**компонентное/модульное/юнит тестирование**

**Модульное тестирование (юнит-тестирование).** Модульные тесты используются для тестирования какого-либо одного логически выделенного и изолированного элемента системы (отдельные методы класса или простая функция, subprograms, subroutines, классы или процедуры) в коде. Очевидно, что это тестирование методом белого ящика и чаще всего оно проводится самими разработчиками. Целью тестирования модуля является не демонстрация правильного функционирования модуля, а демонстрация наличия ошибки в модуле, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования. На уровне модульного тестирования проще всего обнаружить дефекты, связанные с алгоритмическими ошибками и ошибками кодирования алгоритмов, типа работы с условиями и счетчиками циклов, а также с использованием локальных переменных и ресурсов. Ошибки, связанные с неверной трактовкой данных, некорректной реализацией интерфейсов, совместимостью, производительностью и т.п. обычно пропускаются на уровне модульного тестирования и выявляются на более поздних стадиях тестирования. Изоляция тестируемого блока достигается с помощью заглушек (stubs), манекенов (dummies) и макетов (mockups). Являясь по способу исполнения структурным тестированием или тестированием "белого ящика", модульное тестирование характеризуется степенью, в которой тесты выполняют или покрывают логику программы (исходный текст). Тесты, связанные со структурным тестированием, строятся по следующим принципам:

* На основе анализа потока управления. В этом случае элементы, которые должны быть покрыты при прохождении тестов, определяются на основе структурных критериев тестирования С0, С1,С2. К ним относятся вершины, дуги, пути управляющего графа программы (УГП), условия, комбинации условий и т. п.
* На основе анализа потока данных, когда элементы, которые должны быть покрыты, определяются при помощи потока данных, т. е. информационного графа программы.

Тестирование на основе потока управления. Особенности использования структурных критериев тестирования С0,С1,С2 были рассмотрены ранее. К ним следует добавить критерий покрытия условий, заключающийся в покрытии всех логических (булевских) условий в программе. Критерии решений (ветвей - С1) и условий не заменяют друг друга, поэтому на практике используется комбинированный критерий покрытия условий/решений, совмещающий требования по покрытию и решений, и условий.

К популярным критериям относятся критерий покрытия функций программы, согласно которому каждая функция программы должна быть вызвана хотя бы один раз, и критерий покрытия вызовов, согласно которому каждый вызов каждой функции в программе должен быть осуществлен хотя бы один раз. Критерий покрытия вызовов известен также как критерий покрытия пар вызовов (call pair coverage).

Тестирование на основе потока данных. Этот вид тестирования направлен на выявление ссылок на неинициализированные переменные и избыточные присваивания (аномалий потока данных ). Предложенная там стратегия требовала тестирования всех взаимосвязей, включающих в себя ссылку (использование) и определение переменной, на которую указывает ссылка (т. е. требуется покрытие дуг информационного графа программы). Недостаток стратегии в том, что она не включает критерий С1, и не гарантирует покрытия решений.

Стратегия требуемых пар также тестирует упомянутые взаимосвязи. Использование переменной в предикате дублируется в соответствии с числом выходов решения, и каждая из таких требуемых взаимосвязей должна быть протестирована. К популярным критериям принадлежит критерий СР, заключающийся в покрытии всех таких пар дуг v и w, что из дуги v достижима дуга w, поскольку именно на дуге может произойти потеря значения переменной, которая в дальнейшем уже не должна использоваться. Для "покрытия" еще одного популярного критерия Cdu достаточно тестировать пары (вершина, дуга), поскольку определение переменной происходит в вершине УГП, а ее использование - на дугах, исходящих из решений, или в вычислительных вершинах.

Методы проектирования тестовых путей для достижения заданной степени тестированности в структурном тестировании. Процесс построения набора тестов при структурном тестировании принято делить на три фазы:

* Конструирование УГП.
* Выбор тестовых путей.
* Генерация тестов, соответствующих тестовым путям.

Первая фаза соответствует статическому анализу программы, задача которого состоит в получении графа программы и зависящего от него и от критерия тестирования множества элементов, которые необходимо покрыть тестами. На третьей фазе по известным путям тестирования осуществляется поиск подходящих тестов, реализующих прохождение этих путей. Вторая фаза обеспечивает выбор тестовых путей. Выделяют три подхода к построению тестовых путей:

* Статические методы.
* Динамические методы.
* Методы реализуемых путей.

Статические методы. Самое простое и легко реализуемое решение - построение каждого пути посредством постепенного его удлинения за счет добавления дуг, пока не будет достигнута выходная вершина управляющего графа программы. Эта идея может быть усилена в так называемых адаптивных методах, которые каждый раз добавляют только один тестовый путь (входной тест), используя предыдущие пути (тесты) как руководство для выбора последующих путей в соответствии с некоторой стратегией. Чаще всего адаптивные стратегии применяются по отношению к критерию С1. Основной недостаток статических методов заключается в том, что не учитывается возможная нереализуемость построенных путей тестирования.

Динамические методы. Такие методы предполагают построение полной системы тестов, удовлетворяющих заданному критерию, путем одновременного решения задачи построения покрывающего множества путей и тестовых данных. При этом можно автоматически учитывать реализуемость или нереализуемость ранее рассмотренных путей или их частей. Основной идеей динамических методов является подсоединение к начальным реализуемым отрезкам путей дальнейших их частей так, чтобы: 1) не терять при этом реализуемости вновь полученных путей; 2) покрыть требуемые элементы структуры программы.

Методы реализуемых путей. Данная методика заключается в выделении из множества путей подмножества всех реализуемых путей. После чего покрывающее множество путей строится из полученного подмножества реализуемых путей.

Достоинство статических методов состоит в сравнительно небольшом количестве необходимых ресурсов, как при использовании, так и при разработке. Однако их реализация может содержать непредсказуемый процент брака (нереализуемых путей). Кроме того, в этих системах переход от покрывающего множества путей к полной системе тестов пользователь должен осуществить вручную, а эта работа достаточно трудоемкая. Динамические методы требуют значительно больших ресурсов как при разработке, так и при эксплуатации, однако увеличение затрат происходит, в основном, за счет разработки и эксплуатации аппарата определения реализуемости пути (символический интерпретатор, решатель неравенств). Достоинство этих методов заключается в том, что их продукция имеет некоторый качественный уровень - реализуемость путей. Методы реализуемых путей дают самый лучший результат.

Компонентное тестирование — тип тестирования ПО, при котором тестирование выполняется для каждого отдельного компонента отдельно, без интеграции с другими компонентами. Его также называют модульным тестированием (Module testing), если рассматривать его с точки зрения архитектуры. Как правило, любое программное обеспечение в целом состоит из нескольких компонентов. Тестирование на уровне компонентов (Component Level testing) имеет дело с тестированием этих компонентов индивидуально. Это один из самых частых типов тестирования черного ящика, который проводится командой QA. Для каждого из этих компонентов будет определен сценарий тестирования, который затем будет приведен к Test case высокого уровня -> детальным Test case низкого уровня с предварительными условиями.

Исходя из глубины уровней тестирования, компонентное тестирование можно классифицировать как:

* Тестирование компонентов в малом (CTIS — Component testing In Small). Тестирование компонентов может проводиться с или без изоляции остальных компонентов в тестируемом программном обеспечении или приложении. Если это выполняется с изоляцией другого компонента, то это называется CTIS. Пример: веб-сайт, на котором есть 5 разных веб-страниц, тестирование каждой веб-страницы отдельно и с изоляцией других компонентов.
* Тестирование компонентов в целом (CTIL — Component testing In Large). Тестирование компонентов, выполненное без изоляции других компонентов в тестируемом программном обеспечении или приложении, называется CTIL. Давайте рассмотрим пример, чтобы понять это лучше. Предположим, что есть приложение, состоящее из трех компонентов, таких как Компонент A, Компонент B и Компонент C. Разработчик разработал компонент B и хочет его протестировать. Но для того, чтобы полностью протестировать компонент B, некоторые его функции зависят от компонента A, а некоторые — от компонента C. Функциональный поток: A -> B -> C, что означает, что существует зависимость от B как от A, так и от C, заглушка - вызываемая функция, а драйвер - вызывающая функция. Но компонент A и компонент C еще не разработаны. В этом случае, чтобы полностью протестировать компонент B, мы можем заменить компонент A и компонент C заглушкой и драйверами по мере необходимости. Таким образом, в основном, компоненты A & C заменяются заглушками и драйверами, которые действуют как фиктивные объекты до тех пор, пока они фактически не будут разработаны.

|  |  |
| --- | --- |
| Unit testing | Component testing |
| Тестирование отдельных программ, модулей, функций для демонстрации того, что программа выполняется согласно спецификации | Тестирование каждого объекта или частей программного обеспечения отдельно с или без изоляции других объектов |
| Проверка в(на) соответствии с design documents | Проверка в(на) соответствии с test requirements, use case |
| Пишутся и выполняются(обычно) разработчиками | Тестировщиками |
| Выполняется первым | Выполняется после Unit |

Разница между компонентным и модульным тестированием: По-существу эти уровни тестирования представляют одно и тоже, разница лишь в том, что в компонентном тестировании в качестве параметров функций используют реальные объекты и драйверы, а в модульном тестировании - конкретные значения.