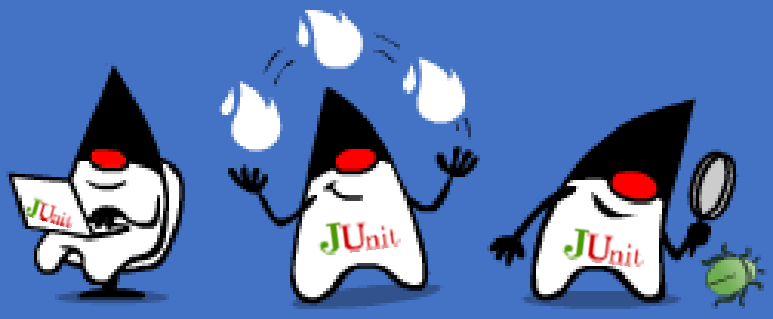
Laboratory Work #26

Test Driven Development. Unit-testing in Java with jUnit and TestNG



LEARN. GROW. SUCCEED.

® 2020-2021. Department: <Software of Information Systems and Technologies>
Faculty of Information Technology and Robotics
Belarusian National Technical University
by Viktor Ivanchenko / ivanvikvik@bntu.by / Minsk

ЛАБОРАТОРНАЯ РАБОТА #26

Методология разработки Test Driven Development. Использование тесто-вых фреймворков jUnit и TestNG для проведения модульного тестирования

Цель работы 

Освоить методологию разработки ПО через тестирование (TDD), а также изучить соответствующие тестовые фреймворки для модульного тестирования jUnit и TestNG и закрепить их практически.

Требования 

1) Необходимо скорректировать UML-диаграмму взаимодействия классов и объектов программной системы с учётом дополнения тестового проекта.
2) При корректировки программной системы необходимо полностью использовать своё ООП-воображение и по максимум использовать возможности, которые предоставляет язык Java для программирования с использованием методологии ООП.
3) Каждый пользовательский тип должен иметь адекватное осмысленное имя и информативный состав (соответствующие конструкторы: по умолчанию, с параметрами, конструктор-копирования; ***get***- и ***set***-методы для доступа к состоянию объекта; корректно переопределённые методы базового класса *Object*: ***toString()***, ***equals()***, ***hashCode()*** и др.).
4) Также рекомендуется придерживаться ***Single Responsibility Principle***, ***SRP*** (принципа единственной ответственности): у каждого пакета, класса или метода должна быть только одна ответственность (цель), т.е. должна быть только одна причина изменить в дальнейшем соответствующий блок кода.

5) Добавляемые классы необходимо грамотно разложить по соответствующим пакетам, которые должны иметь «адекватные» названия и быть вложены в указанные стартовые пакеты: ***by.bntu.fitr.poisit.nameofstudent.nameofproject***.
6) В соответствующих важных компонентах программной системы необходимо предусмотреть «защиту от дурака».
7) В качестве хранилище данных использовать Java-массивы!!! Все контейнерный класс должны поддерживать расширяемость, т.е. они не ограничены в объёме хранимых данных и динамически должны расширяться.
8) Проверить все тестовые случаи работы основной бизнес-логики приложения.
9) При разработке программ придерживайтесь соглашений по написанию кода на *Java* (***Java Code-Convention***) !!!

Основное задание 

Необходимо произвести рефакторинг программной системы, созданной в предыдущей лабораторной работе, следующим образом:

- создать отдельный тестовый подпроект в рамках текущего программного решения;
- максимально покрыть основную бизнес-логику модульными тестами;
- реализовать тестовые планы, которые будут включать как методы, которые будут вызываться перед и после каждого тестового случая, так и методы, которые должны вызываться на уровне всего класса;
- предусмотреть также тестирование методов на исключительные ситуации и временные интервалы.

Какому навыку в программировании обучить сложнее всего?

Стив Гэринг, технический директор Easil

Есть пара вещей, которые кажутся мне особенно сложными и по-моему именно они отличают разработчика от программиста. ***Недостаточно сделать, необходимо проверить.***

Нередко можно услышать: «Задача X выполнена, осталось только протестировать программу и можно переходить к задаче Y». Однако на следующий же день можно услышать от разработчика: «Я все еще работаю над задачей X, в ней обнаружилось несколько багов и мне придется исправлять их весь день».

Большинство разработчиков не считает тестирование частью своей работы. Для них это обременительная обязанность, которой должны заниматься тестировщики. То, что выполнено с точки зрения разработчика, как правило проваливается на стадии тестирования. Однако если задачу выполнял программист, проблем на стадии тестирования возникнуть не должно.

Один мой бывший коллега ненавидел тестирование, но я настаивал на том, чтобы он проводил более тщательную проверку продукта на стадии разработки. В конце концов коллега ушел и стал руководителем собственной команды и проекта. Несколько месяцев спустя он сказал мне: «Теперь я понимаю, почему ты заставлял меня заниматься тестированием, ведь оно действительно полезно и приносит значительные плоды». Ваш код должен читаться легко, как книга.

Код не должен выглядеть, как сложное алгебраическое уравнение, он должен легко читаться. Каждый его отрывок должен быть понятен без необходимости обращаться к ссылкам и подсказкам. Переменные и идентификаторы метода должны быть наглядными, а операторы – лаконичными.

Я видел функции с именем *codeToRun(id)*. Можете ли вы догадаться, что делала эта функция? Блокировала 60 строк кода, ожидая загрузки изображений в документе. Поэтому, когда я столкнулся с этим, мне пришлось залезть в код, найти эту функцию и прочитать те 60 строк кода, чтобы понять, какого черта происходит. Если бы функция называлась *blockForImages*, я мог бы избежать этих манипуляций.

Жаль, что у меня не сохранился пример этого кода, но просто скажу, что самое худшее, что я когда-либо видел, это одна строка *JavaScript*, на которую у меня ушло примерно 15 минут, просто чтобы ее понять. Это была одна строка, 7 одиночных символьных переменных и 3 функции. Имя функции не дало мне никаких указаний на ее цель, код внутри был невероятно запутанным, и когда я посмотрел ссылки на функцию, входы и выходы также были одиночными символьными переменными. Я должен был прочитать примерно 50 строк кода, чтобы понять эту единственную строчку.

Причесанный, читабельный код сэкономит вашей команде и всей компании сотни часов в год, а также серьезную денежную сумму, если придется заниматься отладкой уже функционирующего продукта.

Source: https://www.kv.by/post/1055590-kakomu-navyku-v-programmirovanii-obuchit-slozhnee-vsego?utm_source=weekly_letter