особенностях организации тестового окружения пойдет речь далее в данной лекции.

Типичная процедура тестирования состоит в подготовке и выполнении тестовых примеров (также называемых просто тестами). Каждый тестовый пример проверяет одну "ситуацию" в поведении модуля и состоит из списка значений, передаваемых на вход модуля, описания запуска и выполнения переработки данных - тестового сценария и списка значений, которые ожидаются на выходе модуля в случае его корректного поведения. Тестовые сценарии составляются таким образом, чтобы исключить обращения к внутренним данным модуля, все взаимодействие должно происходить только через его внешние интерфейсы.

Выполнение тестового примера поддерживается тестовым окружением, которое включает в себя программную реализацию тестового сценария. Выполнение начинается с передачи модулю входных данных и запуска сценария. Реальные *выходные данные*, полученные от модуля в результате выполнения сценария, сохраняются и сравниваются с ожидаемыми. В случае их совпадения тест считается пройденным, в противном случае - не пройденным. Каждый не пройденный тест указывает на дефект либо в тестируемом модуле, либо в тестовом окружении, либо в описании теста.

Совокупность описаний тестовых примеров составляет тест-план - основной документ, определяющий процедуру тестирования программного модуля. Тест-план задает не только сами тестовые примеры, но и порядок их следования, который также может быть важен. Структура и особенности тест-планов, а также проблемы, связанные с порядком следования тестовых примеров, будут рассмотрены в следующих лекциях.

При тестировании часто бывает необходимо учитывать не только требования к системе, но и структуру программного кода тестируемого модуля. В этом случае тесты составляются таким образом, чтобы детектировать типичные ошибки программистов, вызванные неверной интерпретацией требований. Применяются проверки граничных условий, проверки классов эквивалентности. Отсутствие в системе возможностей, не заданных требованиями, гарантировано различными оценками покрытия программного кода тестами, т.е. оценками того, какой *процент* тех или иных языковых конструкций выполнен в результате выполнения всех тестовых примеров. Обо всем этом пойдет речь в завершение рассмотрения процесса тестирования программного кода.

**Методы тестирования**

**Черный ящик**

Основная идея в тестировании системы как черного ящика состоит в том, что все материалы, которые доступны тестировщику, - требования на систему, описывающие ее поведение, и сама система, работать с которой он может, только подавая на ее входы некоторые внешние воздействия и наблюдая на выходах некоторый результат. Все внутренние особенности реализации системы скрыты от тестировщика, - таким образом, система представляет собой "черный ящик", правильность поведения которого по отношению к требованиям и предстоит проверить.

С точки зрения программного кода черный ящик может представлять с собой набор классов (или модулей) с известными внешними интерфейсами, но недоступными исходными текстами.

Основная задача тестировщика для данного метода тестирования состоит в последовательной проверке соответствия поведения системы требованиям. Кроме того, тестировщик должен проверить работу системы в критических ситуациях - что происходит в случае подачи неверных входных значений. В идеальной ситуации все варианты критических ситуаций должны быть описаны в требованиях на систему и тестировщику остается только придумывать конкретные проверки этих требований. Однако в реальности в результате тестирования обычно выявляется два типа проблем системы.

1. Несоответствие поведения системы требованиям
2. Неадекватное поведение системы в ситуациях, не предусмотренных требованиями.

Отчеты об обоих типах проблем документируются и передаются разработчикам. При этом проблемы первого типа обычно вызывают изменение программного кода, гораздо реже - изменение требований. Изменение требований в данном случае может потребоваться из-за их противоречивости (несколько разных требований описывают разные модели поведения системы в одной и той же самой ситуации) или некорректности (требования не соответствуют действительности).

Проблемы второго типа однозначно требуют изменения требований ввиду их неполноты - в требованиях явно пропущена ситуация, приводящая к неадекватному поведению системы. При этом под неадекватным поведением может пониматься как полный крах системы, так и вообще любое поведение, не описанное в требованиях.

Тестирование черного ящика называют также тестированием по требованиям, т.к. это единственный источник информации для построения тест-плана.

**Стеклянный (белый) ящик**

При тестировании системы как стеклянного ящика тестировщик имеет доступ не только к требованиям к системе, ее входам и выходам, но и к ее внутренней структуре - видит ее программный код.

Доступность программного кода расширяет возможности тестировщика тем, что он может видеть соответствие требований участкам программного кода и определять тем самым, на весь ли программный код существуют требования. Программный код, для которого отсутствуют требования, называют кодом, не покрытым требованиями. Такой код является потенциальным источником неадекватного поведения системы. Кроме того, прозрачность системы позволяет углубить анализ ее участков, вызывающих проблемы - часто одна проблема нейтрализует другую, и они никогда не возникают одновременно.

**Тестирование моделей**

Тестирование моделей находится несколько в стороне от классических методов верификации программного обеспечения. Причина прежде всего в том, что объект тестирования - не сама система, а ее модель, спроектированная формальными средствами. Если оставить в стороне вопросы проверки корректности и применимости самой модели (считается, что ее корректность и соответствие исходной системе могут быть доказаны формальными средствами), то тестировщик получает в свое распоряжение достаточно мощный инструмент анализа общей целостности системы. На модели можно создать такие ситуации, которые невозможно создать в тестовой лаборатории для реальной системы. Работая с моделью программного кода системы, можно анализировать его свойства и такие параметры системы, как оптимальность алгоритмов или ее устойчивость.

Однако тестирование моделей не получило широкого распространения именно из-за трудностей, возникающих при разработке формального описания поведения системы. Одно из немногих исключений - системы связи, алгоритмический и математический аппарат которых достаточно хорошо проработан.

**Анализ программного кода (инспекции)**

Во многих ситуациях тестирование поведения системы в целом невозможно - отдельные участки программного кода могут никогда не выполняться, при этом они будут покрыты требованиями. Примером таких участков кода могут служить обработчики исключительных ситуаций. Если, например, два модуля передают друг другу числовые значения и функции проверки корректности значений работают в обоих модулях, то функция проверки модуля-приемника никогда не будет активизирована, т.к. все ошибочные значения будут отсечены еще в передатчике.

В этом случае выполняется ручной анализ программного кода на корректность, называемый также просмотрами или инспекциями кода. Если в результате инспекции выявляются проблемные участки, то информация об этом передается разработчикам для исправления наравне с результатами обычных тестов.