НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Факультет компьютерных наук
Департамент программной инженерии

ОБЕСПЕЧЕНИЕ КАЧЕСТВА И ТЕСТИРОВАНИЕ

Семинар 3: Тестирование. Модульное тестирование (xUnit)

Москва, 2020

ДИНАМИЧЕСКАЯ ВЕРИФИКАЦИЯ

1. Методы использующие результаты реальной работы проверяемой программной системы или ее прототипов, чтобы проверять соответствие этих результатов требованиям и проектным решениям.

РЕАЛЬНАЯ И ИМИТАЦИОННАЯ ДИНАМИЧЕСКАЯ ВЕРИФИКАЦИЯ

1. **Реальная верификация** - в ходе верификации используется само ПО

2. **Имитационная верификация** - в ходе верификации используется прототип или исполнимая модель

ТЕСТИРОВАНИЕ И МОНИТОРИНГ

1. **Мониторинг** - наблюдение, запись и оценка результатов работы ПО при его обычном использовании
2. **Тестирование** - проверяемое ПО выполняется в рамках заранее подготовленных сценариев

ДИНАМИЧЕСКАЯ ВЕРИФИКАЦИЯ И ОТЛАДКА (DEBUGGING)

1. **Динамическая верификация** - обнаружение наличия ошибок и оценка качества ПО.
2. **Отладка (debugging)** - определение точного местоположения и исправление ошибок.

ТЕСТИРОВАНИЕ

1. **Тестирование** (testing) является методом верификации, в рамках которого результаты работы тестируемой системы или компонента в ситуациях из выделенного конечного набора проверяются на соответствие проектным решениям, требованиям, общим задачам проекта, в рамках которого эта система разрабатывается или сопровождается.

ВИДЫ КЛАССИФИКАЦИЙ ТЕСТИРОВАНИЯ

1. По объекту тестирования
2. По степени автоматизации
3. По степени изолированности
4. По времени проведения тестирования
5. По знанию внутреннего строения системы

КЛАССИФИКАЦИЯ ПО ЗНАНИЮ ВНУТРЕННЕГО СТРОЕНИЯ СИСТЕМЫ

1. Тестирование черного ящика
2. Тестирование белого ящика
3. Тестирование серого ящика

КЛАССИФИКАЦИЯ ПО ЗНАНИЮ ВНУТРЕННЕГО СТРОЕНИЯ СИСТЕМЫ

1. Тестирование черного ящика
2. Тестирование белого ящика
3. Тестирование серого ящика

КЛАССИФИКАЦИЯ ПО СТЕПЕНИ ИЗОЛИРОВАННОСТИ

- Приемочное тестирование
- Системное тестирование
- Интеграционное тестирование
- Модульное тестирование (тестирование компонентов)

ПРИЕМОЧНОЕ ТЕСТИРОВАНИЕ

- **Приемочное тестирование** – это комплексное тестирование, необходимое для определения уровня готовности системы к последующей эксплуатации. Тестирование проводится на основании набора тестовых сценариев, покрывающих основные бизнес-операции системы.

СИСТЕМНОЕ ТЕСТИРОВАНИЕ

- **Системное тестирование** – это тестирование программного обеспечения выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям, как функциональным, так и не функциональным.

ИНТЕГРАЦИОННОЕ ТЕСТИРОВАНИЕ

- **Интеграционное тестирование** – вид тестирования, при котором на соответствие требований проверяется интеграция модулей, их взаимодействие между собой, а также интеграция подсистем в одну общую систему.

МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

- **Модульное тестирование** — тестирование каждой атомарной функциональности приложения отдельно, в искусственно созданной среде.
- **Цель модульного тестирования** — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

1. Ранее обнаружение багов
2. Предотвращение регрессионных багов
3. Рефакторинг без страха
4. Грамотная декомпозиция кода
5. Документация

МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

1. Когда модульное тестирование НЕ работает

- Сложный код
- Результат известен лишь приблизительно
- Ошибки интеграции и производительности
- При общей низкой культуре программирования
- Проблемы с объектами-заглушками

ФАЗЫ МОДУЛЬНОГО ТЕСТИРОВАНИЯ

1. Стандарт IEEE 1008 описывает процедуру подготовки модульных тестов, их выполнения и оценки результатов, состоящую из следующих видов деятельности.
2. Фаза планирования тестирования
3. Фаза получения набора тестов
4. Фаза измерений тестируемого модуля

ФАЗА ПЛАНИРОВАНИЯ ТЕСТИРОВАНИЯ

- **Этап планирования** основных подходов к тестированию, ресурсное планирование и календарное планирование
  - определение общего подхода к тестированию модулей
  - определение требований к полноте тестирования
  - определение требований к завершению тестирования
  - определение требований к ресурсам
  - определение общего плана-графика работ
- **Этап определения свойств**, подлежащих тестированию
  - изучение функциональных требований
  - определение дополнительных требований и связанных процедур
  - определение состояний тестируемого модуля
  - определение характеристик входных и выходных данных
  - выбор элементов, подвергаемых тестированию
- **Этап уточнения основного плана**, сформированного на этапе (1)

ФАЗА ПОЛУЧЕНИЯ НАБОРА ТЕСТОВ

- **Этап разработки** набора тестов
  - разработка архитектуры тестового набора
  - разработка явных тестовых процедур (тест-требований)
  - разработка тестовых примеров
  - разработка тестовых примеров, основанных на архитектуре (в случае необходимости)
  - составление спецификации тестовых примеров
- **Этап реализации** уточнённого плана

ФАЗА ИЗМЕРЕНИЙ ТЕСТИРУЕМОГО МОДУЛЯ

- **Этап выполнения** тестовых процедур
- **Этап определения достаточности** тестирования
- **Этап оценки результатов** тестирования и тестируемого модуля.

- Сбору подлежит следующая информация:
  - результат выполнения каждого тестового примера (прошел/не прошел);
  - информация об информационном окружении системы в случае, если тест не прошел;
  - информация о ресурсах, которые потребовались для выполнения тестового примера.

ИНСТРУМЕНТЫ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ

- Инструменты управления информацией о тестах (test management tools)
- Инструменты мониторинга
- Инструменты сбора данных о тестовом покрытии (test coverage tools)
- Каркасы выполнения тестов (test execution frameworks).
- Инструменты генерации тестовых данных (test input generators).
- Инструменты доступа к специализированным интерфейсам
  - Инструменты тестирования пользовательского интерфейса
  - Специализированные инструменты тестирования протоколов
- Инструменты автоматизации построения тестов на основе моделей

UNIT-ТЕСТ

- **Unit-тест** метод, написанный на языке программирования. Данный метод вызывает тестируемый метод с некоторыми входными параметрами и сравнивает ожидаемое значение с полученным. Если полученное значение соответствует ожидаемому, то данный модульный тест считается пройденным.

TEST EXECUTION FRAMEWORKS

- Тесты запускаются как исполнимые модули или оформляются как программы, использующие API фреймворка для мониторинга их работы.
- Предоставляются возможности по оценке того, выполнился тест успешно или нет, и дополнительные библиотеки для организации проверок в тестах и сброса трассировочной информации.

СЕМЕЙСТВО ФРЕЙМВОРКОВ XUNIT

- xUnit - это собирательное название семейства фреймворков для модульного тестирования, структура и функциональность которых основана на SUnit, предназначавшегося для языка программирования Smalltalk.

СЕМЕЙСТВО ФРЕЙМВОРКОВ XUNIT

- CppUnit - фреймворк для C++
- DUnit - инструмент для среды разработки Delphi
- JUnit - библиотека для Java
- NUnit, xUnit.NET - среда юнит-тестирования для .NET
- Unittest – для Python
- phpUnit - библиотека для PHP
- И другие - List of unit testing frameworks

SETUP-EXERCISE-VERIFY-TEARDOWN

setup	инициализация системы и ресурсов
exercise	выполнение нужного тестового воздействия
verify	проверка корректности результата
teardown	освобождение ресурсов и, возможно, возвращение в исходное состояние

ARANGE-ACT-ASSERT

- Обычно unit-тест включает три действия (также именуемые AAA):

Arange	задать исходное состояние объектов
Act	выполнить действия (взаимодействия)
Assert	проверить результат выполненных операций на соответствие ожидаемым значениям

- (Операция Assert может выполняться несколько раз в одном unit-тесте. Обычно в операции Assert проверяют тип данных, значение переменной, нахождение значения переменной в требуемом диапазоне значений и др.)

МОДУЛЬНОЕ ТЕСТИРОВАНИЯ ПО НА ЯЗЫКЕ JAVA

- **JUnit** - фреймворк для модульного тестирования программного обеспечения на языке Java, принадлежит семье фреймворков xUnit для разных языков программирования
- **TestNG** - фреймворк для модульного тестирования программного обеспечения на языке Java, вдохновленная JUnit и NUnit.

МЕТОДЫ ASSERT (СЕМЕЙСТВО XUNIT)

Equal	Проверяет равенство
NotEqual	Проверяет неравенство
InRange	Проверяет факт присутствия значения в диапазоне
NotInRange	Проверяет факт отсутствия значения в диапазоне
IsType	Проверяет тип значения
IsNotType	Проверяет тип значения
Null	Проверяет ссылку на равенство значению null
NotNull	Проверяет, что ссылка не равна значению null
True	Проверяет логическое значение на равенство значению true
False	Проверяет логическое значение на равенство значению false
Throws	Проверяет факт появления исключительной ситуации определенного типа
DoesNotThrow	Проверяет факт не появления исключительной ситуации определенного типа

АННОТАЦИИ

- В языке Java специальная форма синтаксических метаданных, которая может быть добавлена в исходный код.
- Аннотации используются для анализа кода, компиляции или выполнения.

  Аннотируемы пакеты, классы, методы, переменные и параметры.

ПРИМЕРЫ АННОТАЦИЙ В XUNIT

Аннотация	Описание
@Test public void method()	Аннотация **@Test** определяет что метод method() является тестовым.
@Before public void method()	Аннотация **@Before указывает** на то, что метод будет выполнятся перед каждым тестируемым методом @Test.
@After public void method()	Аннотация **@After указывает** на то что метод будет выполнятся после каждого тестируемого метода @Test
@BeforeClass public static void method()	Аннотация **@BeforeClass** указывает на то, что метод будет выполнятся в начале всех тестов, а точней в момент запуска тестов(перед всеми тестами @Test).
@AfterClass public static void method()	Аннотация **@AfterClass** указывает на то, что метод будет выполнятся после всех тестов.
@Ignore	Аннотация **@Ignore** говорит, что метод будет проигнорирован в момент проведения тестирования.
@Test (**expected** = Exception.class)	(**expected** = Exception.class) — указывает на то, что в данном тестовом методе вы преднамеренно ожидается Exception.
@Test (**timeout**=100)	(**timeout**=100) — указывает, что тестируемый метод не должен занимать больше чем 100 миллисекунд.

ЗАГЛУШКИ

- **Заглушки(агл. stub)** — В объектно-ориентированном программировании mock-объект имитирует поведение реального объекта заданным образом.
- Во время unit-тестирования mock-объекты могут симулировать поведение бизнес-объектов и бизнес-логику, что иногда необходимо из-за сложности реального поведения

ЛИТЕРАТУРА

1. Кент Бек – «Экстремальное программирование. Разработка через тестирование»
2. Gerard Meszaros – «xUnit Test Patterns»
3. JUnit – https://junit.org/
4. TestNG – https://testng.org
5. JaCoCo – https://www.jacoco.org
6. Цикломатическая сложность – https://ru.wikipedia.org/wiki/Цикломатическая_сложность

СПАСИБО! ВОПРОСЫ?

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ