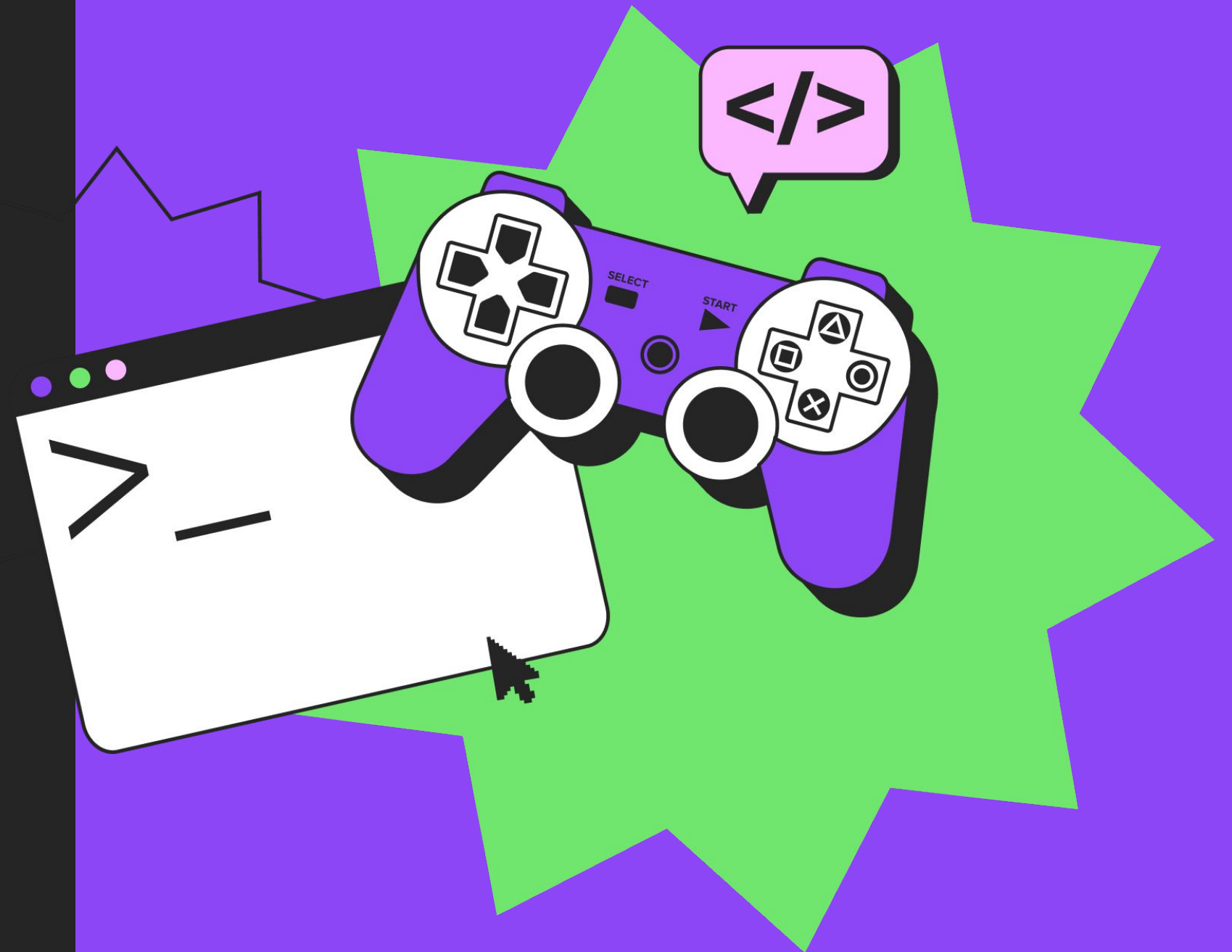


Разработка сетевого приложения на C#

Урок 6
Тестирование





План курса





Содержание урока

- Тестирование
- Модульное тестирование и фреймворки
- Мокирование
- TDD



Тестирование

Тестирование кода – это важная часть разработки программного обеспечения, которая позволяет обнаруживать и исправлять ошибки, убеждаться в правильной работе вашего кода и обеспечивать его надежность.





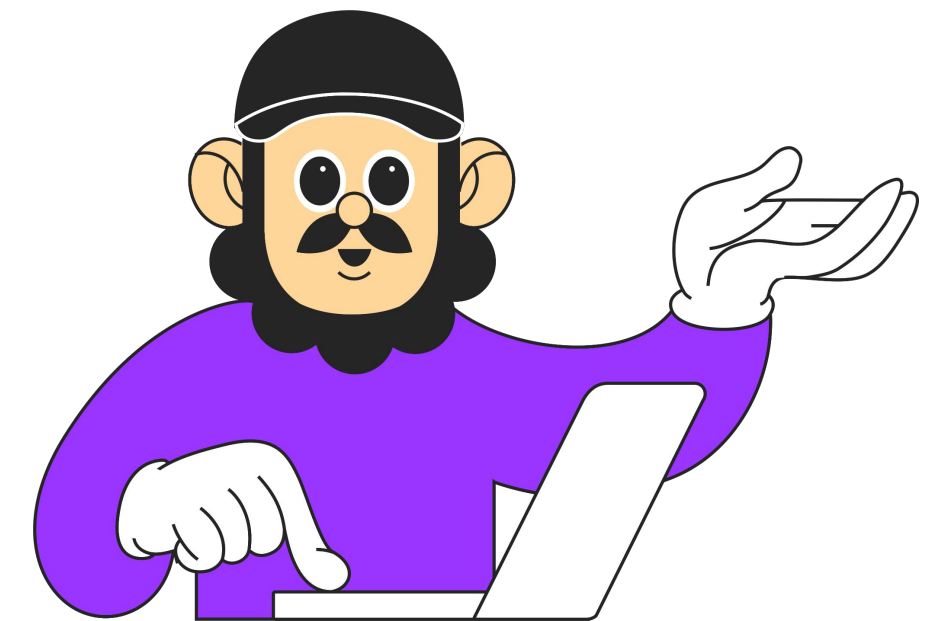
Виды тестирования

- Модульное тестирование (Unit Testing)
- Интеграционное тестирование (Integration Testing)
- Функциональное тестирование (Functional Testing)
- Приемочное тестирование (Acceptance Testing)
- Испытание на проникновение (Penetration Testing)
- Нагрузочное тестирование (Load Testing)
- Тестирование безопасности (Security Testing)
- Манипуляционное тестирование (Manipulation Testing)
- Автоматизированное тестирование (Automated Testing)



Модульное тестирование

Процесс тестирования отдельных модулей или компонентов программного обеспечения для обеспечения их корректности и соответствия ожиданиям.

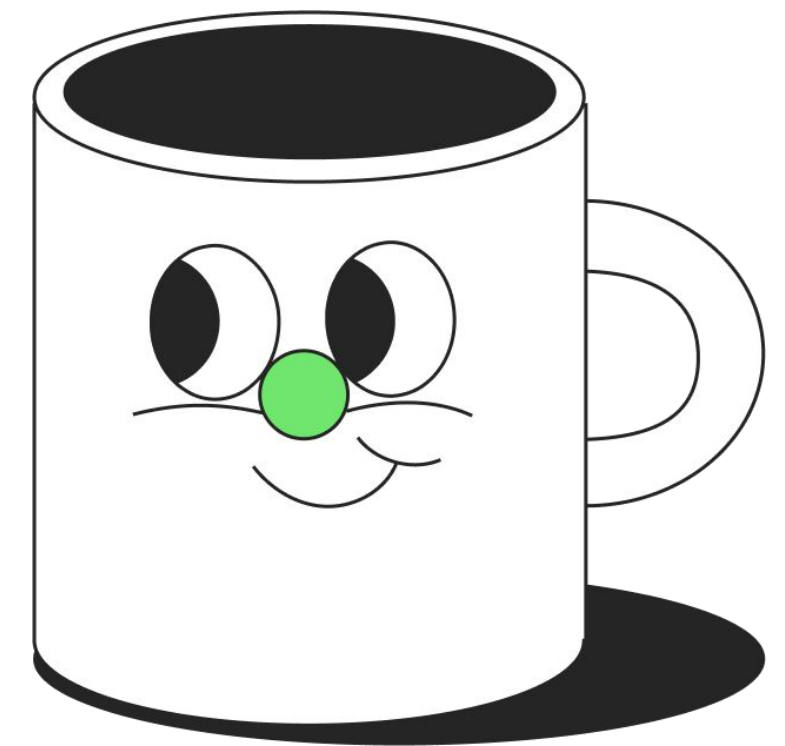




Модульное тестирование

Вот ключевые понятия и принципы модульного тестирования:

- Тестовый кейс (тест-кейс)
- Изоляция
- Автоматизация
- Ассерты (утверждения)
- Покрытие кода
- Регрессионное тестирование
- Фиктивные объекты (Моки)





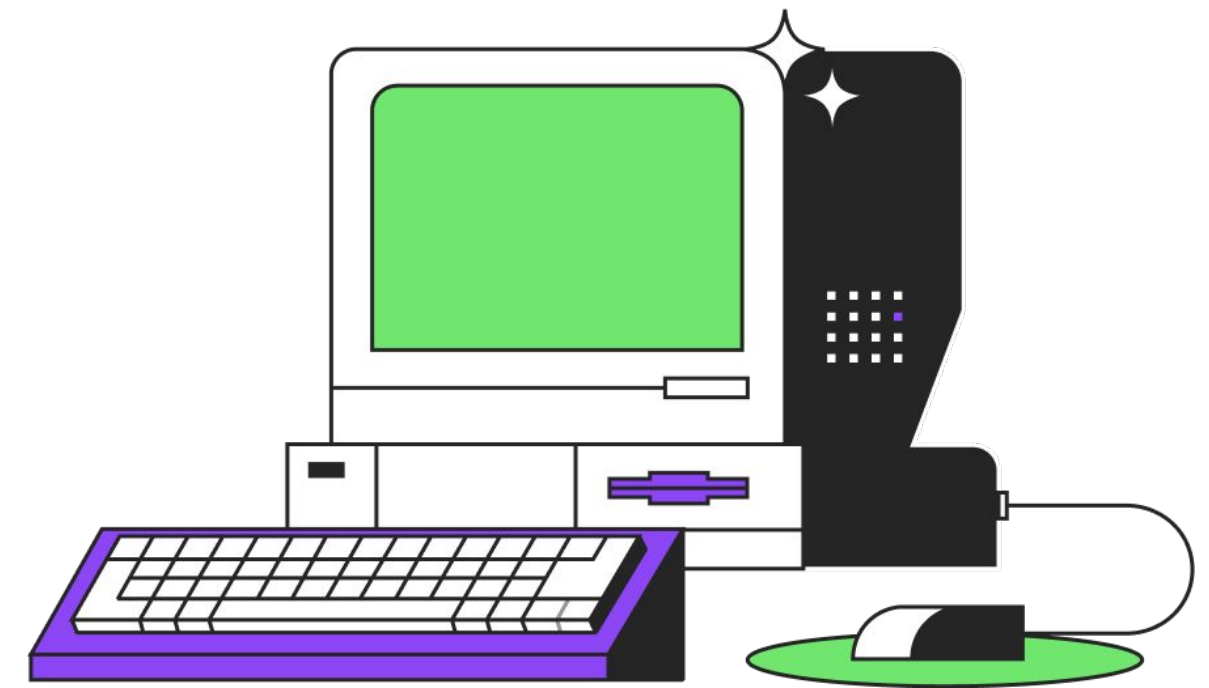
Модульное тестирование. Этапы

- Выбор модуля для тестирования
- Подготовка тестовых данных
- Написание тестовых случаев (тест-кейсов)
- Настройка окружения тестирования
- Выполнение тестовых случаев
- Проверка результатов (assertions)
- Завершение и очистка
- Отчеты и анализ результатов
- Исправление ошибок
- Повторение
- Покрытие кода
- Документация
- Интеграция в процесс разработки



Assert

Класс, предоставляющий методы для поддержки тестирования.





Мокирование

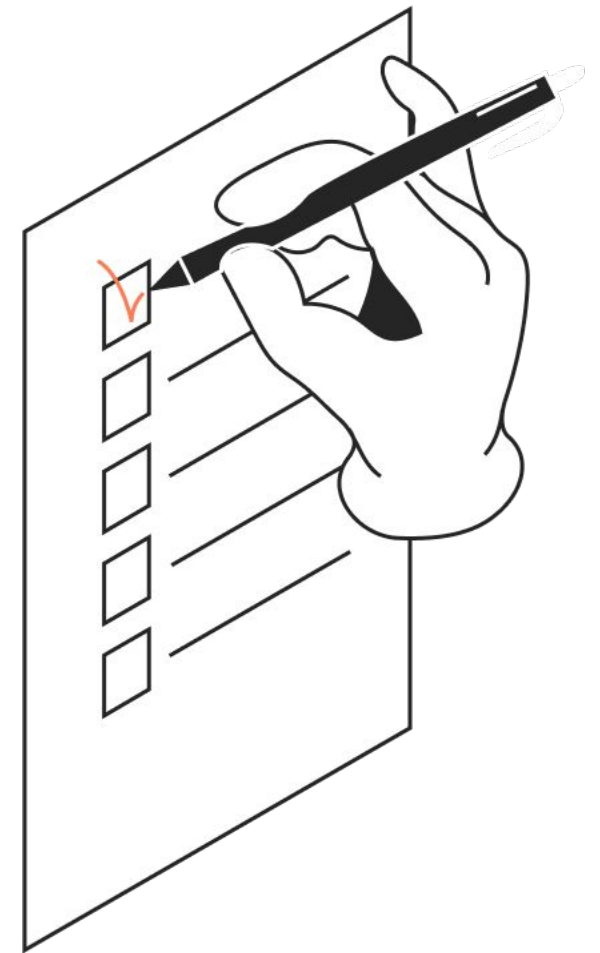
Мокирование позволяет заменить реальные объекты заглушками, отдающими тестовые данные.





Настройки тестов

Все три тестовых фреймворка, представленных в Visual Studio: MSTest, NUnit и XUnit, предоставляют возможность настройки.





TDD

Test-Driven Development (Разработка через тестирование) – это методология разработки программного обеспечения, в которой написание тестовых случаев (юнит-тестов) предшествует написанию кода приложения. Процесс TDD состоит из трех основных шагов: "Red-Green-Refactor" (Красный-Зеленый-Рефакторинг).

- Красный (Red)
- Зеленый (Green)
- Рефакторинг (Refactor)





TDD

Преимущества TDD:

- **Более высокое качество кода:** TDD обеспечивает более тщательное тестирование кода и позволяет обнаруживать ошибки на ранних этапах разработки, что способствует улучшению качества и надежности программы.
- **Проще вносить изменения:** Поскольку у вас уже есть комплект тестов, вы можете вносить изменения в код с большей уверенностью, зная, что тесты помогут обнаружить любые нарушения.
- **Документация:** Тесты служат в качестве документации к коду. Они описывают ожидаемое поведение компонентов.
- **Быстрые обратные связи:** TDD предоставляет быструю обратную связь о работе кода. Если тест не проходит, вы сразу узнаете об ошибке.
- **Спецификация перед реализацией:** TDD позволяет определить спецификацию для кода до его написания.



TDD

Основные недостатки TDD:

- **Дополнительное время:** Написание тестов может потребовать дополнительное время, особенно в начале проекта.
- **Сложность начала:** Некоторым разработчикам может быть сложно начать с написания тестов, особенно если они не имеют опыта TDD.
- **Постоянная поддержка тестов:** Тесты нужно поддерживать и обновлять вместе с изменениями кода.



Замер скорости работы алгоритмов

Для измерения скорости работы алгоритмов в C# вы можете использовать класс Stopwatch из пространства имен System.Diagnostics. Stopwatch предоставляет удобный способ измерения времени выполнения участка кода.






Подведение итогов

На этой лекции вы:

- Научились работать с тестами
- Узнали про Моск-объекты
- Поработали с 3 тестовыми фреймворками
- Научились измерять скорость работы алгоритмов (производительность)

Спасибо 
за внимание

