



# Как полюбить модульное тестирование

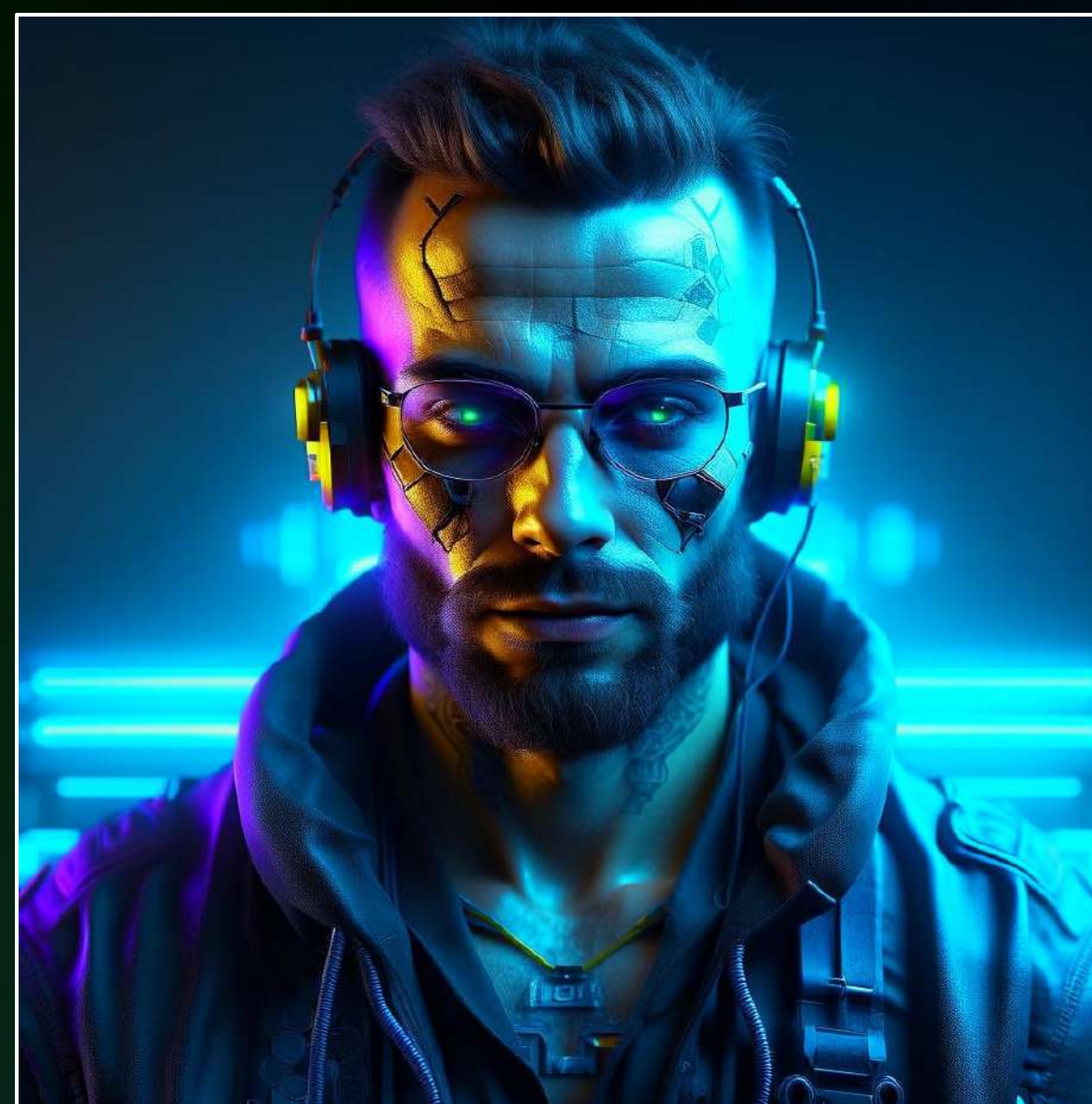
Обратная сторона TDD

**Вячеслав Чернышов**  
Backend-разработчик, СберТех

# Test-Driven Development



Пишу тесты  
кое-как



Покрываю тестами  
80% кода



Использую Test-Driven  
Development



**Вячеслав Чернышов**

Backend-разработчик  
СберТех, Platform V Flow

## Platform V

Облачная платформа  
для разработки бизнес-решений

Набор продуктов для быстрого создания  
и легкого масштабирования  
промышленных приложений любой сложности

Pangolin SE | DataGrid | Works | DataSpace  
SOWA | IAM SE | Synapse | Flow

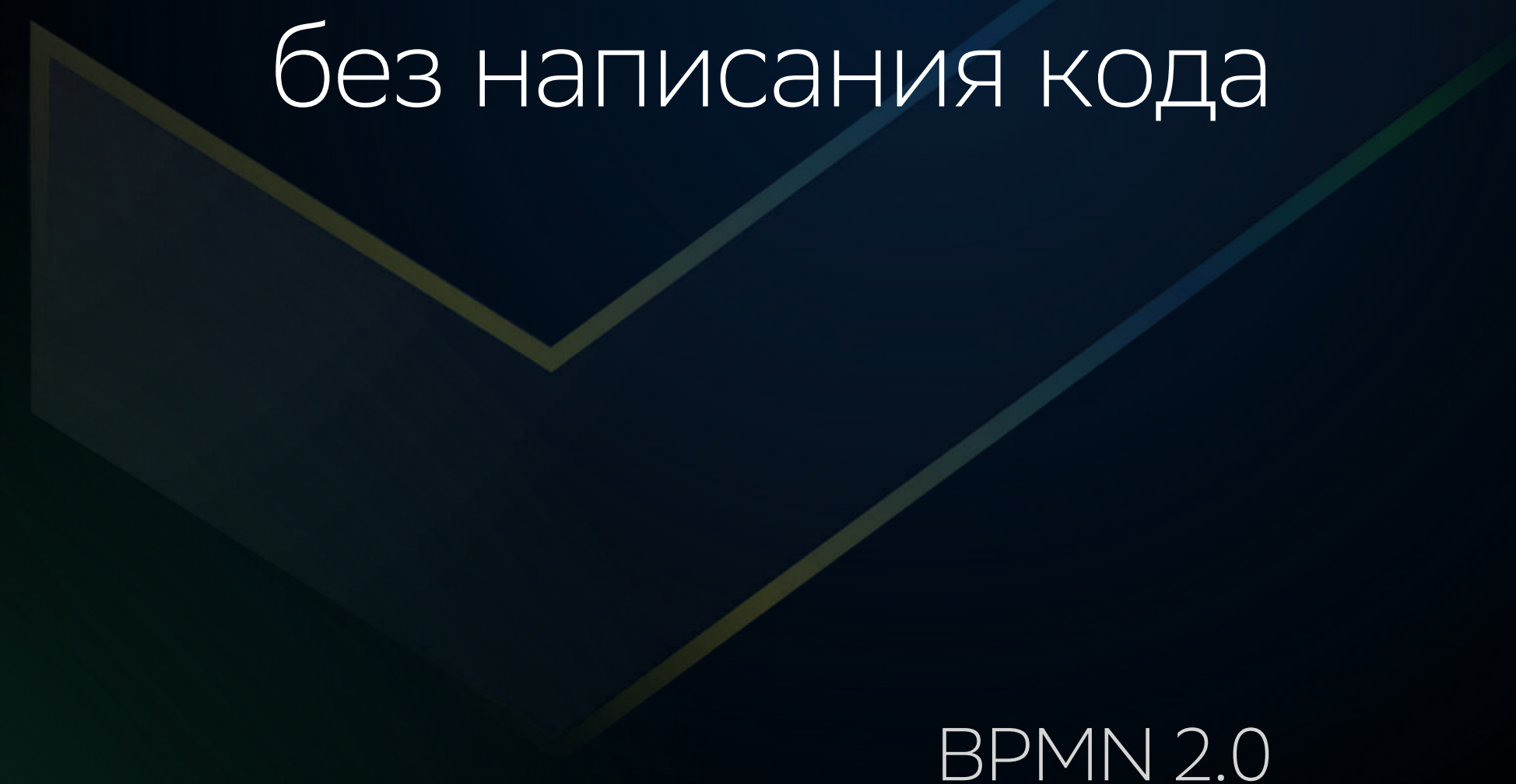


**Вячеслав Чернышов**

Backend-разработчик  
СберТех, Platform V Flow

## Platform V Flow

Автоматизация процессов  
без написания кода



BPMN 2.0

Low-code платформа

Встроенный дизайнер процессов



**Вячеслав Чернышов**

Backend-разработчик  
СберТех, Platform V Flow



Профиль на Хабре



**Вячеслав Чернышов**

Backend-разработчик  
СберТех, Platform V Flow



Канал в Telegram



**Вячеслав Чернышов**

Backend-разработчик  
СберТех, Platform V Flow

## Краткий профессиональный путь

Казахский стартап 2017 – 2018

---

Букмекерская компания с  
сильной командой разработки 2018 – 2019

---

Галера 2019

---

IT-дочка госкорпорации 2019 – 2022

---

Сбер 2022 – н. в.

---



**Вячеслав Чернышов**

Backend-разработчик  
СберТех, Platform V Flow

**В общем, мне повезло  
участвовать в самых разных  
проектах.**

**Разговор будет предметный.**

Как и многие,  
я сопротивлялся TDD.

Пока не начал читать книги Роберта Мартина.

# Test-Driven Development



**Роберт Мартин**

Идеальный программист:  
как стать профессионалом  
разработки

Идеальная работа:  
программирование без прикрас

# Test-Driven Development

## Три закона **TDD**

Сначала – тест,  
потом –  
функциональность

Не пишите код, пока не напишете тест.

Один тест проверяет  
одну функцию

Добейтесь прохождения теста, написав  
необходимый код.

Функция  
обеспечивает только  
свой тест

Написанный код должен обеспечивать  
прохождение своего теста, и ничего  
более.

# Test-Driven Development:

## Основные преимущества

### Смелость

Вы можете вносить любые изменения.  
Ваши тесты проконтролируют работоспособность кода.

### Документация

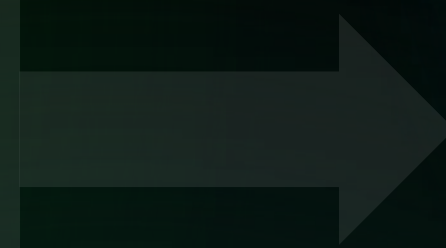
Если по коду можно понять, что он делает, то по тестам можно понять, как его использовать.

### Архитектура

Правильно написанные тесты обеспечивают правильную архитектуру.

Как Test-Driven Development  
влияет на разработку приложения?

**ЧТО**  
МЫ ХОТИМ?



**КАК**  
МЫ ЭТО СДЕЛАЕМ?

# Этапы реализации любой фичи

## 1 этап

### Функциональное требование

Детальное описание поведения системы:  
функции и поведение.

Функция описывает, **что** делает система, а поведение – **как** система выполняет функцию.

## 2 этап

### Техническое задание

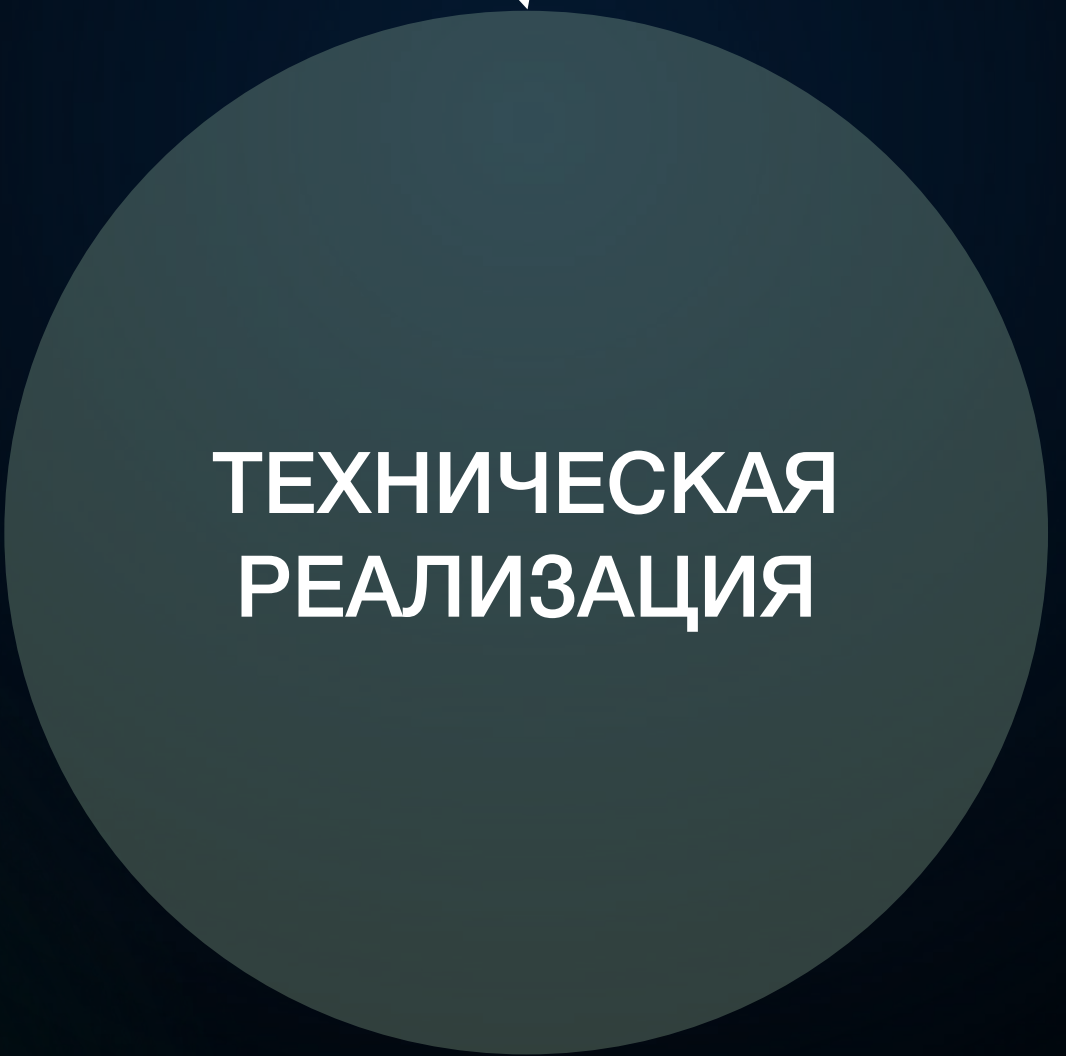
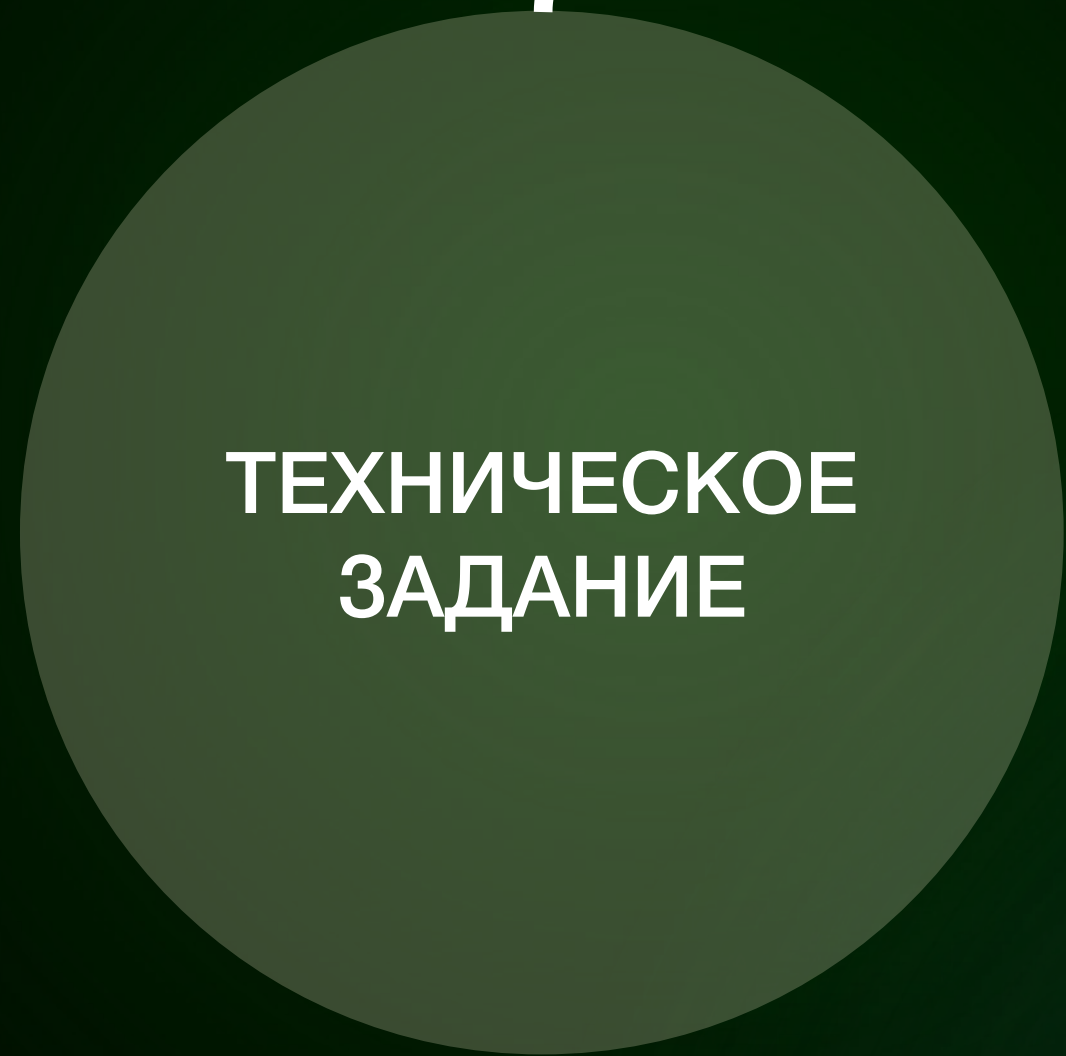
Подробное описание будущей реализации функционального требования.

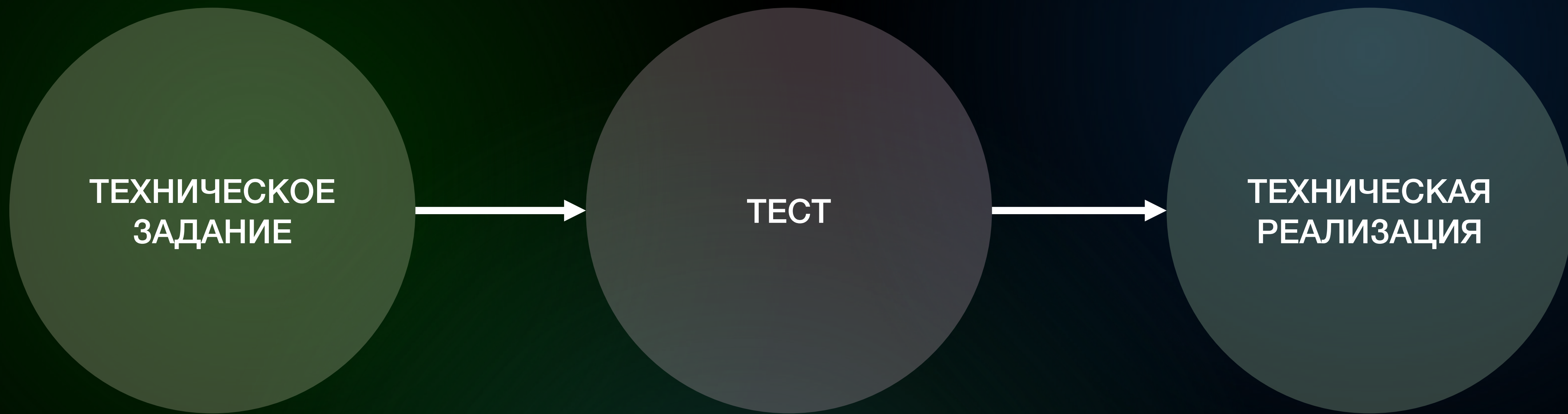
Включает описание всех компонентов, которые будут созданы или изменены.

## 3 этап

### Техническая реализация

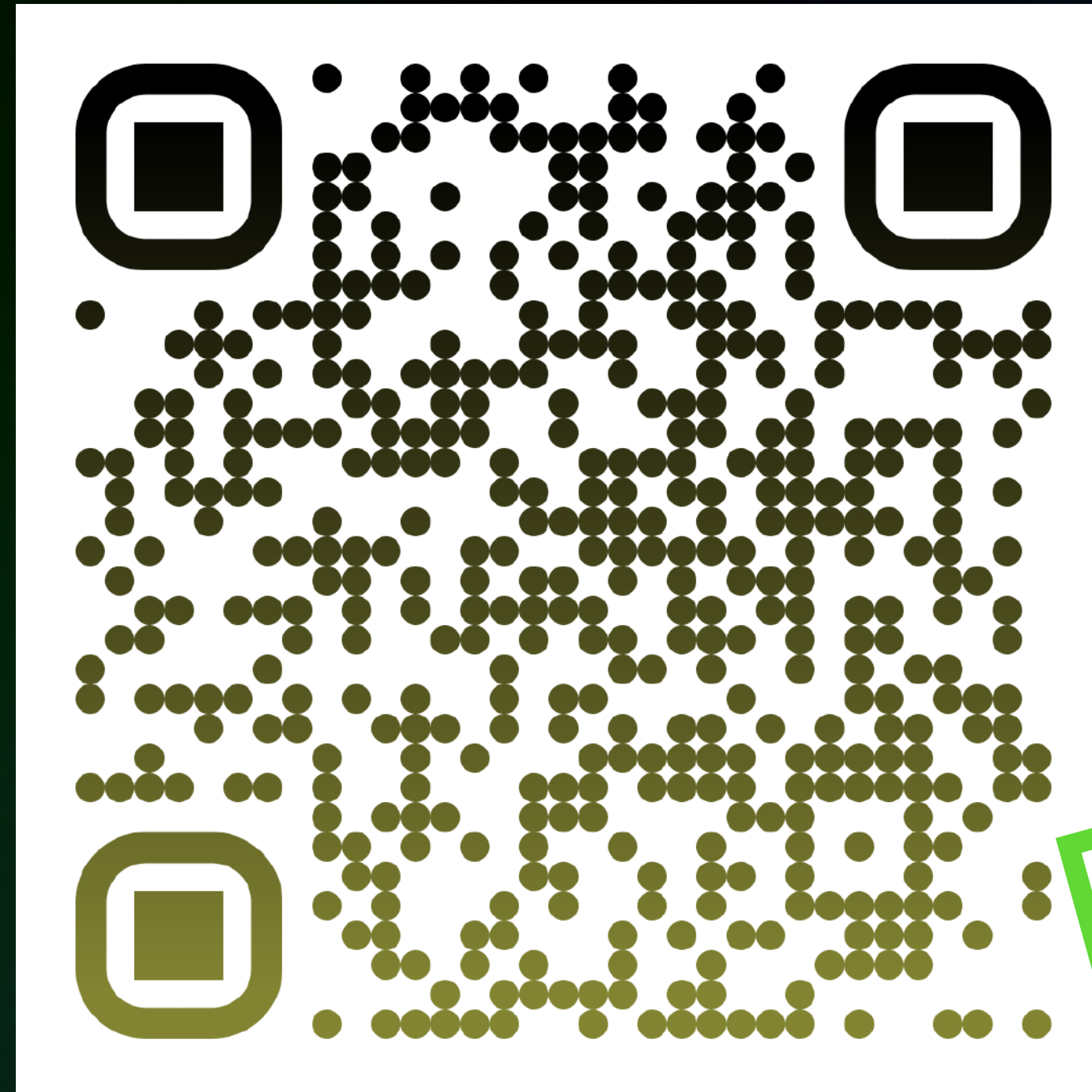
Конкретные изменения в коде, сделанные разработчиком при выполнении технического задания и реализующие функциональность, описанную в техническом задании.





На каком этапе мы **пишем тесты**?

# Этапы разработки приложения



Есть статья на  
Хабре

Разработка приложения:  
этапы реализации от проекта до релиза

# Этапы разработки приложения

## 1 этап



**Проектирование предметной области**

### **Бизнес-сущности**

Описание – визуализация - код

## 2 этап



**Проектирование логического скелета**

### **Наши любимые интерфейсы**

Архитектура на основе стабильных компонентов

## 3 этап



**Реализация логики**

### **Test-Driven Development**

Сначала понимаем, чего хотим, потом – как этого добиться.

## 4 этап



**Интеграция с внешним миром**

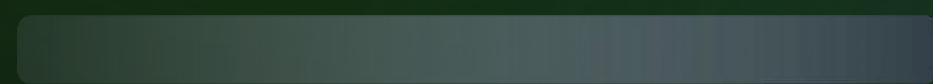
### **Слой DAO**

Клиенты, контроллеры, репозитории и прочие внешние интерфейсы

*2 этап*



*Проектирование  
логического  
скелета*



*Декларация  
возможностей*

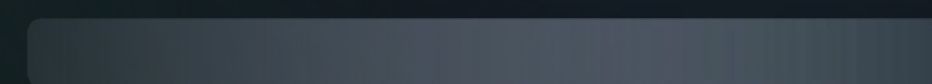
*Тесты*



*3 этап*



*Реализация логики*



*Реализация  
функциональности*

Теперь давайте  
посмотрим на наш пэт-проджект.

User

Пользователь.

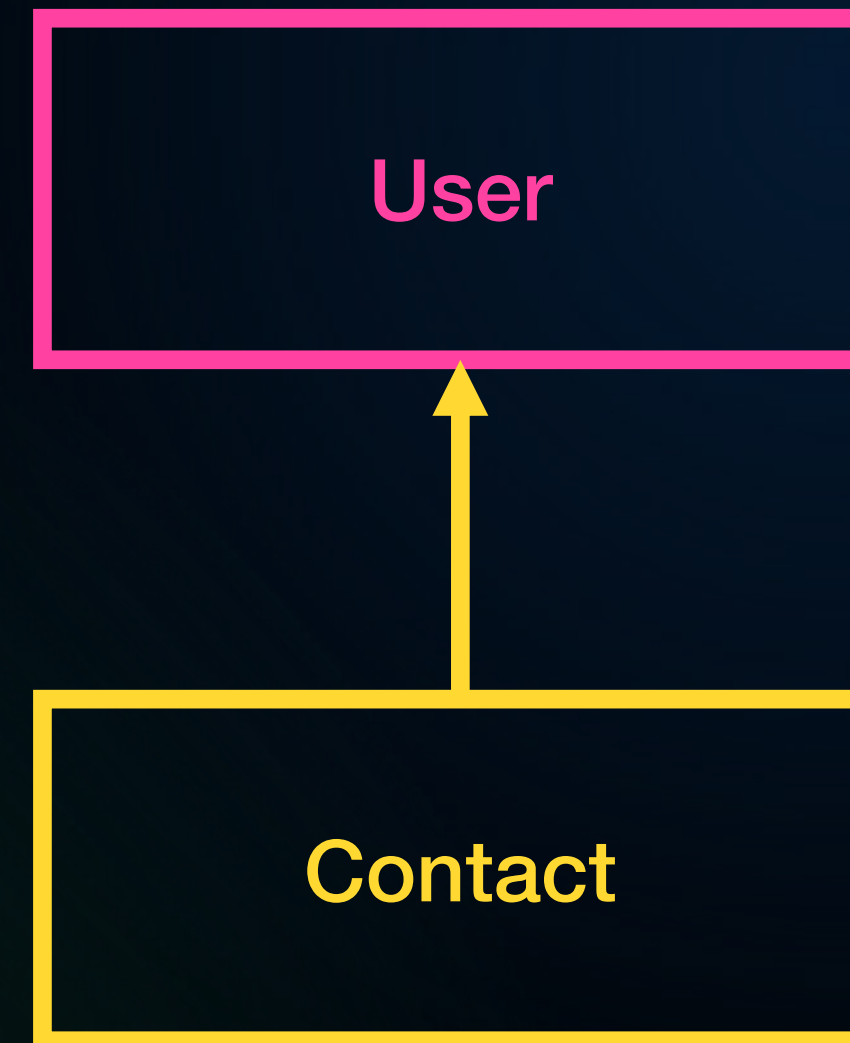
User

User

Пользователь.

Contact

Контакт (телефон, почта и т. д.)



# Этапы написания теста

## Этап 1. Подготовка.

Подготовка окружения, генерация данных.

## Этап 2. Исполнение.

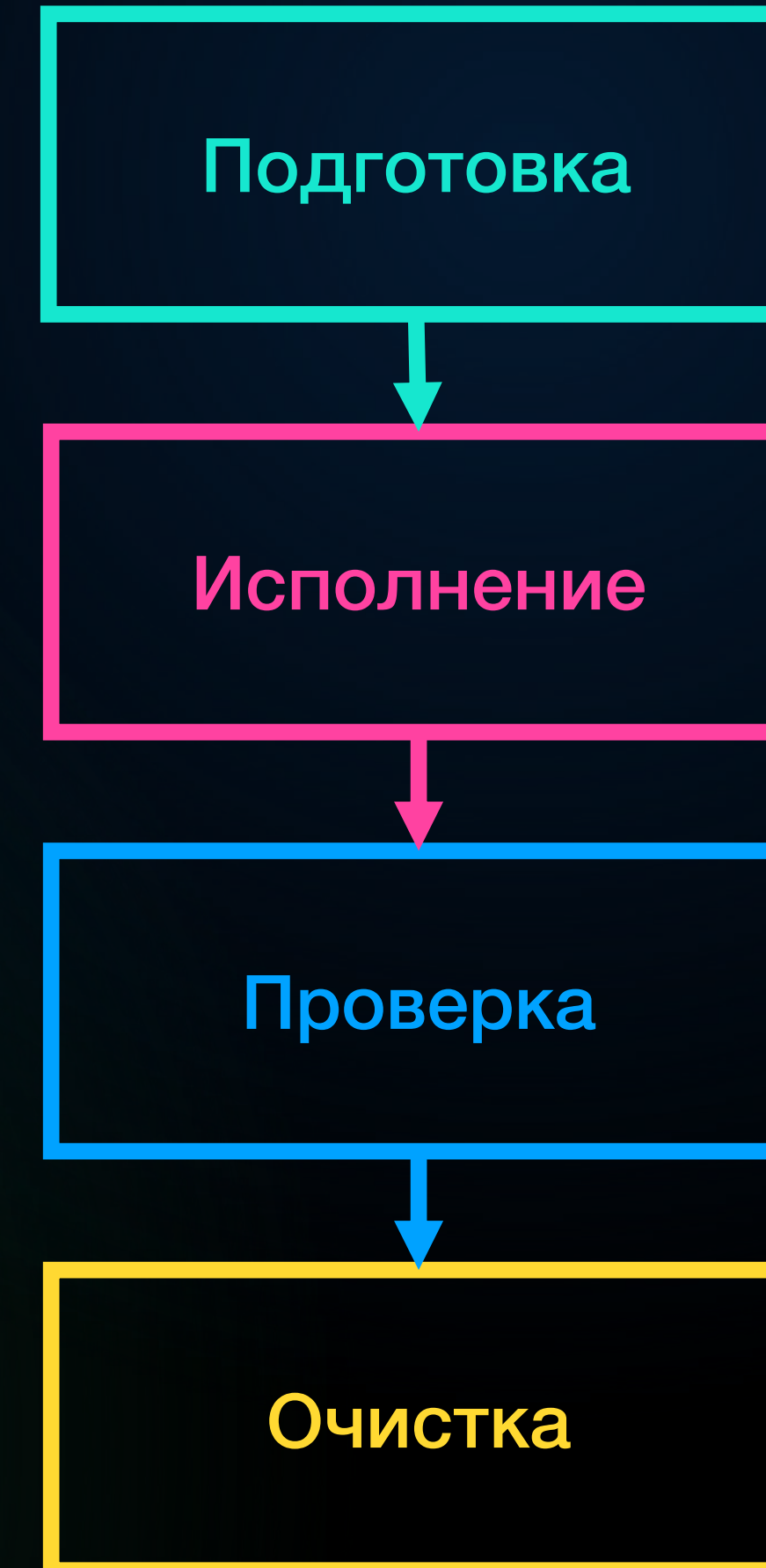
Вызов функции, исполняющей тестируемую логику.

## Этап 3. Проверка.

Проверка возвращаемого результата на соответствие.

## Этап 4. Очистка.

Очистка данных до первоначального состояния.





# Этапы разработки приложения

## 1 этап



**Проектирование предметной области**

### **Бизнес-сущности**

Описание – визуализация - код

## 2 этап



**Проектирование логического скелета**

### **Наши любимые интерфейсы**

Архитектура на основе стабильных компонентов

## 3 этап



**Реализация логики**

### **Test-Driven Development**

Сначала понимаем, чего хотим, потом – как этого добиться.

## 4 этап



**Интеграция с внешним миром**

### **Слой DAO**

Клиенты, контроллеры, репозитории и прочие внешние интерфейсы

Подведём **ИТОГ**

# Тезис первый

Разработка через тестирование **упрощает** поддержку продукта.

С тестами мы не боимся **рефакторинга**. Приложение, покрытое тестами, имеет **меньше багов** и легче проходит последующие этапы тестирования.

# Тезис второй

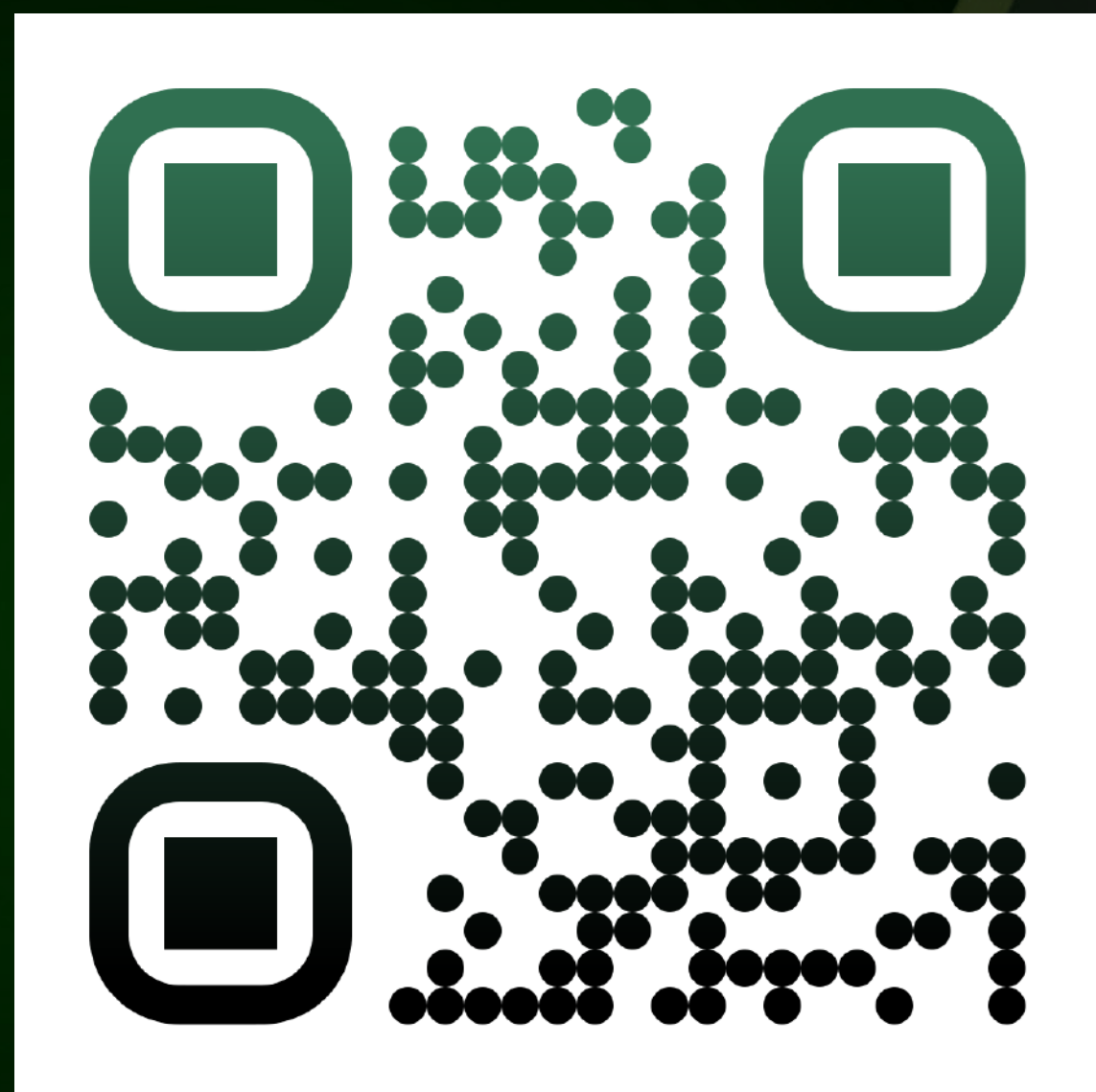
Мок-компоненты позволяют разрабатывать через тестирование **на любом этапе**.

При этом, не приходится нарушать стандартный ход реализации приложения и внепланово писать реализацию компонентов, которые **должны быть разработаны позже**.

# Тезис третий

Компоненты-генераторы пишутся один раз, используются **постоянно**.

Поскольку написание тестов сопровождает разработчика на любом этапе, наличие генераторов объектов более чем **оправдано**.



Канал в Telegram



Профиль на Хабре