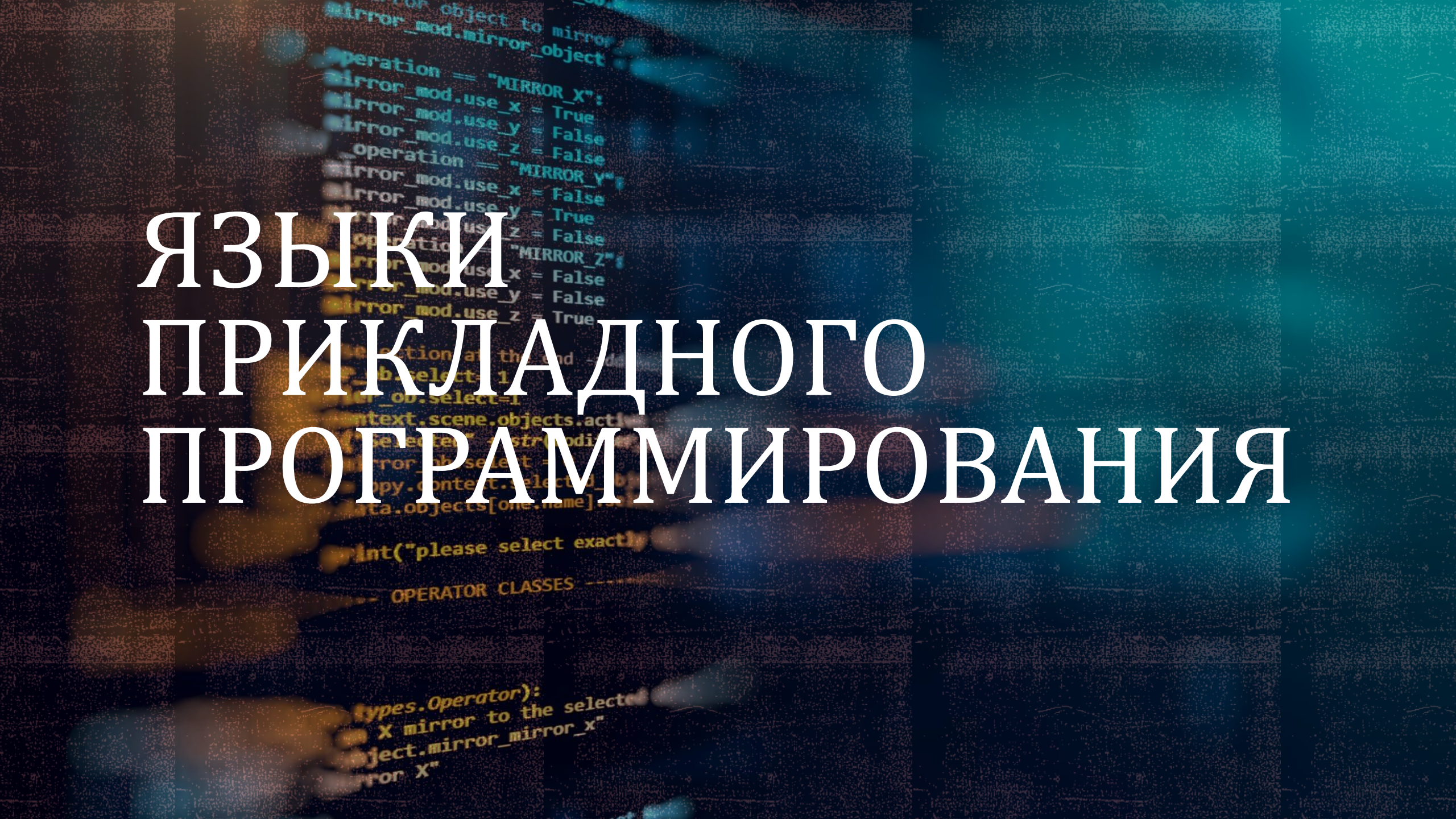


ЯЗЫКИ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ

ТЕСТИРОВАНИЕ ПО

- Тестирование – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

ДЛЯ ЧЕГО НУЖНО ТЕСТИРОВАНИЕ?

ДЛЯ ЧЕГО НУЖНО ТЕСТИРОВАНИЕ?

- **Чтобы определять некорректное поведение ПО и исправлять его**

- Помогает:
  - не сломать старую функциональность при добавлении новой
  - понять, как работает код
  - ускорить стабильную поставку нового функционала пользователям

ДЛЯ ЧЕГО НУЖНО ТЕСТИРОВАНИЕ?

Зачем нужны тесты?

Любой код меняется

Чтение / Создание = [7..200+] : 1

10:1

ПОЧЕМУ ПО ВСЕ РАВНО С БАГАМИ?

- Гарантировать что программа работает корректно практически невозможно.
- При помощи тестирования можно доказать, что программа работает неправильно, однако, доказать что она работает правильно - увы - нет

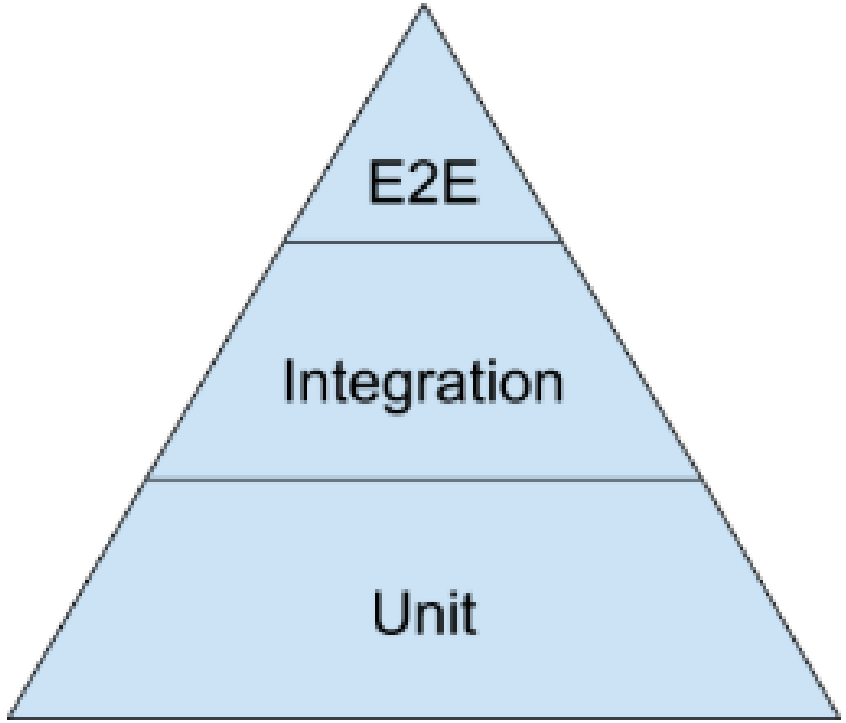
КАКИЕ ВИДЫ ТЕСТИРОВАНИЯ БЫВАЮТ?

РУЧНОЕ VS АВТОМАТИЧЕСКОЕ

- Ручное тестирование - производится тестировщиком без использования программных средств, для проверки программы или сайта путем моделирования действий пользователя

- Автоматическое тестирование - производится с помощью программных средств автоматизации

ВИДЫ ТЕСТИРОВАНИЯ ПО УРОВНЮ АБСТРАКЦИИ



- Модульное (**unit**) тестирование – это проверка корректности работы отдельных модулей программы
- Тестируется корректность работы *всех** нетривиальных методов

Хранение данных

Бизнес логика

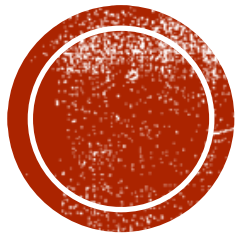
UI

ИНТЕГРАЦИОННОЕ

- Это тестирование корректности работы двух или более модулей в связке друг с другом – их интеграции

- Тестирование работы системы целиком в конкретном пользовательском сценарии – от начала и до конца.

	Модульные	Сквозные
Быстрый		
Надежный		
Позволяет быстро обнаружить ошибки		
Имитирует реального пользователя		



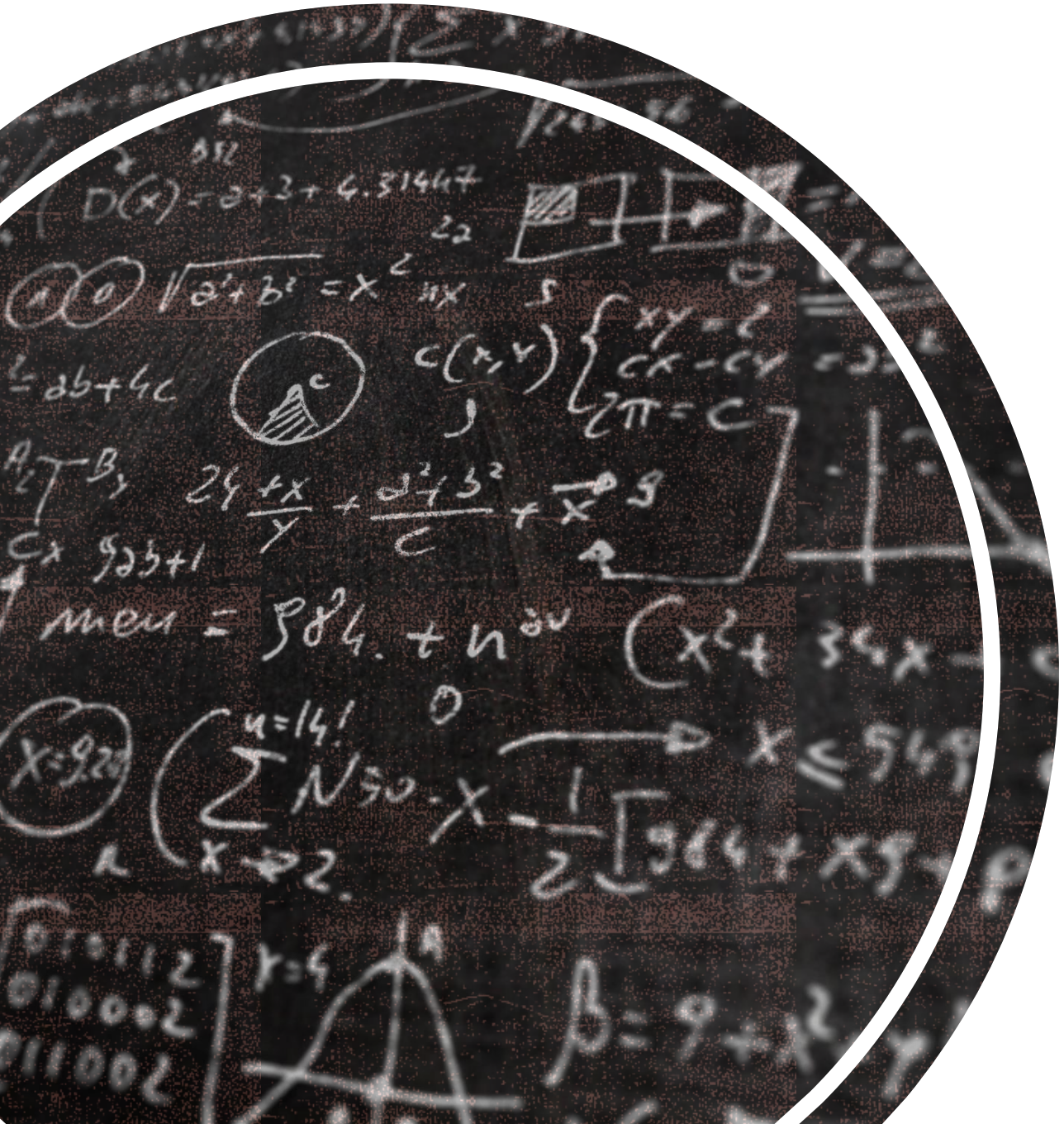
ЕЩЕ НЕСКОЛЬКО ВИДОВ ТЕСТИРОВАНИЯ…

- Тесты, проверяющие, что после внесения изменений старый функционал продолжает работать как ожидается

- Модульные, функциональные и интеграционные тесты могут выступать в роли регрессионного, если запущены после

РЕГРЕССИОННОЕ

- Применяется в основном для тестирования высоконагруженного ПО
- Тестируется работа ПО под большой нагрузкой (например – количество **HTTP** запросов к серверу)



КЛАССЫ ЭКВИВАЛЕНТНОСТИ

- Класс эквивалентности — подмножество из множества допустимых значений, для которых поведение системы считается одинаковым

- ***Сколько тестов нужно написать для каждого класса?***

КЛАССЫ ЭКВИВАЛЕНТНОСТИ

- Для каждого класса эквивалентности достаточно написать 1 тест, также необходимы тесты на границах классов эквивалентности

- Пример: Хотим выдавать скидку 10% на товары, если их 5 и более

КЛАССЫ ЭКВИВАЛЕНТНОСТИ

Решение квадратного уравнения Ax^2 + Bx + C = 0

КЛАССЫ ЭКВИВАЛЕНТНОСТИ

Решение квадратного уравнения Ax^2 + Bx + C = 0

$$ax^2 + bx + c = 0$$

$x = \frac{-b \pm \sqrt{D}}{2a}$, где
$D = b^2 - 4ac$

Пример 1

$D > 0$ – два корня

$x^2 - 5x + 6 = 0$

$$x = \frac{5 \pm \sqrt{5^2 - 4 \cdot 1 \cdot 6}}{2 \cdot 1}$$

$x = 3$ или $x = 2$

Пример 2

$D < 0$ – корней нет

$x^2 - 5x + 7 = 0$

$D = 5^2 - 4 \cdot 7 < 0$

корней нет

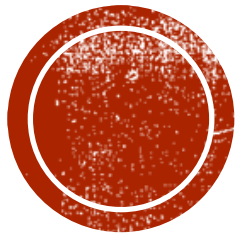
Пример 3

$D = 0$ – один корень

$x^2 - 10x + 25 = 0$

$$x = \frac{10 \pm \sqrt{100 - 4 \cdot 25}}{2}$$

$x = 5$



ПОДРОБНЕЕ О UNIT ТЕСТИРОВАНИИ

КАКИЕ ЗАДАЧИ РЕШАЕТ?

- Определять некорректное поведение модуля и исправлять его
- Помогает не сломать старую функциональность при добавлении новой
- Помогает понять, как работает код (документирование)
- Помогает ускорить стабильную поставку нового функционала пользователям

КАКИЕ ЗАДАЧИ РЕШАЕТ?

ms

Быстро получать **обратную связь** о том, что код **соответствует ожиданиям**

разработчика

пользователя

КАКИЕ ЗАДАЧИ РЕШАЕТ?

- Определять некорректное поведение модуля и исправлять его
- Помогает не сломать старую функциональность при добавлении новой
- Помогает понять, как работает код (документирование)
- Помогает ускорить стабильную поставку нового функционала пользователям

ARRANGE ACT ASSERT

ARRANGE -
ПОДГОТОВКА ДАННЫХ

ACT - ДЕЙСТВИЕ
(КОТОРОЕ ТЕСТИРУЕМ)

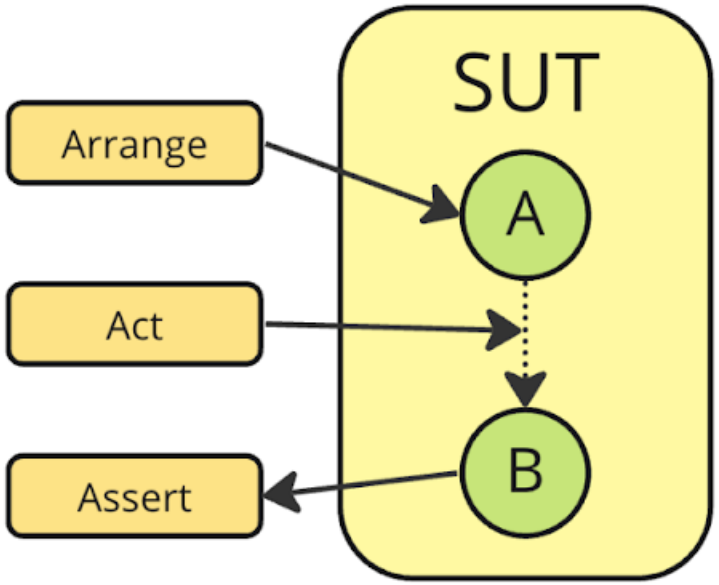
ASSERT – ПРОВЕРКА
КОРРЕКТНОСТИ РАБОТЫ

MOCK, STUB, FAKE

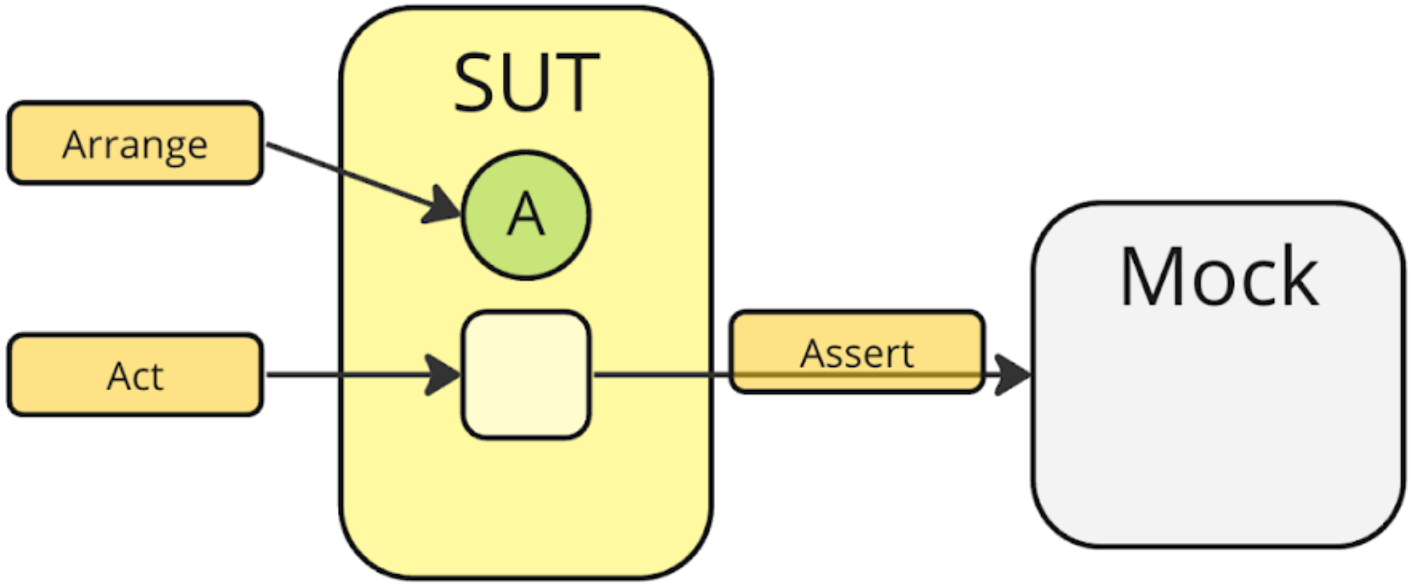
- Используются, чтобы подменить зависимости при тестировании

- **Stub** (заглушка) - реализация объекта для теста – возвращающая фиксированные, известные в тесте результаты
- **Mock –** настраиваемая реализация интерфейса (настраиваются входные, выходные параметры, возможно настроить проверки – вызывался ли метод и т.п.)
- **Fake –** «легкая»/простая реализация аналогичная используемой в реальной программе

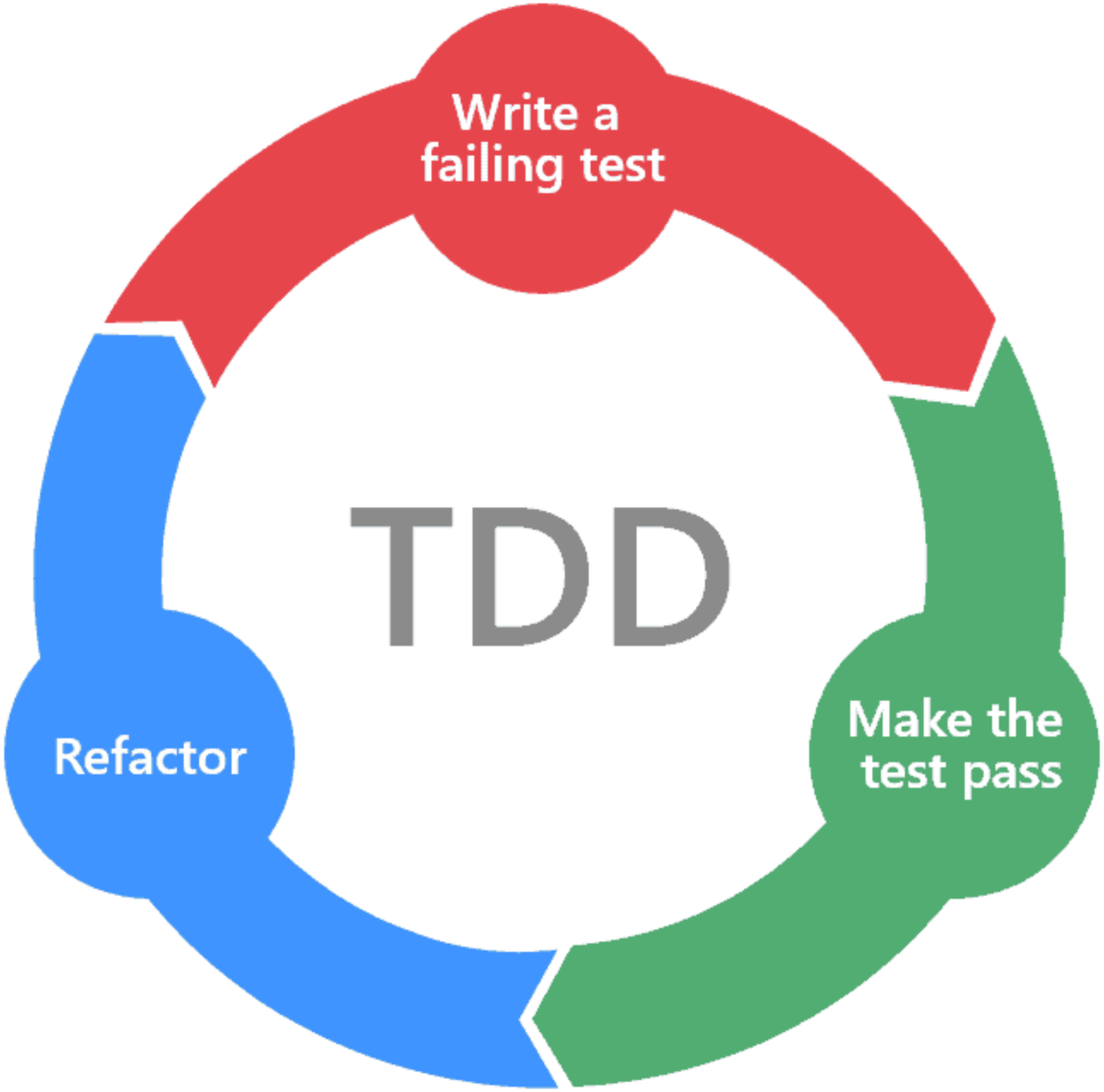
Тесты на состояние

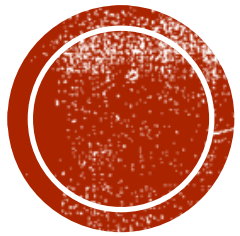


Тесты на поведение



- Имеет понятное название
- Простой
- Быстро работает
- Проверяет один класс эквивалентности/граничное значение
- Если падает – по нему очевидно где проблема
- Изолирован т.е. не зависит от окружения и других тестов, не влияет на другие тесты
- Тестирует намерения а не реализацию


Write a failing test
TDD
Make the test pass
Refactor



ТЕСТЫ НА C#

ТЕСТОВЫЕ ФРЕЙМВОРКИ

- nUnit, xUnit, msTest
- Каждый тест запускается отдельно
- Для создания моков, зачастую, используется библиотека **moq**

ПРИМЕР