



# Тестирование ОО программ

Сергей Андреев, JetBrains

18 апреля 2012

# Некоторый план

Есть система, есть подсистемы, есть отдельные классы и методы.

Есть взаимодействия между подсистемами.

А теперь в обратном порядке:

тестирование методов

тестирование классов

тестирование подсистем

интеграционное тестирование

системное тестирование

...

регрессионное тестирование

**Я не буду учить вас программировать! Я сам не умею...**

# ООП

инкапсуляция;

наследование;

полиморфизм;

интерфейс;

класс;

объект;

сообщение;

# Какие проблемы могут ВОЗНИКНУТЬ

## Инкапсуляция:

- Объект скрывает информацию. Это приводит к тому, что внесенные в объект изменения трудно заметить, в силу чего становится труднее выполнять анализ результатов тестирования.

- Объект может перейти в состояние, в котором затем удерживается на протяжении всего своего жизненного цикла. Это состояние может далее стать неадекватным и служить причиной некорректного поведения.

## Наследование:

- Каждый нижний уровень наследования создаёт новый контекст для наследованных фич, корректное поведение на верхнем уровне не гарантирует корректное поведение на нижних уровнях.

- Наследование от нескольких классов. Класс наследует от 2-х классов в которых могут присутствовать функции с одинаковыми именами.

- Динамическое связывание и сложные структуры наследования создают множество возможностей для ошибок вследствие непредвиденных связываний и неправильного толкования корректного использования
- Интерфейсные ошибки. ОО программы обычно содержат множество маленьких компонентов и как следствие много интерфейсов. Интерфейсные ошибки имеют большую вероятность возникновения.
- Объект обладает жизненным циклом. Такой объект может подвергнуться анализу в любой момент на протяжении ЖЦ с целью определить, находится ли он в состоянии, соответствующем ЖЦ. Запоздалое построение объекта или его преждевременное уничтожение - источник проблемы.
- Объекты сохраняют состояния, но их управление (в виде принимаемых последовательностей событий) обычно распределены по всему приложению. Ошибки управления состояниями..

# Тестирование методов

Ошибки в реализации методов:

В общем случае:

- сообщение отослано объекту у которого нет соответствующего метода
- недостижимый код
- нарушение контракта (предусловия, постусловия, инварианты)
- сообщение послано не тому серверному объекту
- параметры сообщения некорректны или отсутствуют, что приводит к неправильному или неудавшемуся связыванию
- некорректный приоритет сообщения
- сообщение не реализовано на сервере
- формальные и действительные параметры сообщения противоречат
- синтаксическая ошибка

Алгоритм:

- не эффективный, слишком медленный, потребляющий слишком много памяти
- не правильный вывод
- не правильная точность, чрезмерная разрядность или ошибка округления
- объект не сохранён или сохранён не верный объект
- не завершается

- Исключения: (Exceptions):

- отсутствуют
- не правильные
- никем не обрабатываемые (не пойманные)
- не правильный catch
- распространяется out of scope
- Exception not raised
- не корректное состояние после исключения
- отсутствующий объект (указанный, но не определённый)

- Переменные определение/использование

- неиспользованные объекты (определены, но нет ссылок на них)
- ссылки на неопределенные или удаленный объекты
- не корректное использование дружественными функциями
- отсутствующая инициализация
- некорректное приведение типов
- нарушение контракта
- некорректная видимость
- некорректная сериализация, приводящая в “испорченное” состояние
- недостаточная точность, диапазон значений типа

# Category Partition

Не буду переводить... (Map reduce, ведь, не переводят)

Этот подход предполагает, что ошибки зависят от комбинаций параметров сообщений и переменных конкретного инстанса и что эти ошибки приведут к отсутствию результата или некорректному результату.

Ошибки, которые проявляются при определенной последовательности или портят переменные инстанса скрытые за интерфейсом тестируемого метода могут не быть обнаруженными таким способом.



Разберем на примере функции `List::getNextElement()`

Умеет кидать 2 Exception'a:

`NoPosition` - если позиция не была определена предыдущей операцией или если она больше не существует из-за к.л. промежуточного изменения или удаления

`EmptyList` - если список пустой

# 7 шагов к успеху

1. Определяем пригодные для тестирования функции метода
2. Определяем вход и выход каждой функции
3. Определяем категории для каждого входного параметра
4. Для каждой категории создаем тест-кейсы
5. Определяем ограничения для наборов
6. Генерируем тесты перебирая все комбинации наборов
7. Разрабатываем ожидаемый результат для каждого тест-кейса используя соответствующего оракула.

# Классы эквивалентности

Классы эквивалентности (Эквивалентное разбиение) – красивое название простой сути. Вы определяете входные выходные значения, поведение среды или любые другие факторы, которые вам интересны при тестировании, группируете эти факторы в классы, которые система должна считать эквивалентными.

Попробуем?



# Тестовые сценарии эквиваленты, если:

- они тестируют одно и то же;
- если один из них выявляет ошибку, то и остальные выявят ее;
- если одни из них не выявляет ошибку, то и остальные не выявят.

Результат выполнения всех тестов в одном классе эквивалентности «отвечает на один и тот же вопрос».

# Граничные условия

- это ситуации перед границей, на границей и за границей входных КЭ и выходных КЭ.

Опыт показывает, что тест кейсы, которые исследуют граничные условия имеют большую отдачу

# Анализ ГУ

**Анализ граничных значений отличается от эквивалентного разбиения следующим:**

- Выбор любого элемента в классе эквивалентности в качестве представительного осуществляется таким образом, чтобы проверить тестом каждую границу этого класса.
- При разработке тестов рассматриваются не только входные значения (пространство входов), но и выходные (пространство выходов).



# Правила:

1. Если входные данные представляют диапазон значений, то необходимо построить тесты с неправильными входными данными для ситуации незначительного выхода за границы области значений.
2. Также обязательно писать тесты для минимальной и максимальной границы диапазона.
3. Использовать первые два правила для каждого из входных значений (использовать пункт 2 для всех выходных значений).
4. Если вход и выход программы представляет упорядоченное множество, сосредоточить внимание на первом и последнем элементе

# 7 шагов к успеху

1. Определяем пригодные для тестирования функции метода
2. Определяем вход и выход каждой функции
3. Определяем категории для каждого входного параметра
4. Для каждой категории создаем тест-кейсы
5. Определяем ограничения для наборов
6. Генерируем тесты перебирая все комбинации наборов
7. Разрабатываем ожидаемый результат для каждого тест-кейса используя соответствующего оракула.

Пожалуй хватит.  
Спасибо за внимание!

[sergey.andreev@jetbrains.com](mailto:sergey.andreev@jetbrains.com)  
[smandreev@gmail.com](mailto:smandreev@gmail.com)