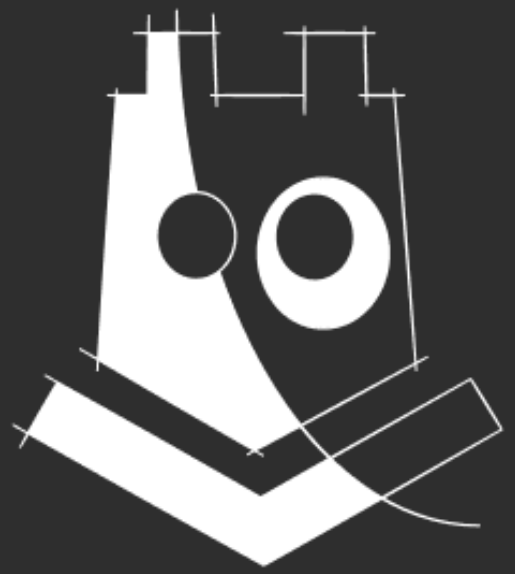
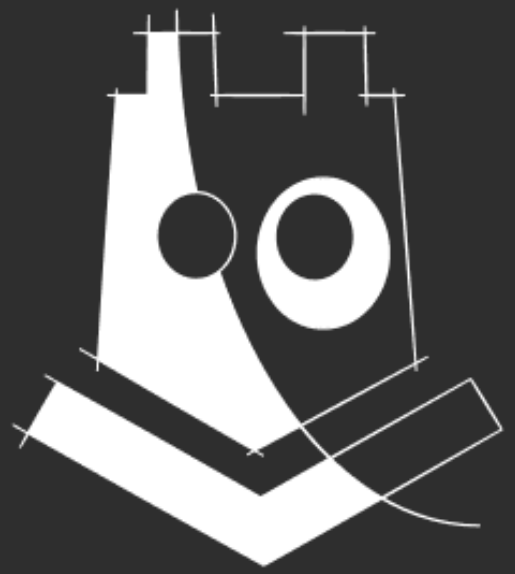


Unit testing: bugs strikes back

План семинара

- Unit testing: определение, назначение, принципы
- Простой unit test
- Testing frameworks
- Изоляция
- Обзор testing и mocking frameworks
- Тестопригодный код
- CI
- Вопросы



Unit testing: определение, назначение, принципы

Цель тестирования

Зачем тестируем?

Чтобы было качественно

Определение модуля

Что такое модуль?

Изолированная часть кода, выполняющая единицу работы

Определение модуля

- Конечный результат может принимать следующие формы
- Возвращенное значение из метода
- Видимое изменение состояния или поведения системы
- Обращение к сторонней системе

Unit тест

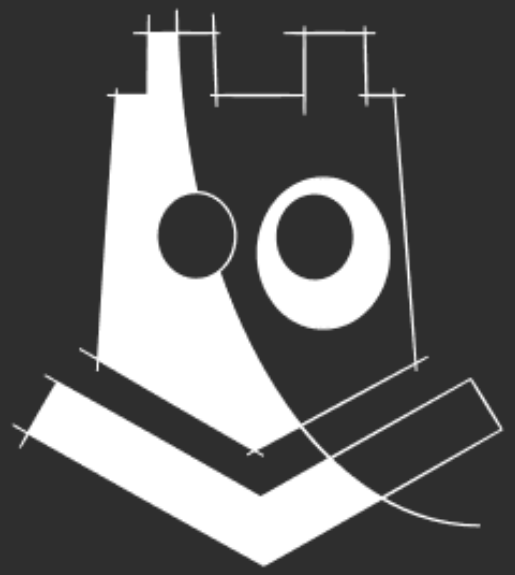
Код, который вызывает единицу работы и затем проверяет ее конечный результат.

Свойства хорошего unit теста

- Автоматизирован и повторяем
- Стабильный результат
- Сохраняет актуальность
- Прост в реализации
- Быстрый запуск
- Быстрая работа
- Полностью контролирует unit
- Полностью изолирован
- Понятная причина ошибки

Подготовка–действие–утверждение

- Рекомендуется следующая структура теста – AAA:
- Подготовка (arrange)
- Действие (act)
- Утверждение (assert)

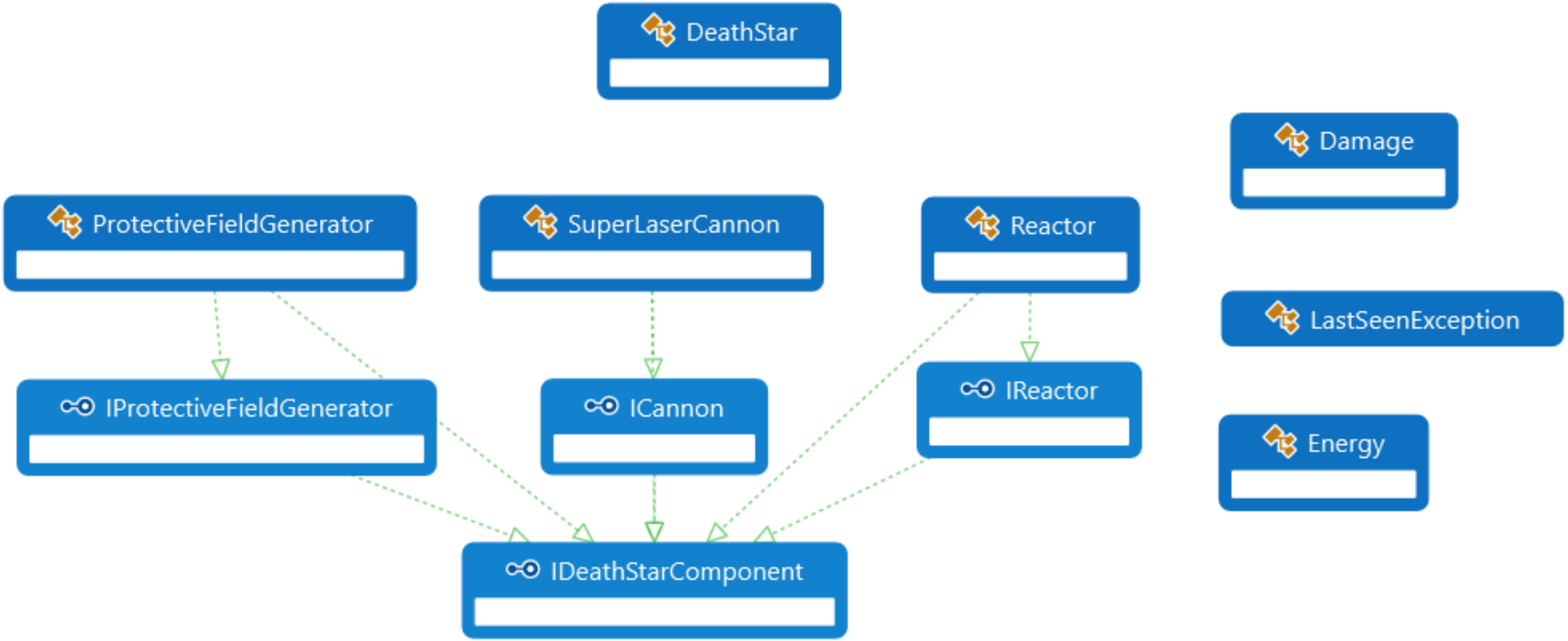


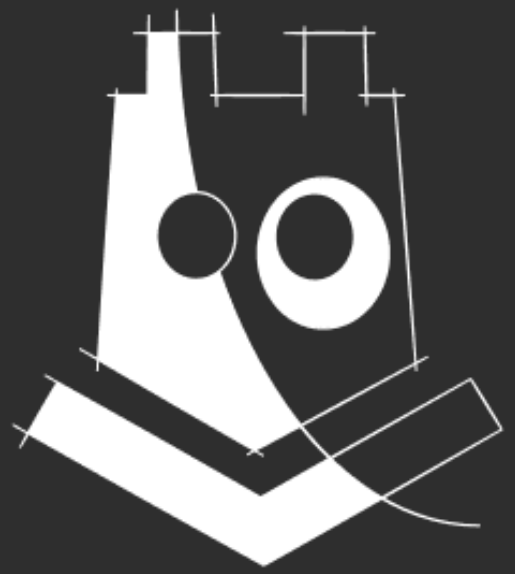
Эпизод I
Простой unit test

UNIT TESTS

Звезда Смерти была уничтожена. Но *Империя решила* восстановить его с *учетом всех* недостатков. Поэтому *жители планеты* Фриланс были выбраны *в качестве* исполнителей. Им были *переданы* чертежи старой Звезды *Смерти, Император* приказал сделать *хорошо, а для* контроля за исполнением *приставил* Дарта Вейдера…

Предметная область

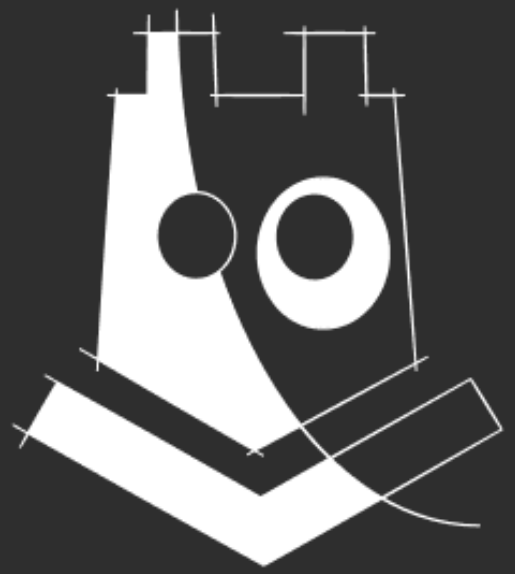




ДЕМО

Резюме

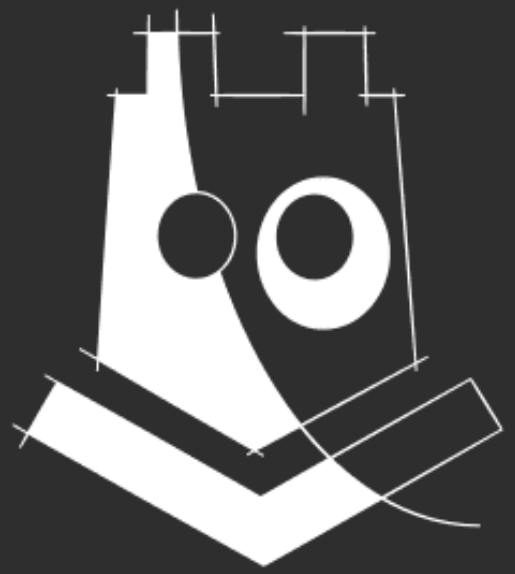
- Модуль – код, выполняющий единицу работы
- Unit тест проверяет модуль
- Знаем свойства хорошего unit теста
- Умеем делать простой unit тест



Эпизод II
Testing frameworks

Польза testing frameworks

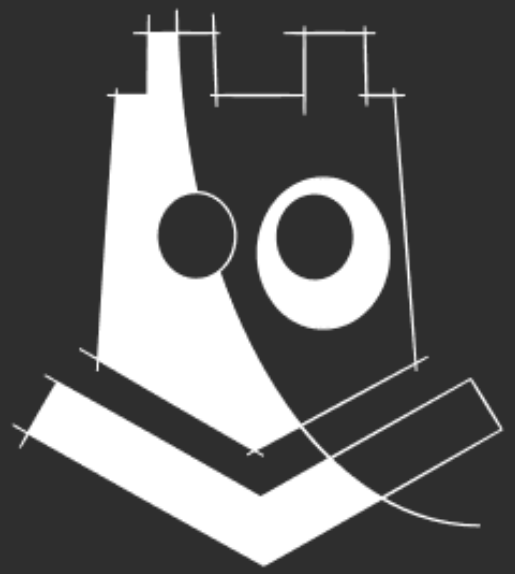
- Простота и упорядоченность написания тестов
- Выполнение одного или всех тестов
- Анализ результатов прогона тестов



ДЕМО

Резюме

- Можем написать простой тест
- Можем пропустить испорченный тест
- Можем делать различные предположения



Эпизод III
Изоляция

Зависимости

- Внутренние
- Внешние

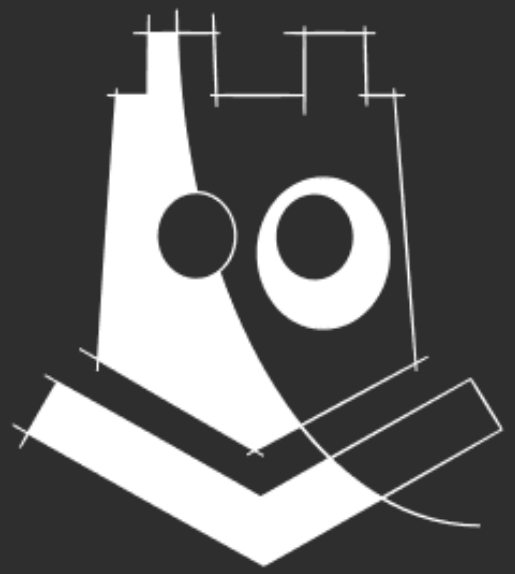
Разрыв зависимости

- Найти интерфейс
- Добавить абстракцию
- Заменить контролируемым объектом

Шов (seam)

Место программы, куда можно подключить иную функциональность взамен существующей

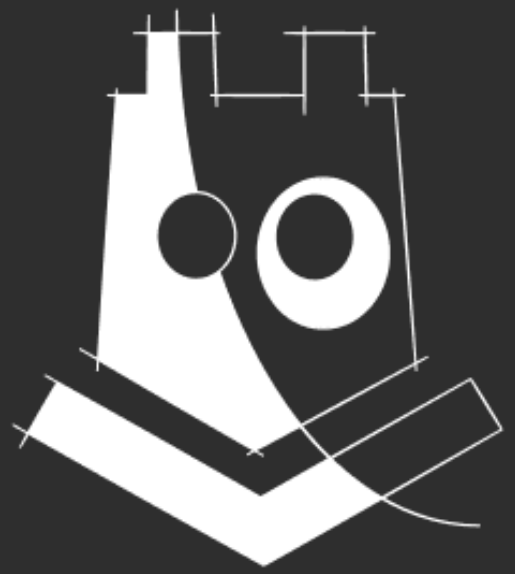
- через конструктор
- установить через свойство или метод
- получить непосредственного перед вызовом:
  - через параметр
  - с помощью фабрики
  - с помощью локального фабричного метода



ДЕМО

Поддельные объекты

Поддельный объект, подделка, fake –имитируют настоящий объект.



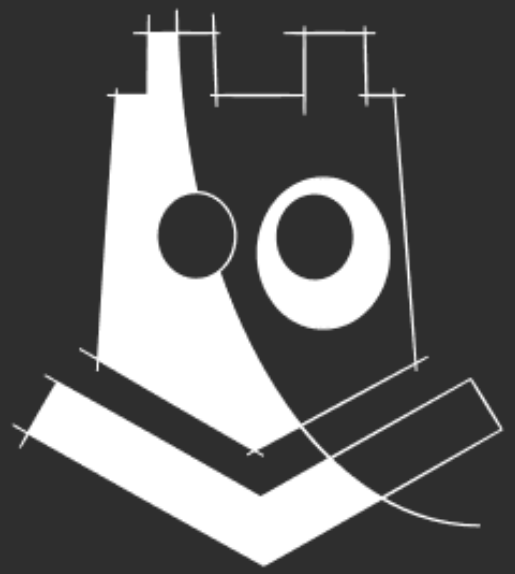
ДЕМО

Проблемы рукописных подделок

- Их написание требует времени
- Трудно писать подделки для интерфейсов и классов с большим число методов, свойств, событий
- Для сохранения состояния подставки требуется писать много стереотипного кода
- Сложно повторно использовать

Изолирующие фреймворки

- Упрощается проверка параметров
- Упрощается создание поддельных объектов



ДЕМО

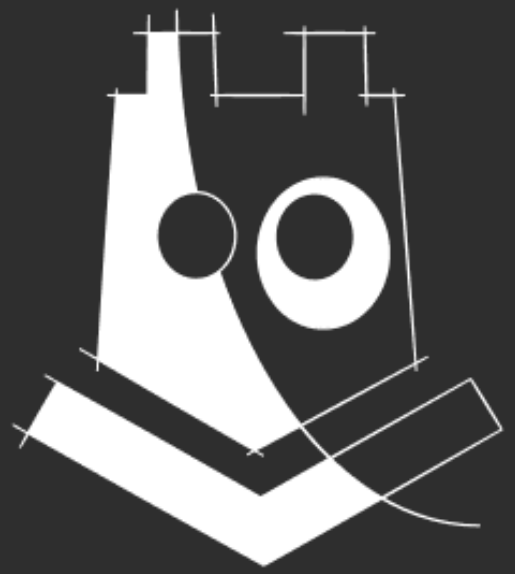
Классификация изолирующих фреймворков

- Ограниченные (не умеют подделывать статические методы, невиртуальные методы, sealed классы и т.д.)

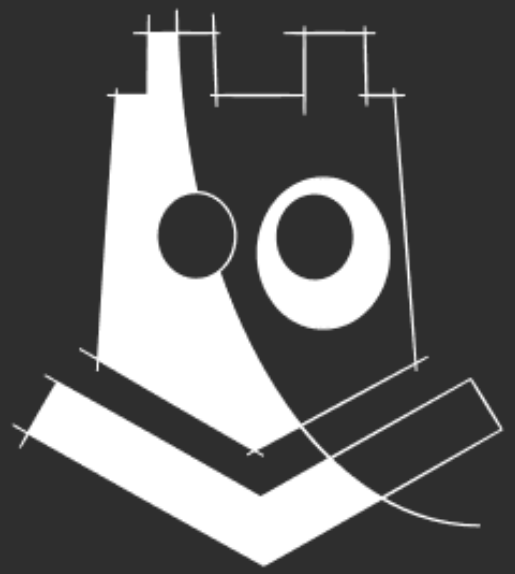
- Неограниченные (можно подделать все, что угодно)

Изоляция с помощью Fakes

- Добавить проект к тестовому проекту
- Создать fakes
- Использовать `ShimContext`

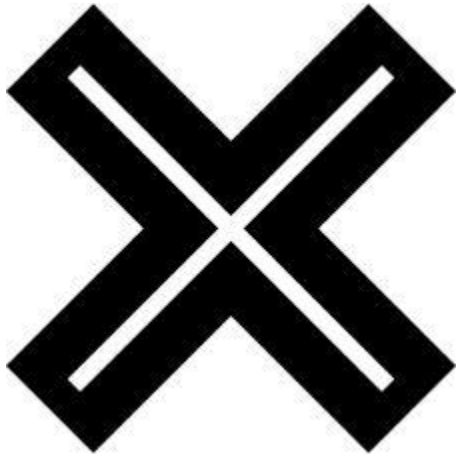


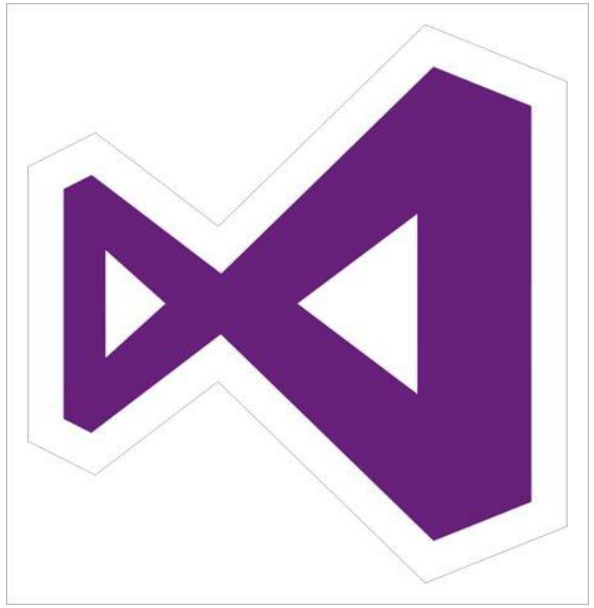
ДЕМО



Обзор testing и mocking frameworks

Обзор testing frameworks



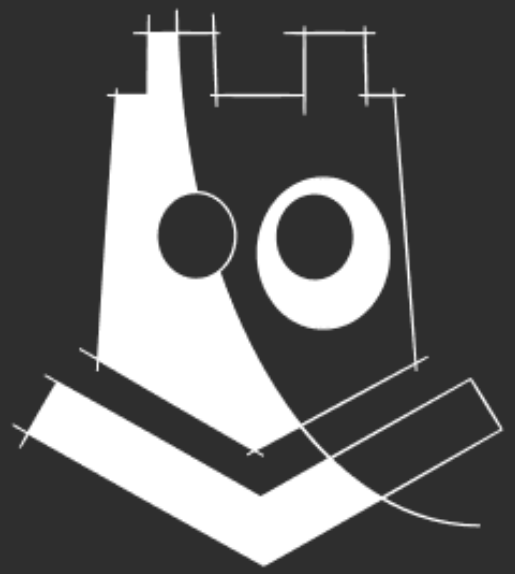


Обзор mocking frameworks

Критерий	Moq	Fakes (Moles)
Ограниченность	Ограниченный	Неограниченный
Создание подделок	+	+
Распространение	Nuget пакет	Интегрирована в VS Enterprise
Рекурсивные подделки	+	-
Массовое подделывание	+	-
Поддержка .Net Core	+	+/-

Резюме

- Можем сделать простой fake
- Можем сделать fake быстро с использованием Moq
- Подделаем все что угодно с MS Fakes
- Можем внедрять зависимости



Эпизод IV
Тестопригодный код

Рекомендации проектирования с учетом тестопригодности

- Делайте методы виртуальными
- Проектируйте на основе интерфейсов
- Делайте классы незапечатанными
- Избегайте создания экземпляров конкретных классов внутри методов с логикой

Рекомендации проектирования с учетом тестопригодности

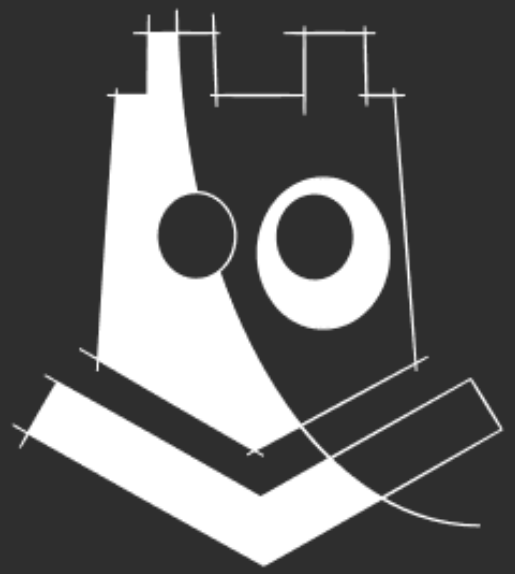
- Избегайте прямых обращений к статическим методам
- Избегайте конструкторов и статических конструкторов, содержащих логику
- Отделяйте логику синглтонов от логики их создания

Минусы проектирования с учетом тестопригодности

- Увеличивается объем работы
- Потенциальное усложнение кода
- Раскрытие внутренней реализации
- Иногда нет возможности

Плюсы проектирования с учетом тестопригодности

- Более простая проверка работоспособности кода
- Надежное исправления дефектов
- Следование принципам SOLID
- Тест – дополнительный пользователь API



Эпизод V
CI

CI

Непрерывная интеграция: Build, Test

- Поддержка тестов в актуальном состоянии
- Постоянная проверка кода
- Быстро обнаружения проблем при merge

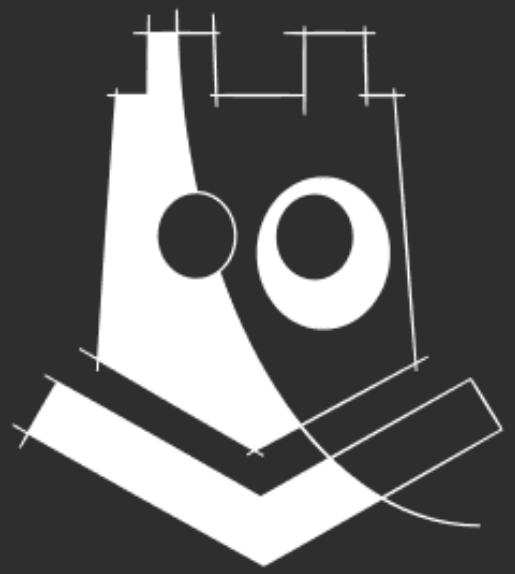
Интеграционный тест

Реальные зависимости вместо подделок

- Медленнее модульных тестов
- Требуют конкретного окружения
- Нужна подготовка

Резюме

- Убедились в дружбе CI и unit tests
- Настроили CI на github через Travis-CI
- Различаем интеграционные и unit тесты



Эпизод VI
Примеры из жизни

Безопасность vs тестопригодность

- internal и `[InternalsVisibleTo]`
- Атрибут `[Conditional]`
- Использование директив `#if` и `#endif` для условной компиляции

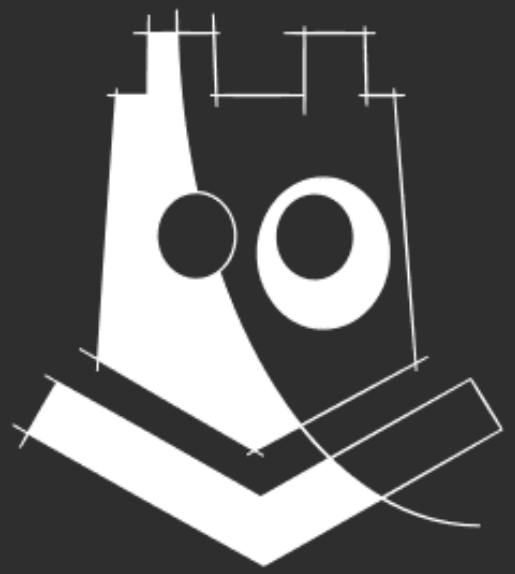
Важные моменты

- Придерживаться понятной иерархии

- Применяйте фабричные методы для повторного использования кода в тестах

Название теста

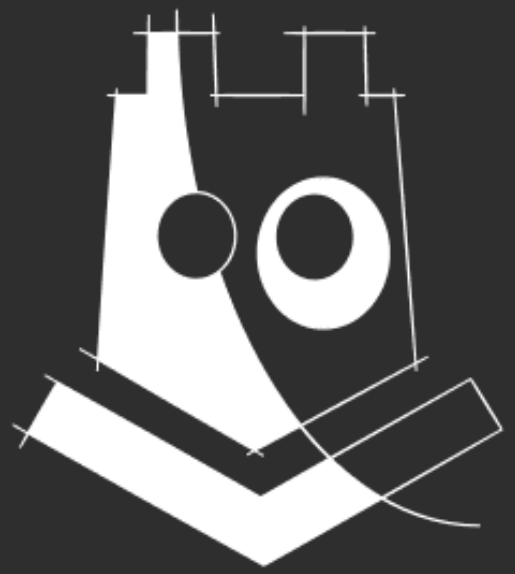
[единица работы]_[сценарий]_[ожидаемое поведение]



Заключение

Заключение

- Сейчас писать тесты значительно легче и быстрее
- Знаем, как написать хороший unit test
- Можно написать тестопригодный код, можно подделать
- Инструментов – много
- CI - помогает



Ссылки и литература

Ссылки и литература

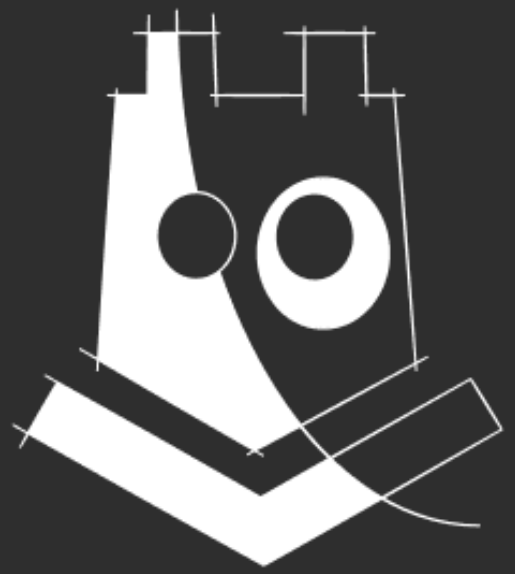
- Рой Ошероув – Искусство автономного тестирования
- Сравнение testing frameworks: https://dingyuliang.me/unit-testing-frameworks-xunit-vs-nunit-vs-mstest-net-net-core/
- xUnit: https://xunit.github.io/docs/comparisons
- Moq: https://github.com/Moq/moq4/wiki/Quickstart

Ссылки и литература

- Создание своих атрибутов для xUnit (чтения данных из файла): https://andrewlock.net/creating-a-custom-xunit-theory-test-dataattribute-to-load-data-from-json-files/

- Список ништяков для тестирования на .Net: https://github.com/dariusz-wozniak/List-of-Testing-Tools-and-Frameworks-for-.NET

- Репозиторий с кодом : https://github.com/Markeli/BugsStrikesBack



Спасибо за внимание!
Вопросы?