

Цели и задачи интеграционного тестирования.  
Расположение фазы интеграционного тестирования в последовательности тестов; предшествующие и последующие виды тестирования ПО.



## Интеграционное тестирование

- Проверяет интерфейсы и взаимодействие модулей (компонент) или систем
  - Вызовы API, сообщения между ОО компонентами
  - Баз Данных, пользовательский графический интерфейс
  - Интерфейсы взаимодействия (сетевые, аппаратные, локальные, .... )
  - Инфраструктурные
- Может проводиться когда два компонента разработаны (спроектированы)
  - Остальные добавляются по готовности

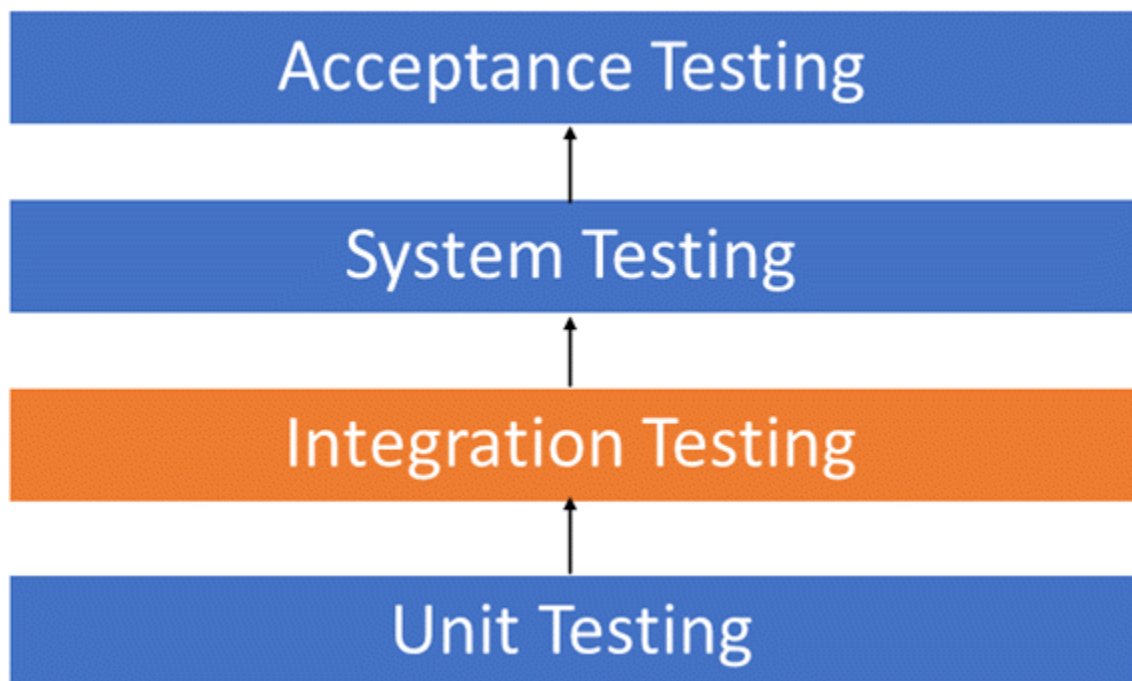
**Интеграционное тестирование** – это тип тестирования, при котором программные модули объединяются логически и тестируются как группа. Как правило, программный продукт состоит из нескольких программных модулей, написанных разными программистами. Целью нашего тестирования является выявление багов при взаимодействии между этими программными модулями и в первую очередь направлен на проверку обмена данными между этими самими модулями.

Его главной задачей является проверка разных модулей системы при их системном объединении. Интеграционное тестирование входит в состав тестирования белого и черного ящика.

- Поскольку, как правило, модули разрабатываются разными специалистами, их понимание и логика программирования могут отличаться. Тут интеграционное тестирование становится необходимым для проверки взаимодействия модулей между собой.
- Во время разработки модуля заказчики часто меняют требования, и если у вас сжатые сроки требования могут попросту не успеть пройти модульное тестирование, и, следовательно, системная интеграция может пройти с помехами. Опять получается, что от интеграционного тестирования не убежать.
- Интерфейсы программных модулей с базой данных могут быть ошибочными
- Внешние аппаратные интерфейсы, если таковые имеются, могут быть ошибочными
- Неправильная обработка исключений может вызвать проблемы.

Уровни тестирования:

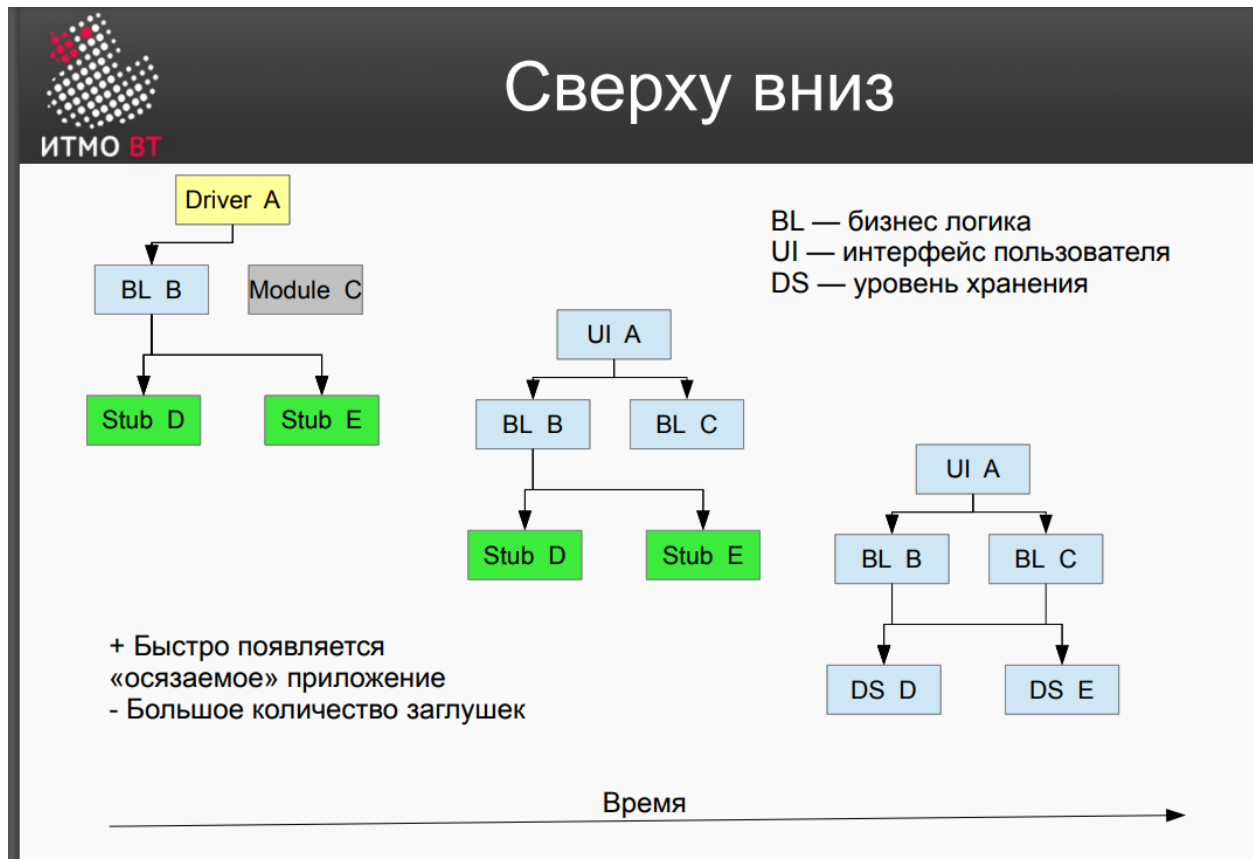
1. Модульное тестирование
2. **Интеграционное тестирование**
3. Системное тестирование
4. Приемочное тестирование



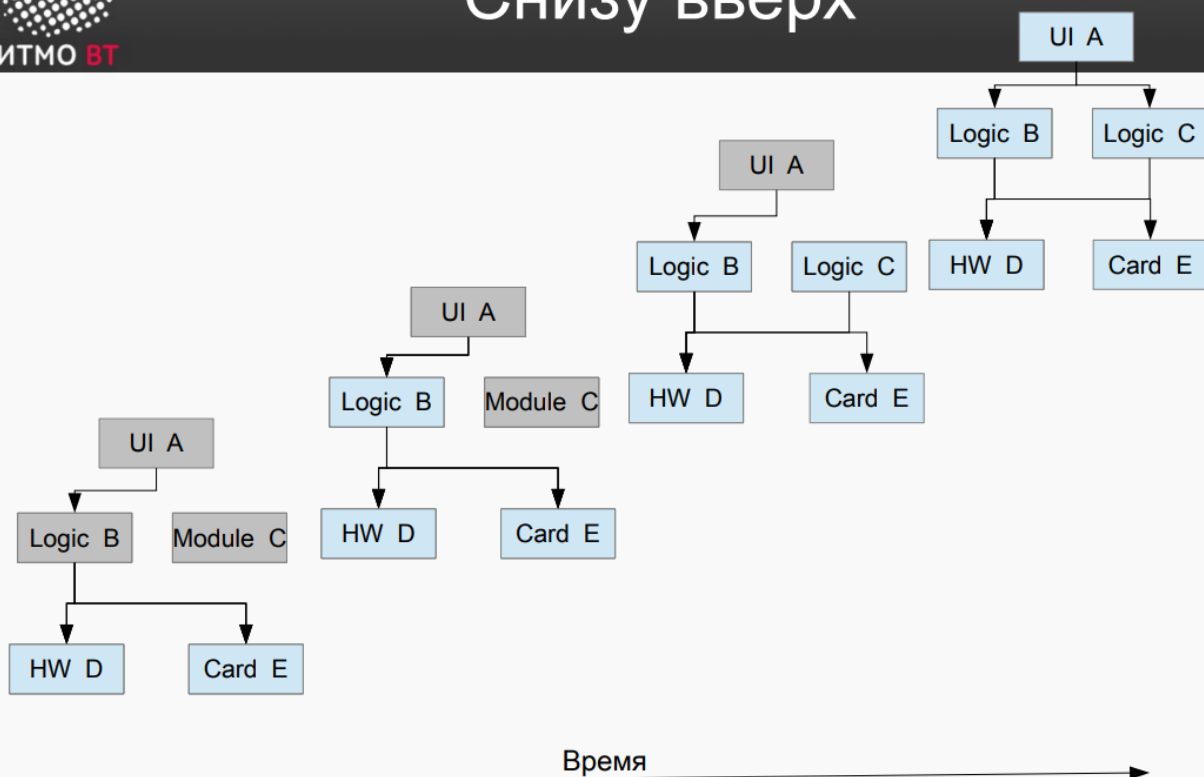
5.

# Алгоритм интеграционного тестирования.

1. Подготовка плана интеграционных тестов
2. Разработка тестовых сценариев.
3. Выполнение тестовых сценариев и фиксирование багов.
4. Отслеживание и повторное тестирование дефектов.
5. Повторять шаги 3 и 4 до успешного завершения интеграции.



# Снизу вверх



- Функциональная (end to end) — по одной функции
  - Собрали 1 сценарий UI-Логика-БД, добавили еще один такой-же
- Ядро (backbone)
  - Экран, клавиатура, мышь работают с минимальными функционалом
  - Добавить цвета на экран, колесо прокрутки ...
- Большой взрыв (big bang)
  - Собрать все вместе и молиться

Концепции и подходы, используемые при реализации интеграционного тестирования.

### Стратегии, методологии и подходы в интеграционном тестировании

Программная инженерия задает различные стратегии интеграционного тестирования:

- Подход Большого взрыва.
- Инкрементальный подход:
  - Нисходящий подход (сверху вниз)
  - Подход «снизу вверх»
  - Сэндвич – комбинация «сверху вниз» и «снизу вверх»

Ниже приведены различные стратегии, способы их выполнения и их ограничения, а также преимущества.

## Подход Большого взрыва

Здесь все компоненты собираются вместе, а затем тестируются.

Преимущества:

- Удобно для небольших систем.

Недостатки:

- Сложно локализовать баги.
- Учитывая огромное количество интерфейсов, некоторые из них при тестировании можно запросто пропустить.
- Недостаток времени для группы тестирования, т.к тестирование интеграции может начаться только после того, как все модули спроектированы.
- Поскольку все модули тестируются одновременно, критические модули высокого риска не изолируются и тестируются в приоритетном порядке. Периферийные модули, которые имеют дело с пользовательскими интерфейсами, также не изолированы и не проверены на приоритет.

## Инкрементальный подход

В данном подходе тестирование выполняется путем объединения двух или более логически связанных модулей. Затем добавляются другие связанные модули и проверяются на правильность функционирования. Процесс продолжается до тех пор, пока все модули не будут соединены и успешно протестированы.

Поэтапный подход, в свою очередь, осуществляется двумя разными методами:

- Снизу вверх
- Сверху вниз

## Заглушка и драйвер

Инкрементальный подход осуществляется с помощью фиктивных программ, называемых заглушками и драйверами. Заглушки и драйверы не реализуют всю логику программного модуля, а только моделируют обмен данными с вызывающим модулем.

Заглушка: вызывается тестируемым модулем.

Драйвер: вызывает модуль для тестирования.

## Интеграция «снизу вверх»

В восходящей стратегии каждый модуль на более низких уровнях тестируется с модулями более высоких уровней, пока не будут протестированы все модули. Требуется помощь драйверов для тестирования

Преимущества:

- Проще локализовать ошибки.
- Не тратится время на ожидание разработки всех модулей, в отличие от подхода Большого взрыва.

Недостатки:

- Критические модули (на верхнем уровне архитектуры программного обеспечения), которые контролируют поток приложения, тестируются последними и могут быть подвержены дефектам.
- Не возможно реализовать ранний прототип

## **Интеграция «сверху вниз»**

При подходе «сверху вниз» тестирование, что логично, выполняется сверху вниз, следуя потоку управления программной системы. Используются заглушки для тестирования.

Преимущества:

- Проще локализовать баги.
- Возможность получить ранний прототип.
- Критические модули тестируются в приоритете; основные недостатки дизайна могут быть найдены и исправлены в первую очередь.

Недостатки:

- Нужно много заглушек.
- Модули на более низком уровне тестируются неадекватно (???)

## **Сэндвич (гибридная интеграция)**


Эта стратегия представляет собой комбинацию подходов «сверху вниз» и «снизу вверх». Здесь верхнеуровневые модули тестируются с нижнеуровневыми, а нижнеуровневые модули интегрируются с верхнеуровневыми, соответственно, и тестируются. Эта стратегия использует и заглушки, и драйверы.

# Программные продукты, используемые для реализации интеграционного тестирования.

## Использование JUnit для интеграционных тестов.

Mockito - фреймворк для работы с заглушками.

Mockito позволяет создать одной строчкой кода так называемый mock (что-то вроде основы для нужной заглушки) любого класса. Для такого mock сразу после создания характерно некое поведение по умолчанию (все методы возвращают заранее известные значения — обычно это `null` либо `0`). Можно переопределить это поведение желаемым образом. В результате mock и становится заглушкой с требуемыми свойствами.

Заглушки: Mockito

- dummy object — объект, который передается, но его методы никогда не используются
- fake objects — работающий объект с упрощенной реализацией
- stub — частичная реализация объекта или интерфейса, с целью использования его методов
- mock object — простая имплементация, с предопределенными значениями

`Mockito.when`. Этот метод принимает в качестве "параметра" вызов переопределяемого метода mock-объекта (таким образом фиксируется определяемое воздействие) и возвращает объект типа `OngoingStubbing`, позволяющий вызвать один из методов семейства `.then...` (так задается реакция на это воздействие).

Если есть параметры:

Если нужно задать реакцию на любой вызов этого метода независимо от аргументов, я должен воспользоваться методом `Mockito.any`:

```
Mockito.when(dataService.getDataItemById(any()))  
    .thenReturn("dataItem");
```



Если же требуется, чтобы mock реагировал только на определённое значение аргумента, можно использовать непосредственно это значение или методы `Mockito.eq` (когда речь об эквивалентности) либо `Mockito.same` (когда требуется сравнение ссылок):

```
Mockito.when(dataService.getDataItemById("idValue"))
    .thenReturn("dataItem");
// or
Mockito.when(dataService.getDataItemById(Mockito.eq("idValue"))
    .thenReturn("dataItem");
```

Автоматизация интеграционных тестов. ПО, используемое для автоматизации интеграционного тестирования.

МБ это

- Средства автоматизации
  - Открытые: Selenium, Sahi, Watir
  - Коммерческие: от HP, Rational (IBM) ....