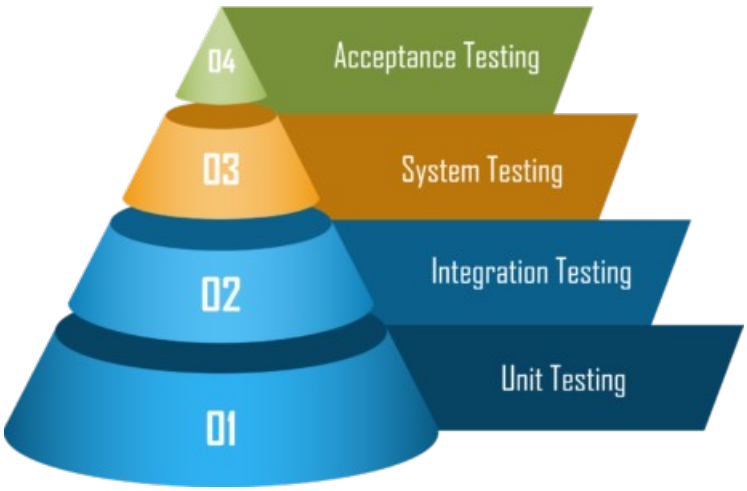
Уровни тестирования

Пирамида тестирования, также часто говорят уровни тестирования, это **группировка тестов по уровню детализации и их назначению**. Эту абстракцию придумал Майк Кон и описал в книге «Scrum: гибкая разработка ПО» (Succeeding With Agile. Software Development Using Scrum).

Пирамиду разбивают на **4 уровня**(снизу вверх), например, по ISTQB (wiki):

- модульное тестирование (юнит);
- интеграционное тестирование;
- системное тестирования;
- приемочное тестирование.



Но можно встретить варианты, где 3 уровня. В этой модели объединяют интеграционный и системный уровни:

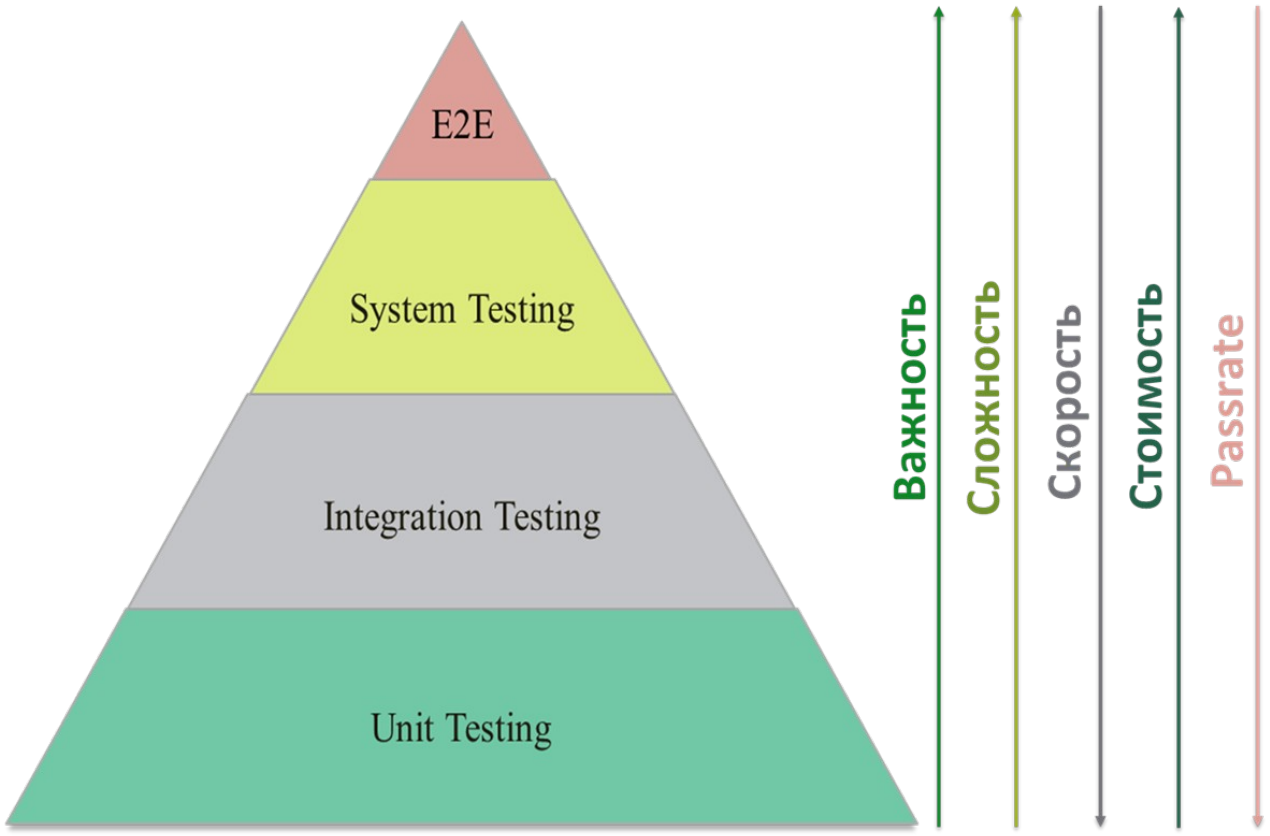
- модульное тестирование (юнит);
- интеграционное тестирование (включает в себя системное);
- приемочное тестирование.

1.Тест (ручной, на высоких уровнях, или автотест, на низких уровнях), должен быть на том же уровне, что и тестируемый объект. Например, модульный тест (проверяющий функции, классы, объекты и т. п.) должен быть на компонентном уровне. Это неправильно, если на приемочном уровне запускается тест, который проверят минимальную единицу кода.

2.Тесты уровнем выше не проверяют логику тестов уровнем/уровнями ниже.

3.Чем выше тесты уровнем, тем они:

- сложней в реализации, и соответственно, дороже в реализации;
- важнее для бизнеса и критичней для пользователей;
- замедляют скорость прохождения тестовых наборов, например, регресса.



Влияние уровней на продукт, процессы и пользователей.

Компонентный уровень

Чаще всего называют юнит тестированием. Реже называют модульным тестированием. На этом уровне тестируют атомарные части кода. Это могут быть классы, функции или методы классов.

Пример: твоя компания разрабатывает приложение "Калькулятор", которое умеет складывать и вычитать. Каждая операция это одна функция. Проверка каждой функции, которая не зависит от других, является юнит тестированием.

Юнит тесты находят ошибки на фундаментальных уровнях, их легче разрабатывать и поддерживать. Важное преимущество модульных тестов в том, что они быстрые и при изменении кода позволяют быстро провести регресс (убедиться, что новый код не сломал старые части кода).

Тест на компонентном уровне:

1. Всегда автоматизируют.
2. Модульных тестов всегда больше, чем тестов с других уровней.

3. Юнит тесты выполняются быстрее всех и требуют меньше ресурсов.
4. Практически всегда компонентные тесты не зависят от других модулей (на то они и юнит тесты) и UI системы.

В 99% разработкой модульных тестов занимается разработчик, при нахождении ошибки на этом уровне не создается баг-репортов. Разработчик находит баг, правит, запускает и проверяет (абстрактно говоря это разработка через тестирование) и так по новой, пока тест не будет пройден успешно.

На модульном уровне разработчик (или автотестер) использует метод белого ящика. Он знает что принимает и отдает минимальная единица кода, и как она работает.

Интеграционный уровень

Проверяет взаимосвязь компоненты, которую проверяли на модульном уровне, с другой или другими компонентами, а также интеграцию компоненты с системой (проверка работы с ОС, сервисами и службами, базами данных, железом и т.д.). Часто в английских статьях называют service test или API test.

В случае с интеграционными тестами редко когда требуется наличие UI, чтобы его проверить. Компоненты ПО или системы взаимодействуют с тестируемым модулем с помощью интерфейсов. Тут начинается участие тестирования. Это проверки API, работы сервисов (проверка логов на сервере, записи в БД) и т.п.

Строго говоря на модульном уровне тестирование тоже участвует. Возможно помогает проектировать тесты или во время регресса смотрит за прогоном этих тестов, и если что-то падает, то принимает меры.

Отдельно отмечу, что в интеграционном тестировании, выполняются как функциональные (проверка по ТЗ), так и нефункциональные проверки (нагрузка на связку компонент). На этом уровне используется либо серый, либо черный ящик.

В интеграционном тестировании есть 3 основных способа тестирования (представь, что каждый модуль может состоять еще из более мелких частей):

- Снизу вверх (Bottom Up Integration): все мелкие части модуля собираются в один модуль и тестируются. Далее собираются следующие мелкие модули в один большой и тестируется с предыдущим и т. д. Например, функция публикации фото в соц. профиле состоит из 2 модулей: загрузчик и публикатор. Загрузчик, в свою очередь, состоит из модуля компрессии и отправки на сервер. Публикатор состоит из верификатора (проверяет подлинность) и управления доступом к фотографии. В интеграционном тестировании соберем модули загрузчика и проверим, потом соберем модули публикатора, проверим и протестируем взаимодействие загрузчика и публикатор.
- Сверху вниз (Top Down Integration): сначала проверяем работу крупных модулей, спускаясь ниже добавляем модули уровнем ниже. На этапе проверки уровней выше данные, необходимые от уровней ниже, симулируются. Например, проверяем

работу загрузчика и публикатора. Руками (создаем функцию-заглушку) передаем от загрузчика публикатору фото, которое якобы было обработано компрессором.

●Большой взрыв ("Big Bang" Integration): собираем все реализованные модули всех уровней, интегрируем в систему и тестируем. Если что-то не работает или недоработали, то фиксим или дорабатываем.

Системный уровень

О системном уровне говорили в интеграционном. Тут отметить только то, что:

1.Системный уровень проверят взаимодействие тестируемого ПО с системой по функциональным и нефункциональным требованиям.

2.Важно тестировать на максимально приближенном окружении, которое будет у конечного пользователя.

Тест-кейсы на этом уровне подготавливаются:

1.По требованиям.

2.По возможным способам использования ПО.

На системном уровне выявляются такие дефекты, как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д.

На этом уровне используют черный ящик. Интеграционный уровень позволяет верифицировать требования (проверить соответствие ПО прописанным требованиям).

Приемочное тестирование

Также часто называют E2E тестами (End-2-End) или сквозными. На этом уровне происходит валидация требований (проверка работы ПО в целом, не только по прописанным требованиям, что проверили на системном уровне).

Проверка требований производится на наборе приемочных тестов. Они разрабатываются на основе требований и возможных способах использования ПО.

Отмечу, что приемочные тесты проводят, когда (1) продукт достиг необходимо уровня качества и (2) заказчик ПО ознакомлен с планом приемки (в нем описан набор сценариев и тестов, дата проведения и т.п.).

Приемку проводит либо внутреннее тестирование (необязательно тестировщики) или внешнее тестирование (сам заказчик и необязательно тестировщик).

Важно помнить, что E2E тесты автоматизируются сложнее, дольше, стоят дороже, сложнее поддерживаются и трудно выполняются при регрессе. Значит таких тестов должно быть меньше.