Модульное тестирование

Модульное тестирование, или юнит-тестирование

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.
Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.
Этот тип тестирования обычно выполняется программистами.

Поощрение изменений

Модульное тестирование позже позволяет программистам проводить рефакторинг, будучи уверенными, что модуль по-прежнему работает корректно (регрессионное тестирование). Это поощряет программистов к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений.

Упрощение интеграции

Модульное тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируя отдельные части программы, а затем программу в целом.

Документирование кода

Модульные тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит-тест в качестве примера.

Отделение интерфейса от реализации

Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним. Например, класс пользуется базой данных; в ходе написания теста программист обнаруживает, что тесту приходится взаимодействовать с базой. Это ошибка, поскольку тест не должен выходить за границу класса. В результате разработчик абстрагируется от соединения с

базой данных и реализует этот интерфейс, используя свой собственный mock-объект. Это приводит к менее связанному коду, минимизируя зависимости в системе.

Сложный код

Тестирование программного обеспечения — комбинаторная задача. Например, каждое возможное значение булевской переменной потребует двух тестов: один на вариант TRUE, другой — на вариант FALSE. В результате на каждую строку исходного кода потребуется 3−5 строк тестового кода.
Как и любая технология тестирования, модульное тестирование не позволяет отловить все ошибки программы. В самом деле, это следует из практической невозможности трассировки всех возможных путей выполнения программы, за исключением простейших случаев.

Результат известен лишь приблизительно

Например, в математическом моделировании. Бизнес-приложения зачастую работают с конечными и счётными множествами, научные — с континуальными. Поэтому сложно подобрать тесты для каждой из ветвей программы, сложно сказать, верен ли результат, выдерживается ли точность, и т. д. А во многих случаях качество моделирования определяется «на глаз», и последний результат записывается как «опорный». Если найдено расхождение, новый результат проверяют вручную и выясняют, какой качественнее: старый или новый.

Ошибки интеграции и производительности

При выполнении юнит-тестов происходит тестирование каждого из модулей по отдельности. Это означает, что ошибки интеграции, системного уровня, функций, исполняемых в нескольких модулях, не будут определены. Кроме того, данная технология бесполезна для проведения тестов на производительность. Таким образом, модульное тестирование более эффективно при использовании в сочетании с другими методиками тестирования.

При общей низкой культуре программирования

Для получения выгоды от модульного тестирования требуется строго следовать технологии тестирования на всём протяжении процесса разработки программного обеспечения. Нужно хранить не только записи обо всех проведённых тестах, но и обо всех изменениях исходного кода во всех модулях. С этой целью следует использовать систему контроля версий ПО. Таким образом, если более поздняя версия ПО не проходит тест,

который был успешно пройден ранее, будет несложным сверить варианты исходного кода и устранить ошибку. Также необходимо убедиться в неизменном отслеживании и анализе неудачных тестов. Игнорирование этого требования приведёт к лавинообразному увеличению неудачных тестовых результатов.

Другими словами

Основная идея модульного тестирования заключается в том, чтобы написать тесты, в которых проверена наименьшая «единица» кода. Модульные тесты обычно написаны на том же языке программирования, что и исходный код приложения. Они создаются непосредственно для проверки этого кода. То есть модульные тесты — это код, который проверяет корректность другого кода. Слово «тест» в контексте я использую достаточно либерально, потому что модульные тесты в каком-то смысле тестами не являются. Они ничего не испытывают. Я имею в виду, что при запуске модульного теста вы обычно не обнаруживаете, что какой-то код не работает. Вы это обнаруживаете во время написания теста, поскольку вы будете менять код до тех пор, пока тест не станет зелёным. Да, код может измениться позже, и тогда ваш тест может потерпеть неудачу. Так что в этом смысле модульный тест является регрессионным тестом. Модульный тест не похож на обычный тест, где у вас есть несколько шагов, которые вы собираетесь выполнить, и вы видите, работает ли программное обеспечение правильно или нет. В процессе написания модульного теста вы обнаруживаете, делает ли код то, что он должен или нет, и будете менять код до тех пор, пока тест не будет пройден.