Итак, как мы сказали, пирамида тестирования построена таким образом, что снизу вверх у нас возрастает сложность и дороговизна производства наших тестов. Поэтому мы сегодня на практике будем рассматривать такой вид тестирования, как модульные тесты. Они проще всего реализуются на практике и быстрее всего выполняются при проверке нашего кода.

Предположим, что нам нужно написать тест для нашей функции вычисления fibonacci. Это уже знакомая нам функция fib, которая внутри себя рекурсивно считает последовательность fibonacci.

Смотрите, в нашей функции есть три ветки. Первая ветка, это когда мы рассчитываем для значения 0. Вторая ветка – для единицы. И третья ветка там, где у нас происходит рекурсия и расчеты данных. Для того, чтобы нам написать тесты к этой функции, мы должны создать новый файлик, который называется fib_test. Если посмотреть на структуру проекта, здесь у меня будет файл fib, и рядом с ним будет лежать файл fib_test.

И на самом деле, при вызове тестов, все файлы с названием – нижнее «_test», Go будет воспринимать, как тестовые файлы. В этом файле у меня есть функция, которая представляет собой Test Case. В этом файле у меня есть функция, которая представляет собой Test Case, и эта функция называется – TestFibZero. Для того, чтобы нам объявить Test Case, нам обязательно нужно начать наше название функции со слова Test. И дальше уже описать как-то наш кейс. В эту функцию мы обязательно должны передать структуру Testing.T по ссылке, так как это требуется для запуска тестирования. Testing.T, это структура из пакета Testing, которая здесь импортируется.

И смотрите, внутри нашего Test Case-а, мы считаем результат для нуля. То есть вызываем нашу функцию fib для значения 0, и проверяем на корректность

результата. Если мы получили в ответе не 0, то мы фейлим наш Test. Запустить наш Test можно с помощью команды go test. Все запустилось, test прошел, мы получили, что все ок. И никаких проблем нет.

Для более детальной информации, можно запустить в расширенном формате, с параметром verbose -v. Здесь мы уже получаем что, здесь мы уже получаем название метода, который мы запускали, и время, сколько этот метод занял.

Помимо такой расширенной информации о тестах, мы еще можем посмотреть, какой процент кода у нас покрыт тестами. Для этого нужно запустить команду go test, с параметром -cover. И мы видим, что вывод говорит нам о том, что 40% нашего кода покрыто тестами.

Но go еще замечателен тем, что у него еще есть очень много встроенных инструментов, которые помогают нам в тех или иных ситуациях. В данном случае в go есть такой инструмент, как cover, который может показать, непосредственно, код, который покрыт тестами, и код, который еще не покрыт тестами.

Для этого давайте мы сгенерируем файл, в котором поместит себе информацию о нашем код coverage. Делается это с помощью параметра coverprofile, куда мы передаем название файлика, в который нужно поместить эти данные. Давайте назовем его cover.out. И мы видим, что здесь у нас появился файлик cover.out. Но читать его достаточно сложно. Поэтому мы можем воспользоваться инструментом go – go tool cover. И сказать ему, что нам нужно сгенерировать html из файла cover.out, и положить результат в файл cover.html. Мы видим, что у нас появился здесь html файл, давайте его откроем и посмотрим, что внутри.

В данном html файле мы наблюдаем? что у нас 40% покрытия кода. И у нас вот как раз выделен код, который действительно покрыт. Как мы видели, в Test Case-е мы описали ветку, где рассчитываем последовательность fibonacci для нуля.

Поэтому покрытие кода у нас только происходит для этой ветки, а все остальное то, что красным, у нас еще не покрыто.

Давайте эту ситуацию исправлять. И напишем Test Case-ы для двух других веток. Перейдем снова в наш файлик fib_test, и давайте я скопирую наш Test Case. Назову его one, где мы протестируем для единицы, и посмотрим, что у нас изменилось. Давайте снова сгенерируем файл cover, и создадим html. Перейдем обратно в наш браузер, и здесь мы видим, что ситуация изменилась. Теперь у нас 80% покрытия. И нам отображает, что этот код тоже покрылся тестами.

И давайте напишем кейс для последней ветки, где происходит расчет с рекурсией. Здесь уже у нас будет произвольное число, давайте, например, передадим 3, и проверим на равенство двум, потому что для третьего числа последовательность fibonacci будет равно двум. Давайте проверим. Да, тест прошел, все хорошо. Мы теперь можем опять сгенерировать файл coverage, и потом html. Посмотрим, что изменилось. Теперь у нас 100% покрытие кода. И весь код нашей функции отображается зеленым, что означает, что все кейсы мы покрыли.