Test_Pyramid

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 :

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

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.

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.

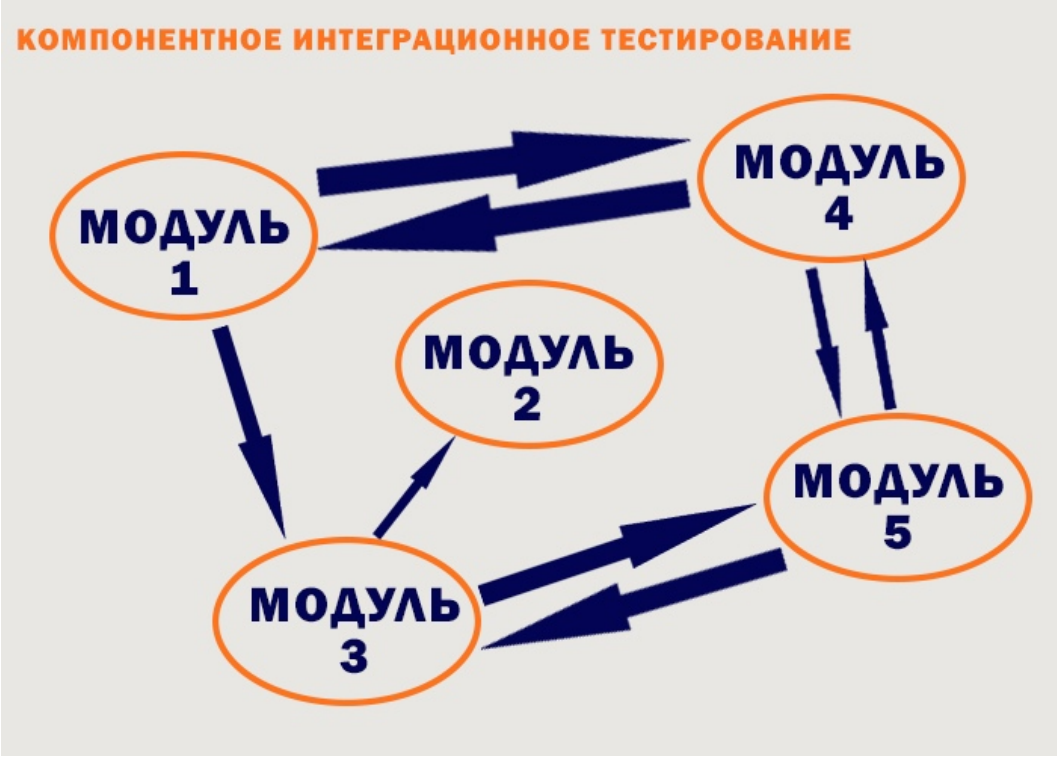
В 99% разработкой модульных тестов занимается разработчик, при нахождении ошибки на этом уровне не создается баг-репортов. Разработчик находит баг, правит, запускает и проверяет *(абстрактно говоря это разработка через тестирование)* и так по новой, пока тест не будет пройден успешно.

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

С помощью компонентного тестирования мы **снижаем риски** и укрепляем свою уверенность в качестве продукта.

Среда тестирования: Среда разработки (Development Env).

Интеграционный уровень



Проверят 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 отметим, 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.

С помощью интеграционного тестирования мы снижаем риски и укрепляем свою уверенность в качестве продукта.

Среда тестирования: Среда тестирования (Test Env), Интеграционная среда (Integration Env)

Системный уровень

При системном тестировании наша задача уже состоит в том, чтобы убедиться в корректности работы в целом всей системы. Программа в этом случае должна быть максимально приближена к конечному результату. А наше внимание должно быть сосредоточено на общем поведении системы с точки зрения конечных пользователей.

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**.

С помощью системного тестирования мы снижаем риски и укрепляем свою уверенность в качестве продукта.

Среда тестирования: Среда тестирования (Test Env), Интеграционная среда (Integration Env)

Приемочное тестирование

Приемочное тестирование — наиболее высокий уровень тестирования. Оно, также как и системное тестирование, необходимо для проверки работы программы в целом.

Тут также смещаются цели тестирования. Ошибок на этом этапе уже не должно быть. Скорее наоборот, программа должна быть **максимально рабочей и пригодной для использования**. Если на данном этапе обнаруживается критичные дефекты, то есть большая вероятность того, программа была плохо протестирована на предыдущих уровнях.

Этот уровень тестирования используется для подтверждения готовности продукта и проводится преимущественно в самом конце цикла разработки программы.

У приемочного тестирования есть также несколько целей:

1. Показать, что программа завершена и готова к использованию так, как от нее ожидалось.

2. Проверить, что работа программы соответствует установленному ТЗ или требованиям.

Также, на этом уровне тестирования мы показываем уверенность в качестве системы.

По версии ISTQB существует несколько **форм приемочного тестирования**:

1. Пользовательское приемочное тестирование.

2. Эксплуатационное приемочное тестирование.

3. Контрактное и нормативное приемочное тестирование.

4. Альфа- и Бета-тестирование.

Пользовательское приемочное тестирование предназначено для проверки программы, как если бы ее использовал конечный пользователь. В этом случае мы должны убедиться, что все функции и части работают так, как задумывалось в требованиях. Если вернуться к примеру с программой по поиску такси, то мы должны быть уверены, что такси вызывается корректно, можно оплачивать поездку через программу, оставлять отзывы, отменять вызов и так далее.

С другой стороны стоит ***эксплуатационное*** приемочное тестирование. Его отличие заключается в том, что мы проводим тестирование не с позиции пользователей, а с позиции тех, кто будет поддерживать работу программы. Наша задача — убедиться в работоспособности таких аспектов, как:

1. Возможность резервного копирования и восстановления данных.

2. Установка, удаление и обновление программы.

3. Восстановление после полного падения системы.

4. Управление пользователями.

5. Возможность сопровождения (обслуживания).

6. Возможность загрузки и миграции данных.

7. Отсутствия уязвимостей.

8. Хорошая производительность.

Если программа разрабатывается у сторонней компании, то иногда заключается контракт, в котором оговорены условия приемки. Проверка на

соответствие таким критериям проводится при ***контрактном*** приемочном тестировании.

Альфа- и Бета- *тестирование* используется, когда есть необходимость в получении обратной связи от пользователей. Поэтому именно они участвуют в таких проверках. Отличие альфа-тестирования от бета-тестирования заключается в том, что альфа-тестирование проводится внутри компании на потенциальных пользователях, а бета-тестирование проводится в ограниченном кругу конечных пользователей программы. Такое часто распространено в играх. Игрокам сначала показывается бета версия игры, а через некоторое время игра выходит в релиз и становится доступной для всех.

Среда тестирования: Среда тестирования (Test Env), Интеграционная среда (Integration Env), Превью среда (Preview, Preprod Env).