Юнит-тесты: что это такое, зачем они нужны и как их проводят

Что такое юниты

Чтобы разобраться с понятием юнит-тестирования, надо для начала узнать определение юнитов. Любая программа состоит из юнитов, или модулей, — отдельных блоков и функций. Кнопка добавления товара в корзину, формула калькулятора стоимости, скрипт для формирования карточки товара — всё это отдельные модули.

Что такое модульное тестирование

Модули взаимодействуют и обеспечивают работу программы или приложения. Чтобы проверить, правильно ли написан модуль, проводят юнит-тесты, или модульное тестирование, — проверку не всего приложения, а одного модуля. Пример юнит-теста — проверка функции подсчёта общей стоимости заказа.

Модульное тестирование проводят сразу после написания кода. Проверить работу кнопки в готовом приложении не получится, потому что на неё уже влияют другие модули.

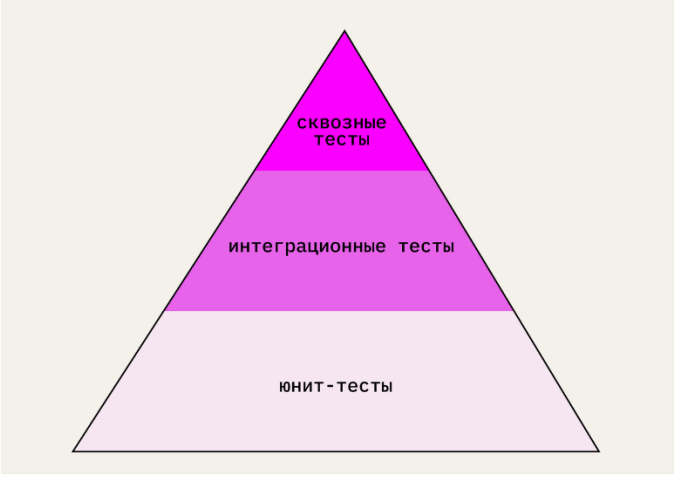
Зачем тестировать юниты

Главная причина написания юнит-тестов — тестирование отдельных модулей. Поскольку каждый модуль пишется отдельно, тестировать его тоже можно изолированно, без связки с другими. Получается простая схема: программист написал модуль → протестировал его → продолжил разработку для связи с другими модулями и других тестов.

Если пропустить этап юнит-тестирования, в следующий раз не получится понять, что именно вызвало ошибку: какой-то из модулей или неправильно настроенная интеграция между ними. Придётся разбираться, тратить время и всё равно тестировать отдельные юниты.

Для примера представим автомобиль. Его «юниты» — это двигатель, подача бензина, зажигание. Можно проверить их по отдельности и ещё до сборки увидеть поломки и починить их. А можно собрать автомобиль, не протестировав юниты, — и он не поедет. Придётся всё разбирать и проверять каждую деталь.

Особенности и преимущества юнит-тестов



Интеграционные тесты — тестирование взаимодействия нескольких юнитов. Например, кнопки «Купить товар» и корзины.

Сквозные (end-to-end) тесты — тестирование работы большого количества юнитов вместе. Это может быть как всё приложение, так и конкретный сценарий, например поиск товара, его помещение в корзину, заказ и оплата.

Если сравнивать с другими тестами, у модульных есть следующие особенности:

Можно провести сразу после написания кода. Программист пишет конкретный модуль и тут же его тестирует — не нужно ждать готовности других модулей или интеграций.

● Быстрее, чем другие тесты, так как охватывают только небольшую функцию. Часто один такой юнит-тест занимает всего пару миллисекунд.

● Не требуют серьёзной инфраструктуры, так как их выполнение не требовательно к вычислительным ресурсам.

● За счёт лёгкости и скорости юнит-тесты самые дешёвые.

● Разные юниты можно тестировать одновременно.

● Легко автоматизировать, так как при таких тестах нет имитации сценария пользователя — только проверка реакции кода на те или иные действия и данные.

● Просто посчитать, какой процент кода покрыт тестами.

Поэтому в пирамиде тестирования юнит-тесты стоят в самом низу — для экономии времени и сил их стоит проводить больше всего. В идеальном случае мы можем вообще обойтись только модульным тестированием проекта, то есть всего написанного кода — проверять только юниты, так как интеграция между ними предсказуемо работает правильно.

Процесс проведения и методы юнит-тестирования

В общих чертах модульное тестирование кода выглядит так:

1. Разработчик пишет код конкретной функции — юнита.
2. Проверяет, чтобы функция была изолирована, то есть не вшита намертво в другие функции. Если вшита — переписывает её, чтобы вынести.
3. Если функции нужны реакции от других модулей, разработчик создаёт моки — заглушки, которые имитируют другие модули и взаимодействие с ними. Например, передают данные, на которые тестируемый юнит должен отреагировать.
4. После этого разработчик пишет тесты и исправляет ошибки.
5. После запускает юнит-тест в режиме покрытия и смотрит, все ли строки функции покрыты, то есть протестированы.
6. В итоге через пару итераций получается хороший протестированный код.

Но есть и другой подход — Test driven development. Его суть в том, что разработчики получают задачу и сначала пишут тесты, основываясь на принципах модульного тестирования: на проверке отдельных юнитов сразу после написания кода. И уже под эти тесты пишут код, стараясь избежать предполагаемых ошибок.

Инструменты для проведения юнит-тестов. Например, если пишете на javascript, существует пакет jest. Он позволяет быстро делать заглушки, содержит инструменты для проверки покрытия, позволяет запускать тесты в разных режимах, в том числе в многопоточном.

Для других языков программирования тоже существуют свои инструменты для создания юнит-тестов — JUnit , Mockk, robolectric, пакет unittes для Python.

Главное о юнит-тестировании

1. Юнит, или модульные тесты, проверяют отдельные блоки и функции написанного кода.
2. Юнит-тесты нужны, чтобы быстро протестировать написанный фрагмент кода и сразу понять, где именно кроется ошибка.
3. Юнит-тесты дешевле и быстрее других, их легко автоматизировать.
4. Чтобы юнит-тест получился, тестируемый модуль должен быть изначально изолирован от другого кода. Если нужны какие-то зависимости, их имитируют заглушками — моками.

Возможность лучше разобраться в коде

Когда вы разбираетесь в плохо документированном сложном старом коде, попробуйте написать для него тесты. Это может быть непросто, но достаточно полезно, так как:

- они позволят вам убедиться, что вы правильно понимаете, как работает код;
- они будут служить документацией для тех, кто будет читать код после вас;
- если вы планируете рефакторинг, тесты помогут вам убедиться в корректности изменений.