

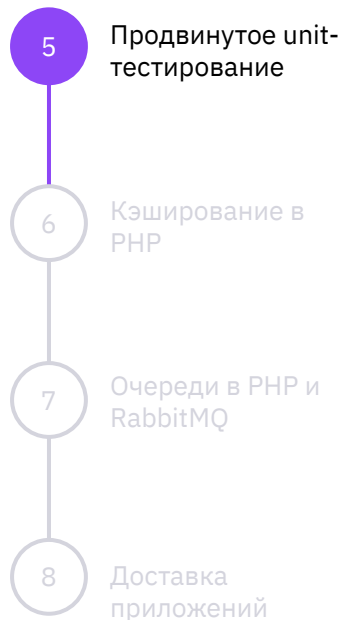
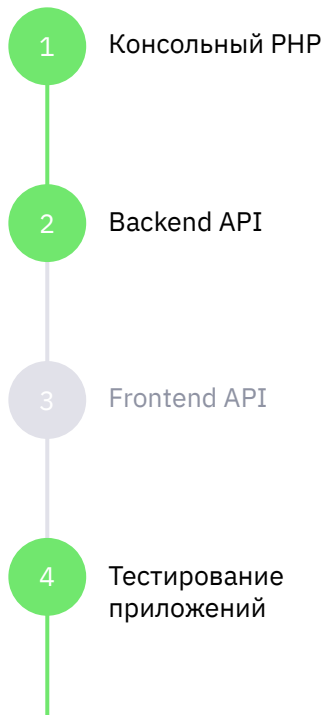
# Продвинутое-unit тестирование

Урок 5





## План курса



WELCOME



## Что будет на уроке сегодня



Викторина, которая построена на основании реальных вопросов, которые задают на собеседовании



Имитация работы выполнения заданий от тимлида



Улучшение unit-тестов, которые написали на прошлом уроке



# Викторина

Построена на основании реальных вопросов,  
которые задают на собеседовании



Преподаватель демонстрирует вопросы викторины, зачитывает их, а студенты пишут ответы в чате



## 1. О каком уровне тестирования идет речь?

1. Приемочное

2. Системное

3. Функциональное

4. Unit

	?
<b>Цель</b>	Валидация функциональных требований модуля
<b>Кто проводит</b>	Разработчики
<b>Окружение</b>	dev
<b>Периодичность</b>	Несколько раз в день
<b>Длительность</b>	От нескольких секунд до нескольких минут
<b>Автоматизация</b>	Всегда автоматизируется



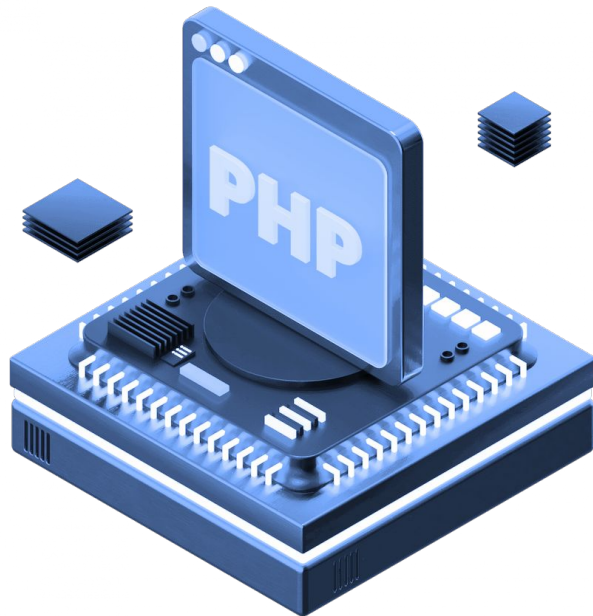


## 2. Что такое контракт в программировании?

1. Формальные спецификации, которые описывают наши ожидания от функции, метода или класса, а также их взаимодействие с окружающей средой и другими компонентами

2. Это, когда класс должен иметь только одну причину для изменения, то есть его методы должны быть связаны друг с другом концептуально и выполнять плюс-минус одно и тоже

3. Код, который можно писать плохо





### 3. Что нужно тестировать: реализацию или контракт?

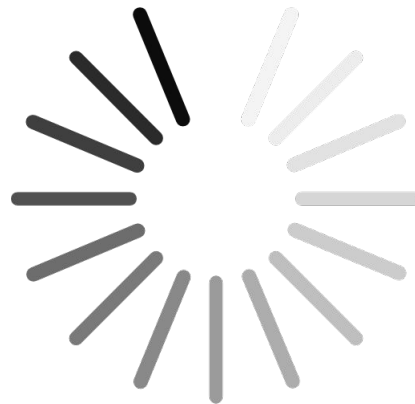
\*Вопрос без вариантов ответа\*



Подсказка: загляните в конспект



Совет: напишите ответ в  
чат или проговорите его





#### 4. Взгляните на перечисления. О каком принципе идет речь?

1. Принцип F.I.R.S.T.

2. Принцип AAA

3. Принцип F.I.F.O.

4. Принцип L.I.F.O.

Быстрота

Независимость

Повторяемость

Очевидность

Своевременность



Подсказка: загляните в конспект



Совет: напишите ответ в чат или проговорите его





## 5. Какие вы помните антипаттерны при проектировании тестов?

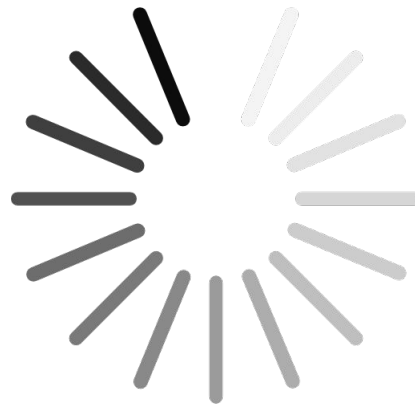
\*Вопрос без вариантов ответа\*



Подсказка: загляните в конспект



Совет: напишите ответ в чат или проговорите его



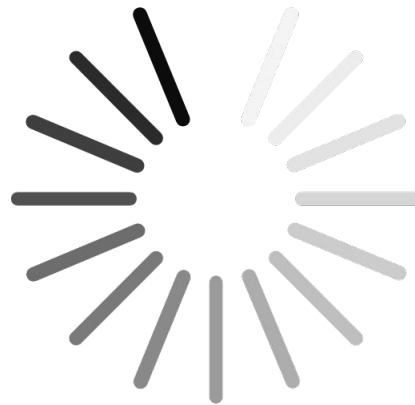


## 6. Что хуже: отсутствие тестов или плохие тесты?

1. Отсутствие тестов

2. Плохие тесты

3. И то, и то





## 7. Каких тестовых двойников вы помните?

1. Dummy, fake, stub, mock и spy

2. Testing, contract, API

3. DRY, AAA, KISS

3. While, function, done

Дополнительный вопрос:

Для чего они используются?



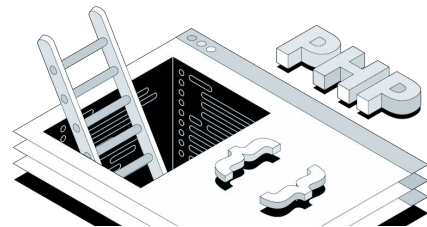


## 8. Какой правильный путь именования тестов?

1. test[ожидаемое поведение][тестируемый метод][сценарий]

2. test[тестируемый метод][сценарий][ожидаемое поведение]

3. test[сценарий][ожидаемое поведение][тестируемый метод]





## Перечень вопросов, которые также могут также задать:

1. Какие вы помните антипаттерны проектирования приложения, которые мешают написанию хороших unit тестов?
2. Что такое чистая функция? Как вы считаете, должен ли unit-тест стремиться быть чистой функцией? Какие для этого есть инструменты?
3. В чем разница между Mock и stub? Какими бывают их подтипы и в чем между ними разница?
4. Как мы можем создать отчет о покрытии кода тестами?
5. Что такое мутационное тестирование? Как мы можем создать отчет о мутационном тестировании?

**Рекомендую на них ответить самостоятельно**  
**Ответы к ним вы найдете в лекции 🙌**



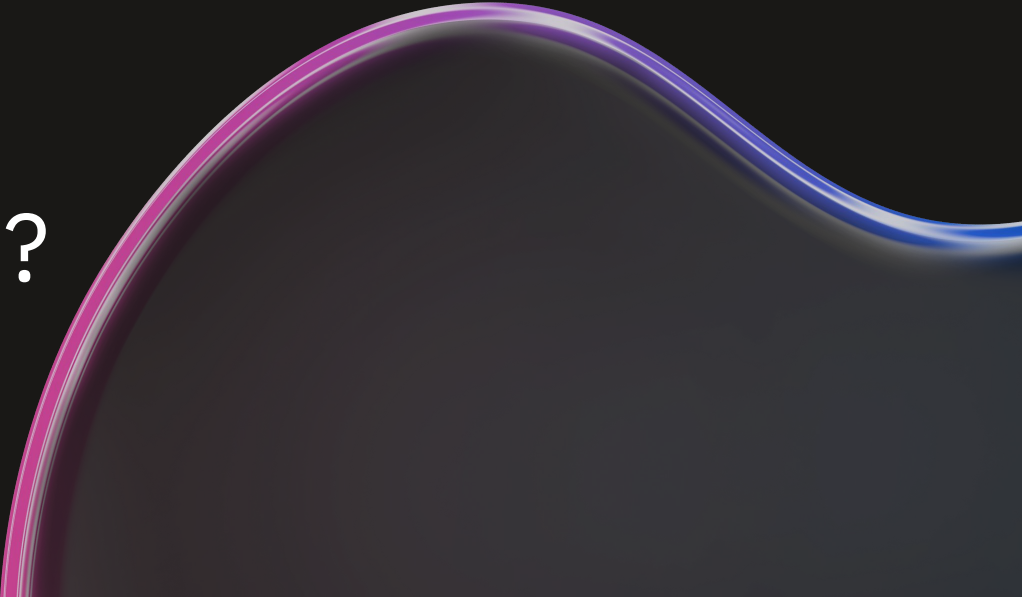


# Разбор домашнего задания





Кому удалось  
выполнить  
дополнительные  
задания № 3, 4, 5?







# Практика



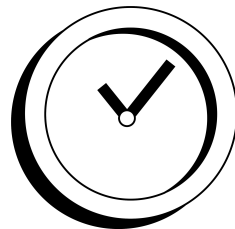


## Задание 1. Разбор ТЗ с тимлидом

На прошлом занятии мы определили много контрактов нашего кода, на которые мы хотим писать unit-тесты.

На часть из них мы уже написали тесты, сегодня нам нужно написать тесты на оставшуюся часть, используя новый функционал тестовых двойников и рефакторинг.

Применим новые знания и определим, что нам нужно сделать, чтобы написать все тесты, которые мы хотим.



15 мин.



## Задание 2. Рефакторинг кода

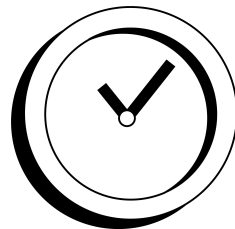
Целью рефакторинга для нас будет являться **оптимизация структуры кода таким образом, чтобы мы могли без проблем создать unit-тесты.**

Для этого мы проработаем пункт лекции “Антипаттерны проектирования”.

Наша задача — **классы команд сделать, по сути, фабриками или контроллерами.**

В них не будет производиться никакой логики, только – создание экземпляров классов и передача управления.

Это поможет вынести получение знаний в конструктор новых классов, после чего их будет просто тестировать, передавая в конструктор тестовых двойников.



20 мин.



## Задание 2. Рефакторинг кода (продолжение)

Исходя из новых классов контракты, которые мы хотим тестировать немного меняются.

Более того, благодаря грамотному рефакторингу и тому, что наши классы теперь следуют принципу SOLID, нам необязательно открывать для изменения приватные методы.

Мы можем тестировать класс целиком, вызывая только один его метод, при этом тестируя различные сценарии.

Давайте еще раз посмотрим на классы и определим, какие сценарии мы будем тестировать.

[Установить таймер](#)



# Перерыв?

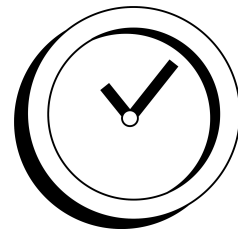
Голосуйте в чате





## Задание 3. Создаем тесты, используя тестовых двойников

Начинаем писать тесты по заданным сценариям, **кроме TgEventsTest**, он остается для домашнего задания из-за большого объема работы.

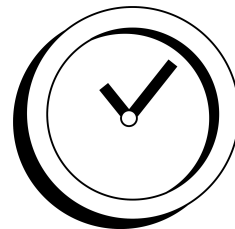


30 мин.



## Задание 4. Работа с тестовым API

Начинаем создавать тестовый API. Ваша первоочередная задача – **вынести все DataProvider в отдельные классы.**



10 мин.

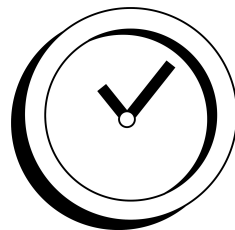




## Задание 5. Создаем отчет о покрытии кода тестами и о мутационном тестировании

### Алгоритм действий:

1. Устанавливаем Xdebug
2. Далее для всех классов тестов нужно определить, какие классы они покрывают
3. Не лишним будет удалить echo в EventSender, а так же создать phpunit.xml
4. Рекомендую также сразу поменять значение в phpunit.xml
5. Далее можно создать отчет
6. Не забыв при этом добавить директорию covers\_html в .gitignore. Далее создаем отчет о мутационном тестировании  
Файл infection.json
7. Устанавливаем infection
8. Затем запускаем



10 мин.



# Домашнее задание





## Домашнее задание

### Задание

1. Сделайте новую ветку из той, которую мы создали на прошлом уроке. Это нужно для того, чтобы мы могли работать с кодом из прошлого урока.
2. Загрузите весь код из сегодняшнего урока в Git в новую ветку, создайте новый pull request. Пришлите на проверку ссылку на pull request.
3. ✖ Создайте тесты на оставшийся класс TgEvents. Обратите внимание, что тут придется работать как с исходящими, так и с входящими зависимостями. Вам придется иметь дело как с Mock, так и с Stub.



В качестве решения приложить:

- ссылку на pull request в вашем репозитории с домашним заданием

✖ – дополнительное задание 💪









# Подведем итоги





## Подведение итогов

-  потренировались отвечать на вопросы про тестирование на собеседовании
-  научились работать в команде, понимать требования и выполнять задачи, поставленные тимлидом
-  научились разбивать задачу на более мелкие подзадачи, чтобы легче было управлять проектом и следить за прогрессом
-  поработали с тестовыми двойниками (Mock Stub)
-  создали тестовое API
-  создали отчеты о покрытии кода



**Спасибо за внимание**