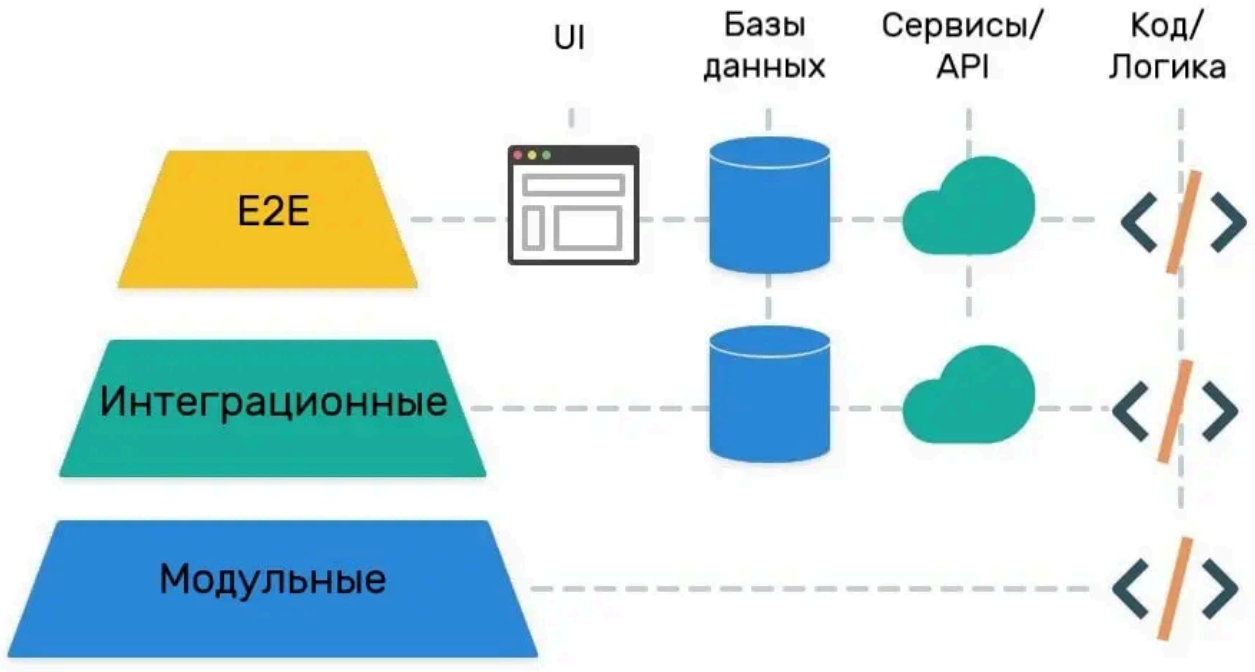
Summary 1. Пирамида тестирования, базовое понимание и ценность

Очень кратко

Пирамида тестирования.



Пирамида тестирования — это модель, которая помогает правильно расставлять приоритеты при написании автотестов в проекте.
Зачем? - Помогает классифицировать тесты в зависимости от их стоимости, уровню интеграции между компонентами, скорости тестирования. Такая классификация позволяет сбалансировать качество и стоимость тестов.

1. Нижний уровень. Unit тесты

- Тестируем отдельные методы или функции
- Самые быстрые, легкие в поддержке, сохраняют логику
- Примитивные, большое количество

2. Средний уровень. Integration тесты

- Тесты взаимодействия между компонентами.
- Их меньше, чем юнитов, в виду более низкой скорости.
- Тесты второго уровня пирамиды тестирования больше похожи на реальные действия пользователей, чем юниты, поэтому позволяют повысить надёжность на более раннем этапе

- Трудность настройки - если интеграционный тест провалился, бывает сложно определить, в какой именно части системы возникла проблема, поддержка также усложнена
- Примеры - api-тесты, проверка взаимодействия фронтенда и бэкенда

3. Верхний уровень. Функциональные, UI, e2e тесты

- Тесты максимально приближены к пользовательскому взаимодействию, имитируют действия пользователей, что позволяет учесть бОльшее количество рисков, обнаружить бОльшее количество дефектов.
- Тестов верхнего уровня еще меньше, так как они долго исполняются, сложнее в написании, тяжелее в поддерживании

Подробнее

Чем ценна пирамида тестирования?

Пирамида тестирования — это классическая концепция проведения тестов продукта перед запуском. С её помощью можно обеспечить баланс между различными видами тестов и оптимизировать процесс тестирования. Пирамида помогает определить приоритеты проверок, распределить ресурсы и усилия на разных уровнях и обеспечить высокое качество программного продукта.
Конкретно, она помогает:

- повысить стабильность тестов
- уменьшить стоимость их поддержки
- не создавать избыточных тестов

Идея разделения на уровни

Идея пирамиды в том, что чем выше тест в пирамиде — тем он:

- **медленнее**
- **дороже**
- **сложнее в поддержке**
- **работает с большей частью системы**

И наоборот: чем ниже — тем тесты:

- **быстрее**
- **дешевле**
- **ближе к коду**
- **тестируют меньшие куски**

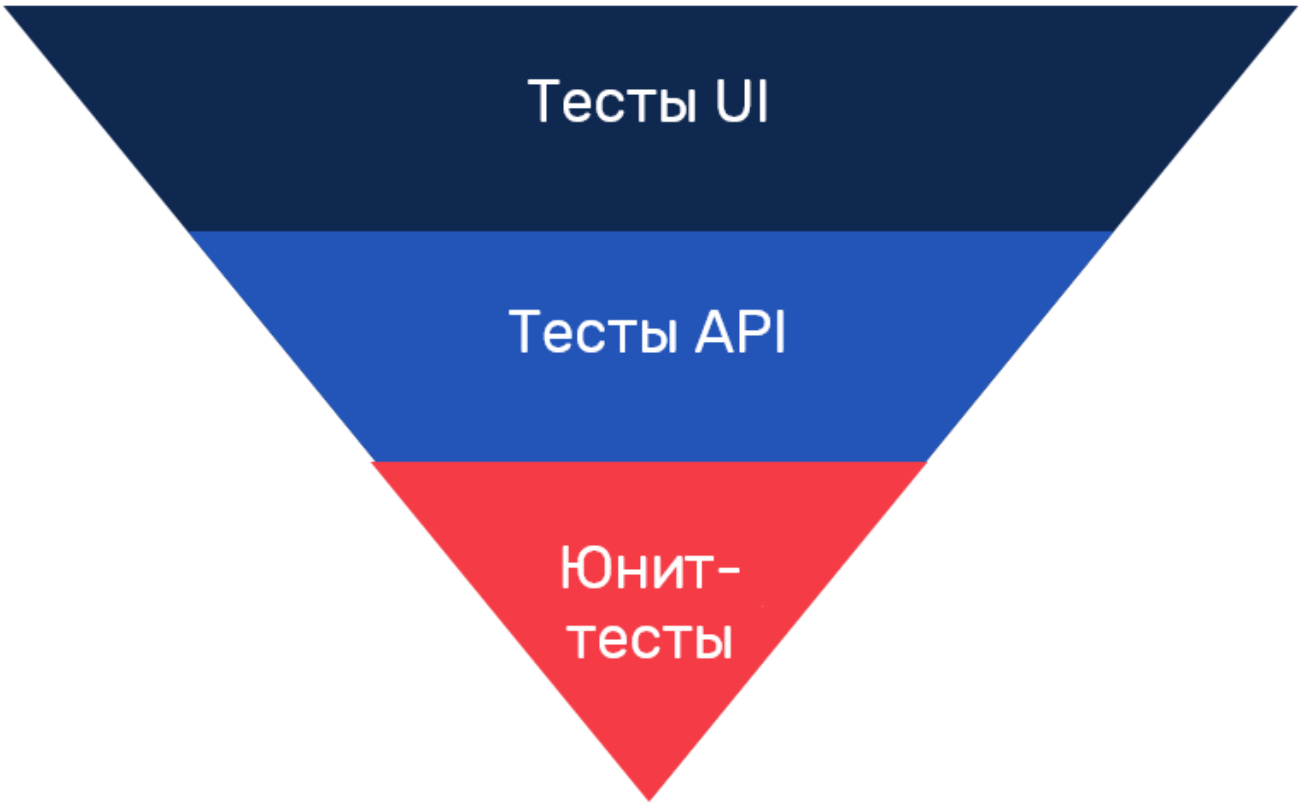
Классическая структура

Нижний уровень.
Тестируют отдельную функцию или класс изолированно от остальной системы.
Почему на нижнем уровне:

- Самые быстрые
- Простые в отладке
- Легко запускаются часто
- Дают точно и быстрое понимание о поломке в коде, а также о том, что именно сломано
  Средний уровень.
  Проверяют, как несколько частей системы работают вместе.
- Медленнее юнитов (из-за взаимодействия компонентов)
- Могут использовать реальные или мок-объекты (например, мок-сервисы, тестовую базу)
- Уже тестируют «связь» между модулями
  Верхний уровень.
  Проверяют всю систему целиком, как её использует реальный пользователь.
- Самые медленные (задействуют сеть, БД, фронтенд, бекенд)
- Дорогие в поддержке (инфраструктура, тестовые пользователи)
- Подвержены ложным срабатываниям (нестабильность окружения)
- Их должно быть мало, только для критических пользовательских сценариев
  Важно понимать, что границы между уровнями - не строгие, а логические.
  Это приводит к тому, что появляется расширенная пирамида тестирования.

Иные структуры

Обратная пирамида



Суть - Это ситуация, когда:

- Много E2E тестов
- Мало интеграционных
- Ещё меньше юнитов
  Проблемы:
- Медленно (каждый E2E-прогон долго идёт)
- Ненадёжно (большая вероятность ложных срабатываний из-за инфраструктуры, сетевых лагов, тестовых данных)
- Дорого в поддержке (меняется интерфейс, приходится переписывать тесты)
- Поздняя обратная связь (баг обнаружится после того, как всё загрузится и начнёт взаимодействовать)
  Когда оправдано?
- Проект очень маленький
- Демо/прототип
- Однофункциональное приложение, где проще прогнать пару E2E, чем настраивать сложную систему юнитов
- Периодически для smoke-сценариев или acceptance тестов

Вулкан

Еще один вариант тестовой пирамиды — «извергающийся вулкан». Это, по ощущениям, часто встречающийся вариант уровней тестирования в современной разработке (2023). Пирамида, точнее «гора» начинается со стандартных и всегда необходимых юнит- и интеграционных тестов; с добавлением, после юнит-, также и компонентных тестов; далее, после тестов интеграции — тестов API, которые уже являются необходимыми в 99% современных проектов; и тестов интерфейса и юзабилити, которые сейчас тоже незаменимы.
Над горой парит «облако», то ли дым извержения, это у нас исследовательские тесты, и ручные тесты, проводимые опытными тестировщиками в крупных QA-командах. Количество таких тестов может быть достаточно большим, а вообще неопределенно и непредсказуемо, как и длительность ручного тестирования.
Пирамида Вулкан рождается там, где:

- Слишком много логики на стыках сервисов
- Сами сервисы маленькие (например микросервисы), и в каждом из них юнит-тесты почти бесполезны, потому что:
  - Логика сильно завязана на внешние вызовы

- Отдельная функция без контекста мало что проверяет

Поэтому максимум тестирования приходится на интеграционный слой и API контракты между модулями.

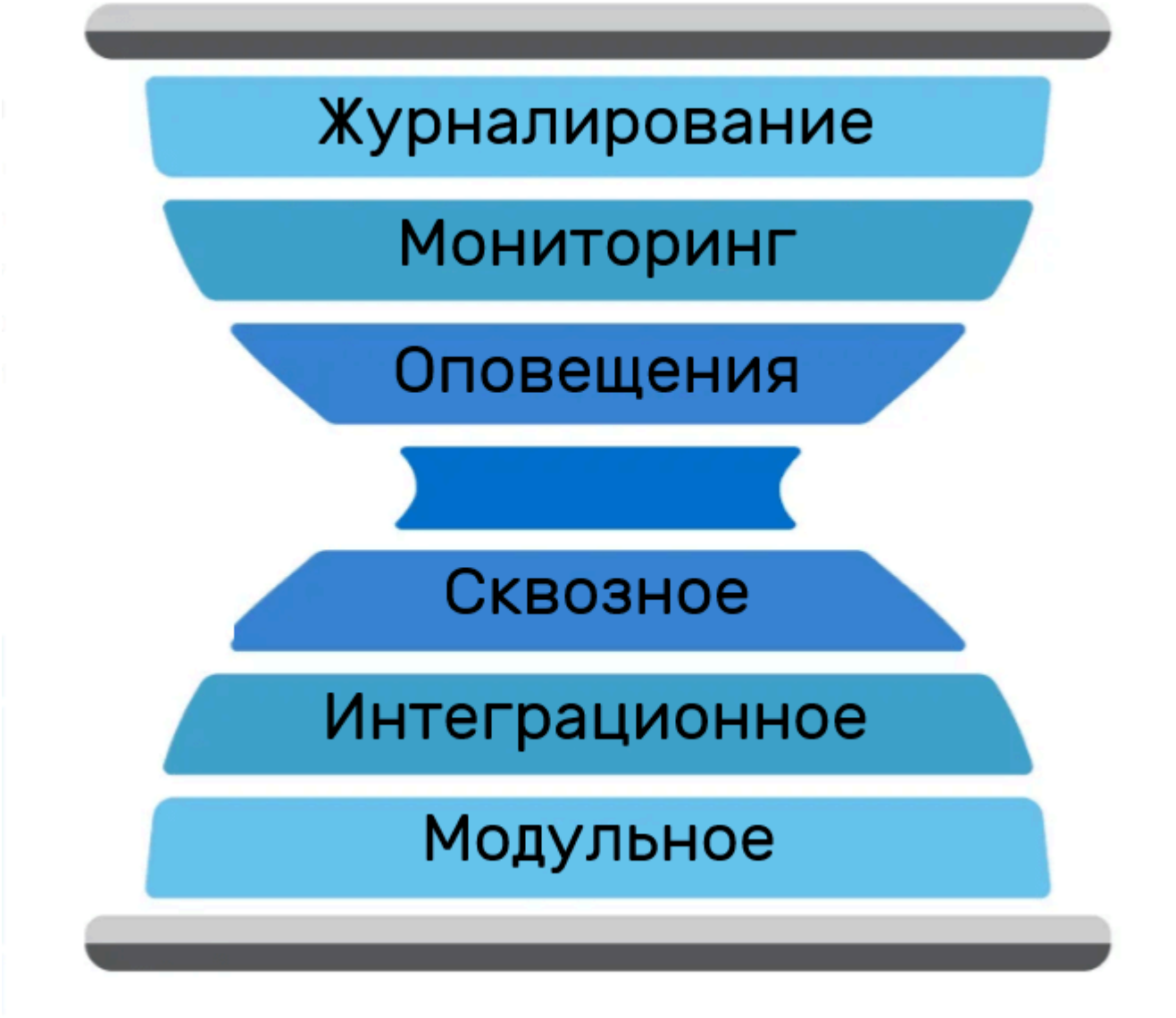
Плюсы

- Отлично подходит для микросервисов
- Позволяет быстро ловить ошибки на стыках
- Упрощает отладку интеграций
- Гибче, чем классическая пирамида
- Снижает зависимость от долгих и дорогих E2E

Минусы

- Требует продуманной системы тестовых окружений
- Интеграционные тесты сложнее в написании и поддержке
- Если нет хороших API-контрактов — тесты быстро начинают "плыть"
- Сложнее отлаживать при сильной связанности сервисов

Песочные часы



К стандартной тестовой пирамиде «присоединяются» последующие процессы Logging-Monitoring-Alerting.

- Журналирование (Logging) — налаженное протоколирование (фиксация) информации о происходящем в приложении
- Мониторинг (Monitoring) — автоматизированное наблюдение за состоянием приложения и его серверной части
- Оповещения (Alerting) — налаженный процесс автоматических оповещений о проблемных точках приложения, чтобы их по возможности немедленно устранить