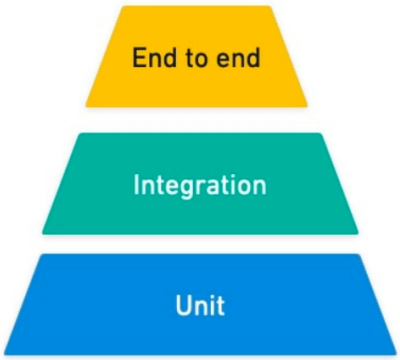
# Test_Pyramid

**«Пирамида тестов»** — метафора, которая означает группировку тестов программного обеспечения по разным уровням детализации. Она также даёт представление, сколько тестов должно быть в каждой из этих групп.




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

## Юнит-тесты (модульные): находят ошибки на фундаментальных уровнях, их легче разрабатывать и поддерживать.

*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.*

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.

**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).*

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.

Среда тестирования - различные слои тестируемого приложения,

## Интеграционные тесты (service test или API test): Проверяют 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.)*

П*ример интегрaционных тестов можно рассмотреть соединение с базой данных и проверку правильной отработки методов, работающих с ней*

В интеграционном тестировании, **выполняются как функциональные** *(проверка по ТЗ)*, так и **нефункциональные проверки** *(нагрузка на связку компонент)*. На этом ровне **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.

Среда тестирования - Тестируемое приложение, ОС, сервисы и службы, базы данных, железо и т.д.

## Системный уровень (входит в интеграционные тесты)

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

1. 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.

## End-to-End тесты (Приемочное тестирование; сквозное тестирование)

**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.**

Приемочные тесты проводят, когда:

1. **продукт достиг необходимо уровня качества**
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.**

Среда тестирования - пользовательский интерфейс тестируемого приложения

**Плюсы автотестов:**

**1. Неутомимость:** автотесты работают даже когда вы спите. Роботы запускаются автоматически, дистанционно, по расписанию. Они не отвлекаются, не забывают, и делают проверку столько раз, сколько нужно.

**2. Скорость:** робот в 99% случаев пройдет тест быстрее ручного тестировщика. В оставшемся 1% случаев не забывайте, что человек может устать, а робот нет.

**3. Многофункциональность:** это не просто перебор значений в формах. Автотесты могут быстро проверить функционал в разном окружении и при разных настройках тестируемого ПО.

**4. Масштаб:** автоматизация позволяет имитировать действия большого количества пользователей.

**5. Экономия сил:** автотесты освобождают ручных тестировщиков от рутины. Часто с помощью автотестов проверяется базовый функционал, а тестировщик сосредотачивается на тестировании новинок.

**6. Экономия средств:** основная задача автотестов в бизнесе — сокращение затрат на тестирование. И они отлично справляются с этой задачей, если были внедрены с умом и в нужном месте.

**Минусы автотестов (Риски):**

**1. Поломки:** автотесты ломаются, иногда даже из-за незначительного изменения кода. На актуализацию нужно время.

**2. Близорукость:** Автотест проверяет только то, на что запрограммирован. Он не заметит ошибку, которую ему не поручали искать.

**3. Трудно поддерживать:** с ростом количества автотестов, время на их актуализацию и анализ старых, превышает время на разработку новых.

**4. Не везде применимы:** есть области тестирования, которые не поддаются автоматизации. Это юзабилити, проверка верстки и переводов, инсталляционное тестирование и другие подобные сферы.

**5. Затратность:** автотест это как промышленное оборудование, в него нужно сначала инвестировать, а потом смотреть на окупаемость. А если “станок” постоянно чинится и перепрошивается — он может и не окупиться.