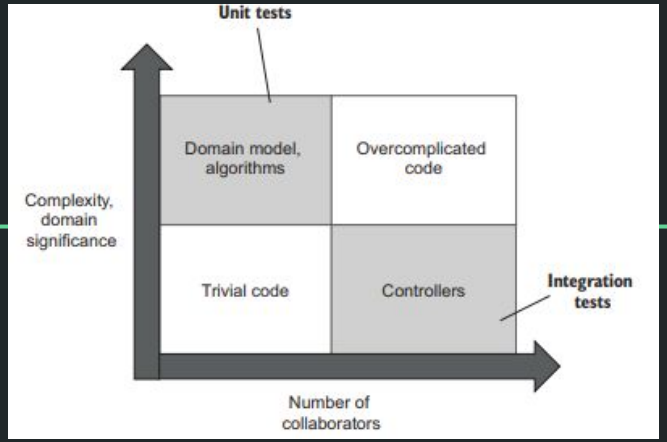
Интеграционное тестирование как понятие

Как всегда есть путаница в терминологии, когда переводим на русский:

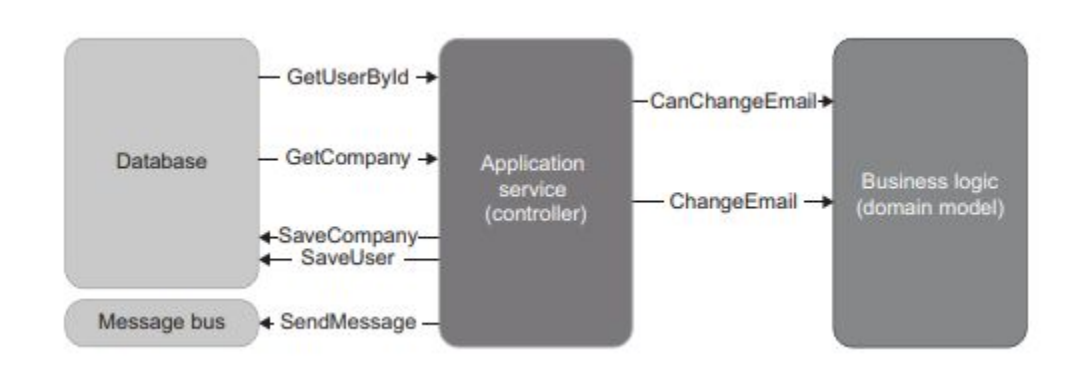
1. Во-первых: это проверка того, как разные компоненты взаимодействуют друг с другом без использования mock\stub. Не стоит это путать с “классическим” подходом к написанию unit-тестов, когда инициализируются разные классы\модули\компоненты чтобы протестировать другой класс. Здесь целью является проверить правильную работу всех компонентов, а не конкретного одного.



2. Во-вторых: часто под этим понимают интеграцию разных систем с помощью какого-либо протокола. На самом деле это правильней называть System Integration Testing (SIT), то есть тестирование интеграции нескольких систем.

Пример, концепция

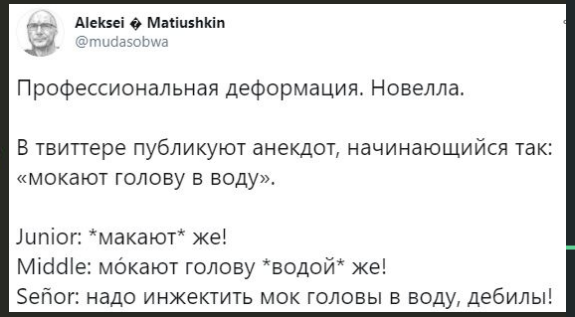
Сценарий использования -- изменение имэйла пользователя
Сценарий тестирования -- необходимо выбрать самую длинную цепочку событий без каких-либо исключений или ошибок



Требуется определить предусловия для выполнения теста для внешних зависимостей -- передача данных по шине, работа с БД
Не надо создавать абстрактные интерфейсы для БД и шины только упрощения создания тестов
Но если решили, то не надо создавать абстрактные интерфейсы для зависимости, пока вам не нужно делать для неё mock

Пример, реализация

```
1  public void Changing_email_from_corporate_to_non_corporate()
2  {
3      // Arrange
4      var db = new Database(ConnectionString);
5      User user = CreateUser("user@mycorp.com", UserType.Employee, db);
6      CreateCompany("mycorp.com", 1, db);
7      var messageBusMock = new Mock<IMessageBus>();
8      var sut = new UserController(db, messageBusMock.Object);
9
10     // Act
11     string result = sut.ChangeEmail(user.UserId, "new@gmail.com");
12
13     // Assert
14     Assert.Equal("OK", result);
15     object[] userData = db.GetUserById(user.UserId);
16     User userFromDb = UserFactory.Create(userData);
17     Assert.Equal("new@gmail.com", userFromDb.Email);
18     Assert.Equal(UserType.Customer, userFromDb.Type);
19
20     object[] companyData = db.GetCompany();
21     Company companyFromDb = CompanyFactory.Create(companyData);
22     Assert.Equal(0, companyFromDb.NumberOfEmployees);
23     messageBusMock.Verify(x => x.SendEmailChangedMessage(user.UserId, "new@gmail.com"),Times.Once);
24 }
```



Рекомендации по ведению интеграционных тестов

- Формальное обозначение границ модели предметной области

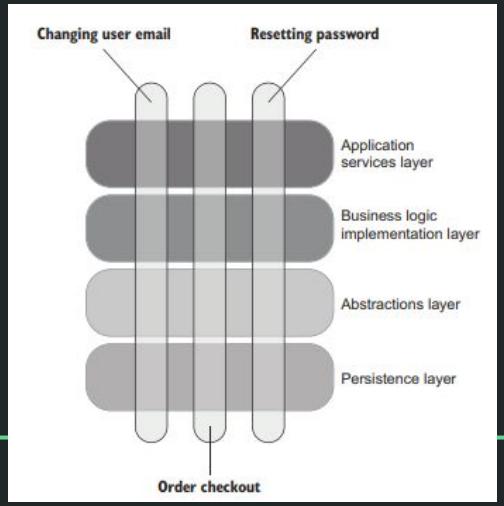
- Избавляться от циклических зависимостей между компонентами, интеграцией которых вы занимаетесь

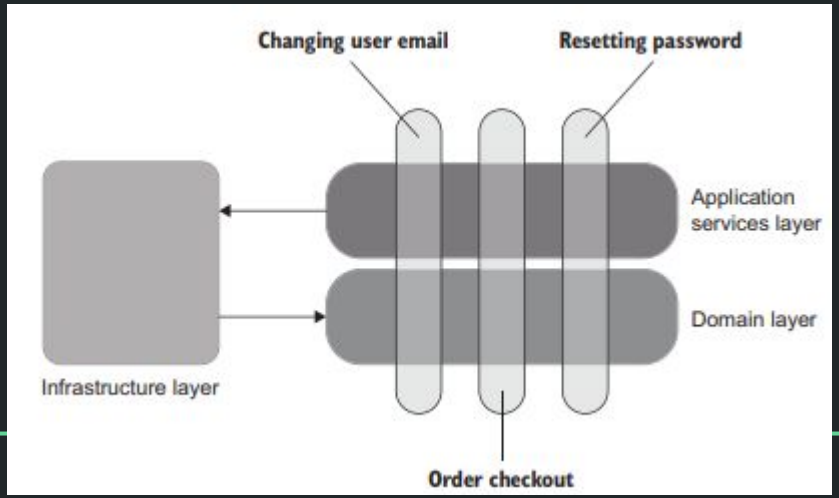
- Не надо стремиться покрыть все слои интеграции одним тестом

- Несколько act-секций допустимо, но лучше оставить это для E2E тестов

Ограничение количества слоёв в коде теста

Структура современных корпоративных приложений стремится к увеличению числа возможных уровней абстракции. Но это не очень удобно для тестов, т.к. задача теста -- проверить конкретный результат чтобы избежать конкретных регрессий.

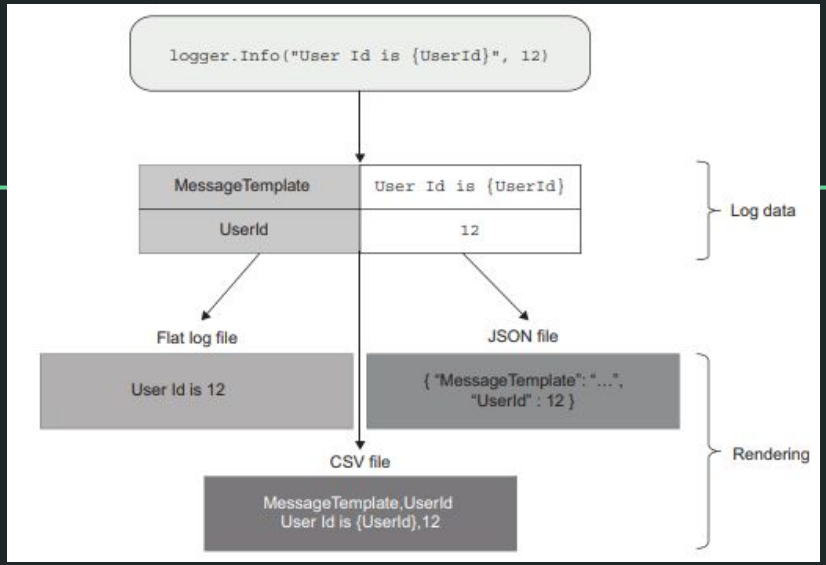




Все проблемы в информатике и разработке ПО могут быть решены с помощью добавления очередного уровня абстракции, кроме проблемы чрезмерно большого количества уровней абстракции (c) David Wheeler

Логирование / logging

- Логировать в целях упрощения поддержки ПО
- Логировать в целях понимания разработчиками как себя ведёт ПО
- Нужно выделять логгеры под области бизнес-логики со специфическим форматом полей
- Структура сообщений должна быть легко адаптирована под разные форматы вывода\хранения
- Логировать надо с меткой времени и указанием таймзоны



Системное тестирование \ System Testing

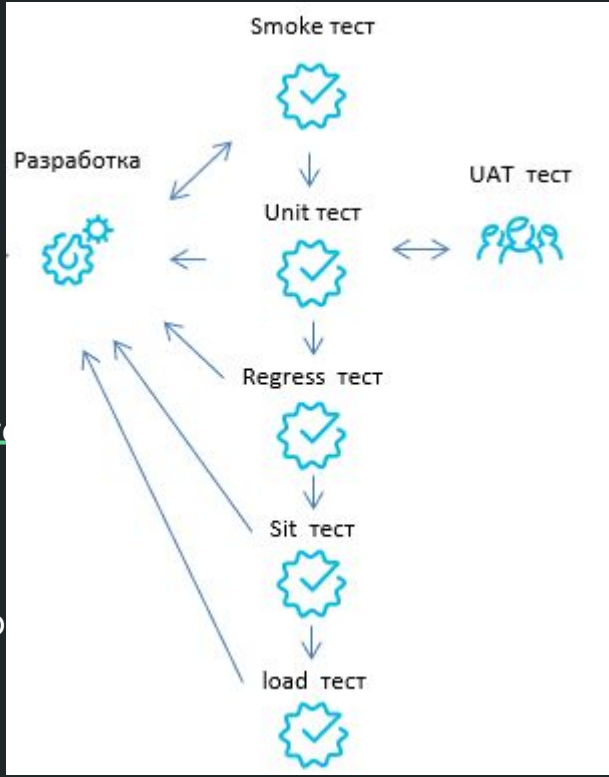
Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований в системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т. д. Для минимизации рисков, связанных с особенностями поведения системы в той или иной среде, во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи.

Можно выделить два подхода к системному тестированию:

- на базе требований (requirements based); Для каждого требования пишутся тестовые случаи (test cases), проверяющие выполнение данного требования.
- на базе случаев использования (use case based); На основе представления о способах использования продукта создаются случаи использования системы (Use Cases). По конкретному случаю использования можно определить один или более сценариев. На проверку каждого сценария пишутся тест кейсы (test cases), которые должны быть протестирован

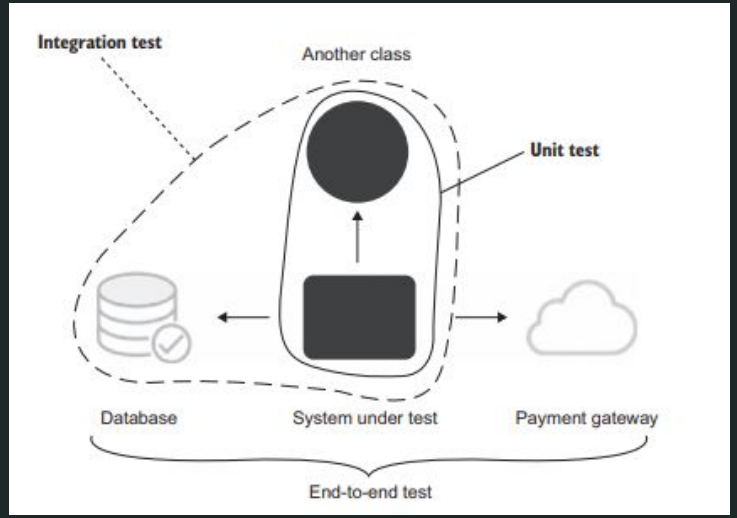
Системное Интеграционное тестирование \ SIT

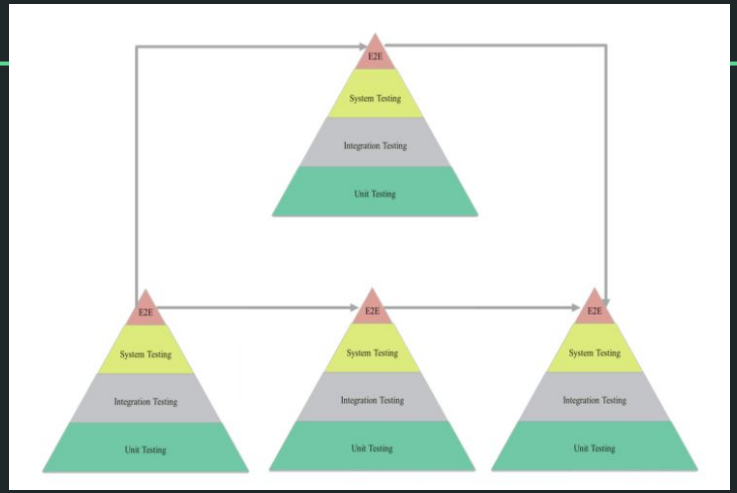
- Тестирование взаимодействия между различными системами после завершения этапа системного тестирования каждого участника интеграции
- Нужно для проверки взаимодействия в рамках сложных сценариев, где выделяются:
  - upstream (утрировано от клиента к серверу)
  - downstream (утрировано от сервера к клиенту)
  - утрировано потому что клиентом может являться что уг
- Сильно связано с User Acceptance Testing (UAT) -- нельзя начать UAT без проведения SIT
- Теоретически SIT тесты может написать разработчик так же как обычные интеграционные, но на практике это трудоемко
- Каждый вновь добавленный unit тест становится частью набора регрессионных тестов



End-to-end aka E2E тестирование

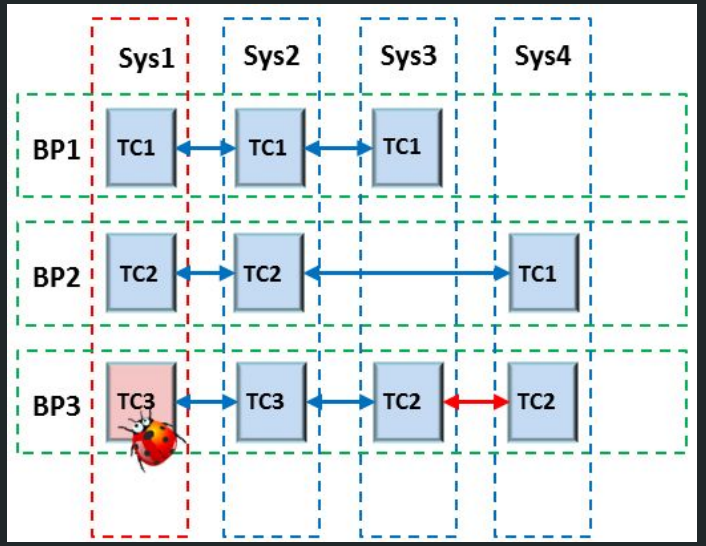
- E2E -- подмножество возможных интеграционных тестов
- При организации прогонов тестов следует размещать в порядке Unit ➜ Integration ➜ E2E
- Часто BDD\TDD подход подразумевает создание именно E2E тестов: понятных конечным пользователям
- Под E2E часто понимают UAT тесты
- Распространенная проблема: переход к E2E\UAT без фазы SIT (особенно в Agile проектах)

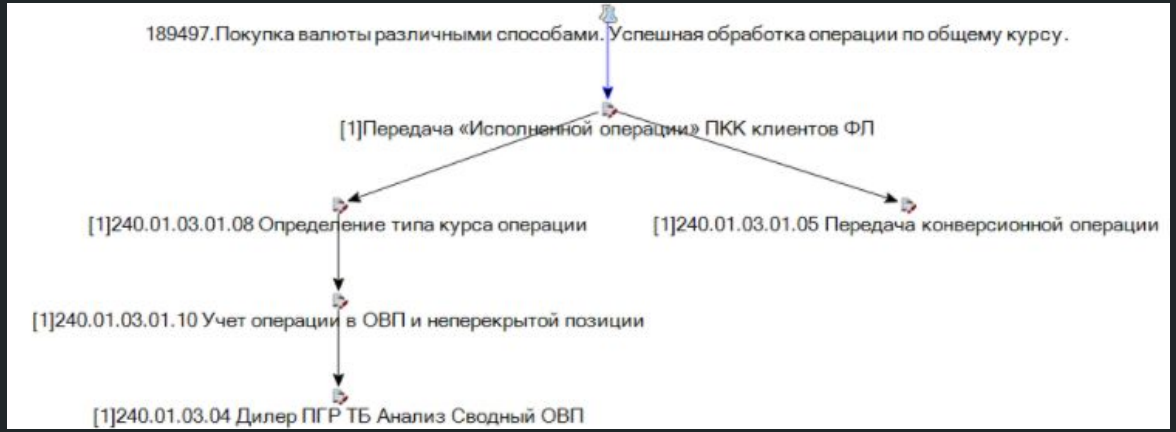




E2E -- тестирование процесса

- критически важные процессы заказчика часто дешевле проверить в ручную
- автоматизировать все тесты невозможно
- в лучшем случае процессы могут быть соотнесены с существующими системными тестами, которые как было сказано выше -- часто тоже не полностью автоматизированы





Инструменты для SIT/E2E тестирования, которыми не будут пользоваться на фазе UAT

Стэк для написания тестов разработчиком может быть очень разнообразный

- Frameworks для организации Unit тестирования: JUnit, pyUnit, TestNG, SpringTest, etc.
- Frameworks для организации моков и стабов: mockito и десятки других
- Frameworks для организации тестирования UI: Selenium WebDriver, Selenide, TestComplete, Ranorex, etc.
- Frameworks для BDD\TDD: Spock, Cucumber, etc.
- сотни библиотек для обработки JSON, XML, CSV, etc.

Категорий инструментов, которые не связаны с написанием кода тестов, меньше:

- отправка http (и не только) запросов: SoapUI (https://www.soapui.org/) , Postman (https://www.postman.com/) , JMeter (https://jmeter.apache.org/) , etc.
- анализ траффика: Fiddler (https://www.telerik.com/fiddler) , Wireshark (https://www.wireshark.org/) , etc.

Тестирование Баз данных (БД, DB)

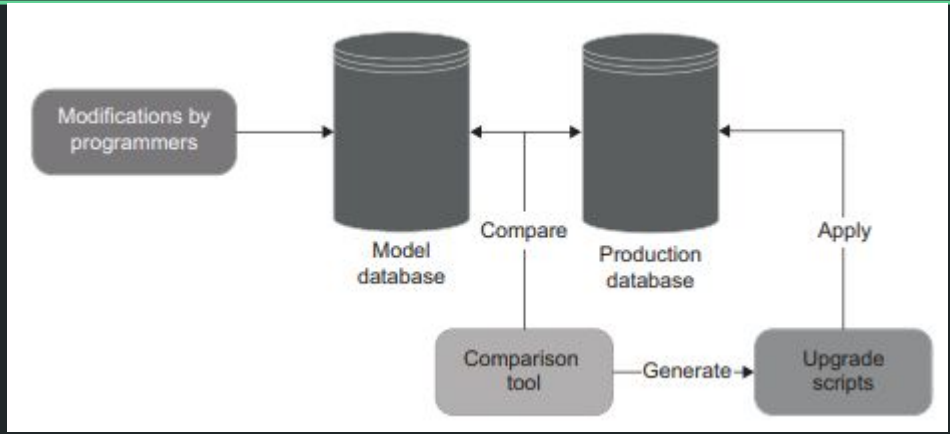
База данных -- внепроцессная зависимость, находится вне контекста SUT (system under test, объект тестирования)
Можно выделить несколько подходов к включению БД в тестовый контур:

- Хранить схему БД в системе контроля версий (НЕ ВСЮ БД ЦЕЛИКОМ!)
- Использовать тестовую БД для каждого разработчика (общую или персональную)
- Реализовать механизм миграции данных

Хранить модель БД -- антипаттерн:

- нет истории изменений
- нет golden source

Сложность хранения схемы: нужно генерить данные



Тестирование Баз данных (БД, DB)

Для подготовки схемы БД необходимы:

- SQL скрипты с созданием таблиц, представлений, индексов, курсоров, хранимых процедур и т.д. и т.п.
- Справочная информация (reference data) -- необходимая информация для инициализации объекта (например БД), не может быть изменена без нанесения вреда целостности объекта, который требуется инициализировать. Пример: типы данных для сущностей, связь между сущностями,

---

Сложности использования общей тестовой БД:

- запуски тестов разных разработчиков могут влиять на результаты выполнения
- не обратносовместимые изменения могут заблокировать выполнение тестов или работу БД в целом

Отдельная БД для каждого разработчика -- не всегда целесообразно с точки зрения распределения ресурсов

Тестирование Баз данных (БД, DB)

State based approach -- явное сохранение состояние БД
Отсутствуют мерж-конфликты, сложные компораторы для генерации скриптов на основанним (часто есть готовые от авторов СУБД) разницы между разными базами, в системе контроля версий хранятся явно скрипты по созданию БД

Migration based approach -- явное сохранение структуры данных БД
Написание отдельных скриптов для изменения тестовой БД, эти скрипты не отражают структуры реальной БД, т.к. являются функцией по ее изменению, возможны мерж-конфликты, в системе контроля версий хранятся скрипты апдейтов

	State of the database	Migration mechanism
State-based approach	✓ Explicit	✗ Implicit
Migration-based approach	✗ Implicit	✓ Explicit