



Разработка через тестирование Legacy code

Test Driven Development

Ivan Dyachenko <IDyachenko@luxoft.com>

Для кого этот тренинг?

1

Beginner

Хорошая точка входа

2

Intermediate

Поможет лучше всё структурировать в голове и
объяснять коллегам

3

Advanced

Можно использовать для обучения и проверки
других

Содержание

1

2

3

4

5

6

7

Причины появления legacy-код(а):

- Появление новых технологий
- Развитие существующих технологий
- Бурное развитие системы

Появление новых технологий



Существуют огромные программные комплексы, написанные на языке C++, который считался передовым всего десять лет назад

Развитие существующих технологий



Частые изменения в языках и платформах Java/С# очень быстро приводят к появлению унаследованных решений, поскольку то, что когда-то считалось новаторским, превращается в неподдерживаемый, устаревший код

Бурное развитие системы



Зачастую, программисты сосредотачивают усилия именно на реализации новых возможностей программного обеспечения – применяются новые технологии, но существующий функционал при этом не совершенствуется

Legacy-код



Итак, нужно ли писать тесты для legacy-кода?

Легасу-код



Да, нужно!

- Unit-тесты позволят вам понять, как работает код
- Unit-тесты позволят вам убедиться, что ваши изменения не нарушили логику работы модуля

- Ваша цель – получить тесты, которым можно доверять
- Если функция полностью покрыта модульными тестами, то вы не будете бояться ее изменить или даже полностью переписать

Избавление от зависимостей

- Как правило, такой код имеет множество зависимостей: подключает различные lib и dll, посылает сообщения в сеть или другим компонентам, отображает что-то в GUI и т.д.
- Итак, первое, что нам необходимо сделать – это избавиться от этих зависимостей
- Для этого можно использовать уже знакомые нам mock- и stub-объекты

Dependency Injection (DI)

- Зачастую, этих средств недостаточно и необходимо проводить рефакторинг
- Здесь на помощь нам приходит паттерн “Dependency injection”
- Суть его заключается в следующем: если класс А использует класс В, то необходимо сделать так, чтобы конкретная реализация класса В передавалась классу А “извне”, а не определялась внутри него

Dependency Injection (DI)



- Пример ...

Dependency Injection (DI)

- Несмотря на то, что данный класс использует интерфейс, смысла в этом нет, поскольку конкретная реализация задается в этом же классе
- Модифицируем код этого класса для того, чтобы он получал реализацию интерфейса “извне”
- Для этого достаточно немного изменить конструктор

Проблемы покрытия legacy-кода

- После того, как все связи с внешним миром у класса или функции оборваны - мы ее полностью контролируем и можем полностью проверить ее работу
- Для этого может понадобиться создать еще несколько заглушек, которые будут выдавать нужные данные для конкретных тестов

Проблемы покрытия legacy-кода

- В итоге получается, что надо написать достаточно много дополнительного кода только для того, чтобы написать первый тест
- Однако для второго теста, такого кода потребуется уже меньше
- «Десятый» тест уже пишется спокойно, с использованием ранее написанного кода



Вопросы ?



Разработка через тестирование

IDyachenko@luxoft.com

```
git clone git://github.com/ivan-dyachenko/Trainings.git
```

```
https://github.com/ivan-dyachenko/Trainings
```