

# Введение в тестирование

## Разработка через тестирование

Кулаков Кирилл Александрович

# Классическая схема разработки

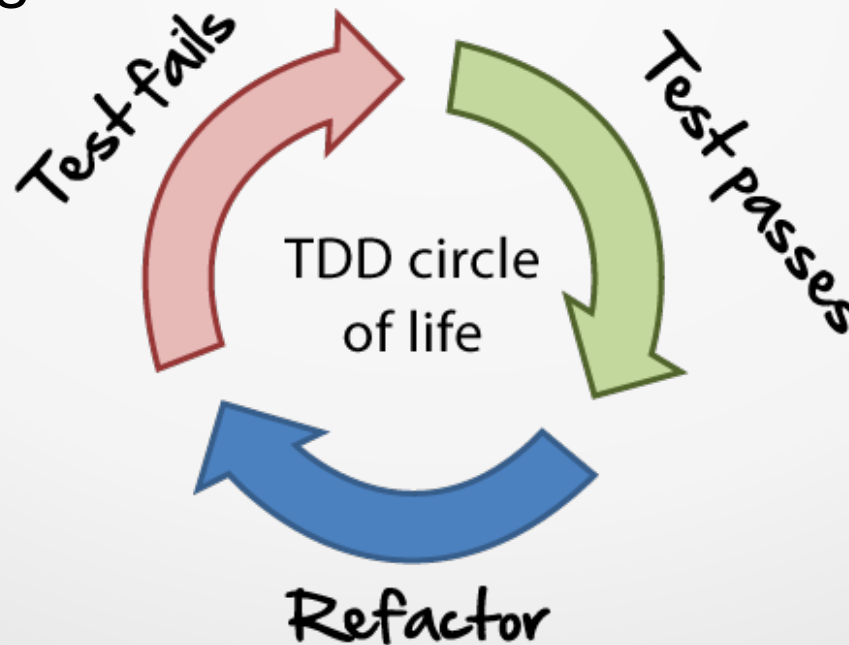
- Получем ТЗ от заказчика
- Немного думаем
- Пишем код
- Немного думаем
- Пишем еще код для предыдущего кода
- ...
- Быстренько проверяем что что-то работает
- Сдаем заказчику

# Недостатки традиционного подхода

- Слишком сложный код для решения простой задачи
- Что-то не работает
- Что-то забыли реализовать
- Монолитный проект: внесение небольшого изменения может привести к краху и требует осмысления
- Нет необходимости (времени и желания) писать тесты и проверки
- Демонстрация заказчику/разработчикам:
  - Что показывать?
  - Как показывать?
  - "Эффект демо"

# Разработка через тестирование

- (Красный) Для требования пишем тесты
- (Зеленый) Пишем новый код только тогда, когда автоматический тест не сработал
- (Синий) Выполняем рефакторинг, устраняем дублирование



# Преимущества

- Всегда работающий код
- Постоянный запуск кода → получаем представление о том, как он работает → помогает нам принимать правильные решения
- Мы самостоятельно пишем свои собственные тесты, так как не можем ждать, пока кто-нибудь напишет их для нас → 100% покрытие тестами
- Быстрое реагирование экосистемы разработки на небольшие модификации кода
- Слабо сцепленные компоненты → упрощение модификации и тестирования
- Меньшее время отладки

# Алгоритм работы

- Написать тест
- Заставить компилироваться
- Запустить тест и убедиться, что он не работает
- Заставить тест работать
- Рефакторинг
- Проверить, что тест работает

# Результаты внедрения

- Если количество дефектов в программе становится достаточно низким, то команда контроля качества может перейти от реактивной к превентивной работе.
- Если количество неприятных сюрпризов будет небольшим, то менеджеры проекта смогут с высокой точностью оценивать трудозатраты и привлекать заказчиков к процессу ежедневной разработки проекта.
- Если темы технических дискуссий будут четко определены, то программисты смогут взаимодействовать друг с другом постоянно, а не раз в день или раз в неделю.
- Если плотность дефектов будет достаточно небольшой мы сможем каждый день получать интегрированный рабочий продукт с добавленной в него новой функциональностью

# Виды тестов

- Тест одного шага (One Step Test)
  - Тест должен добавить новую функциональность.
  - Вы должны реализовать тест за один шаг.
- Начальный тест (Starter Test)
  - Первый тест не выполняет каких либо сложных действий.
  - Тривиальные входные и выходные данные.
- Объясняющий тест (Explanation Test)
  - Объясняйте работу кода в виде тестов.
  - Просите у других объяснить работу их кода в виде тестов.
  - Преобразовывайте диаграммы последовательностей в тесты.



# Виды тестов

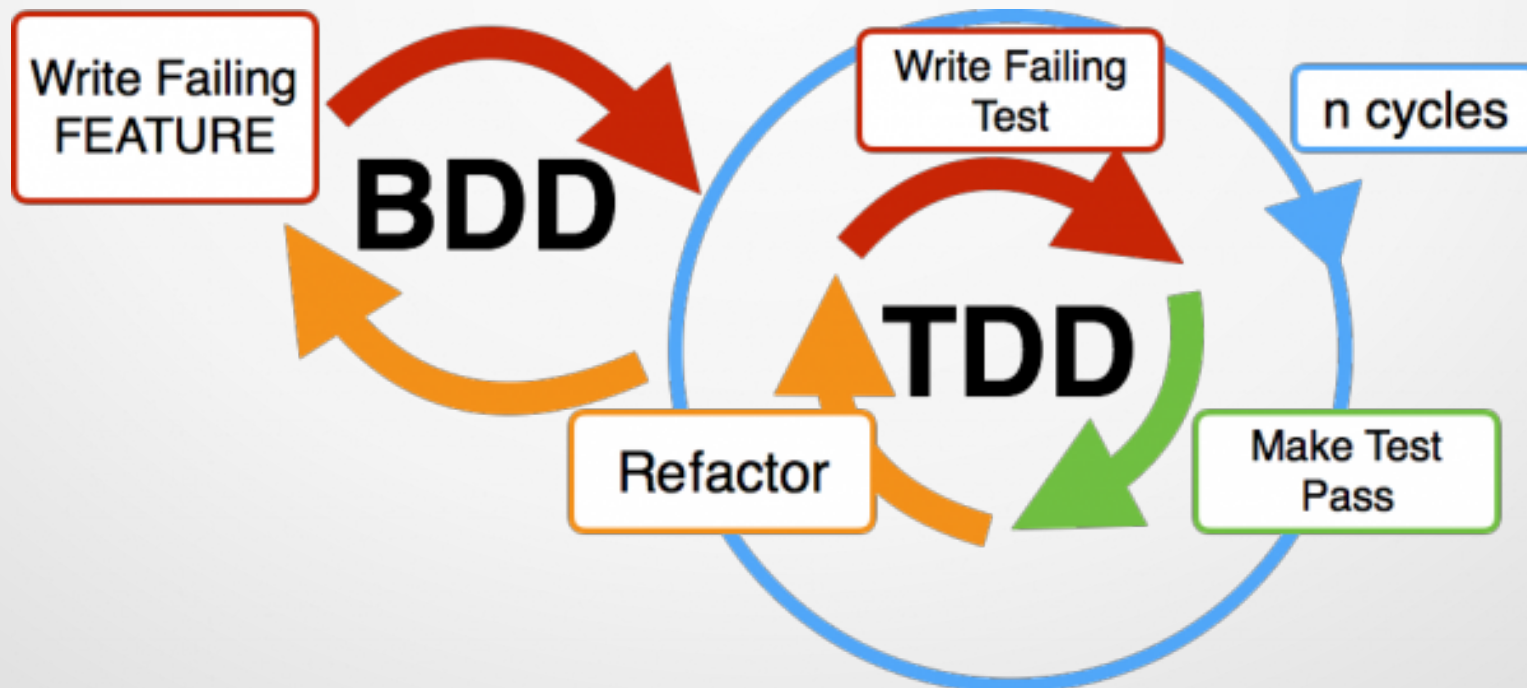
- Тест для изучения (Learning Test)
  - Перед использованием метода библиотеки, поведение которого не совсем ясно.
  - При проверки новой версии библиотеки, которую вы используете.
- Регрессионный тест (Regression Test)
  - Напишите тест прежде чем исправлять ошибку.
- Дочерний тест (Child Test)
  - Разделение большого теста на несколько маленьких

# Проблемы TDD

- Высокий порог вхождения
- Ошибочный тест = ошибочный код
- Поддержка большого количества тестов
- Сложно произвести автоматическое тестирование GUI. Особенно в случае нестандартного интерфейса.
- Сложно тестировать распределенные объекты и многопоточные приложения.
- TDD нельзя использовать для разработки схемы базы данных.
- TDD нельзя использовать для разработки языка программирования.

# Behavior-driven development

- Ответвление TDD
  - Связь кода с требованиями
  - Фокус на поведении а не на тестах
  - Запись требований с помощью обычных фраз



# Пример

- Необходимо реализовать работу со стеком
  - Что такое стек? → структура данных, с порядком объектов «первым вошел, последним вышел» или "последним вошел, первым вышел", есть методы `push()`, `pop()`, `peek()`.
  - Что делает метод `push()`? → `push()` принимает входной объект и помещает во внутренний контейнер, ничего не возвращает
  - Что будет, если передать методу `push()` два объекта, например, сначала `foo`, а потом `bar`? → второй объект при вызове `pop()` должен быть извлечен первым

# Пример

- Что делает метод pop()? → Извлекает последний объект из стека если он там есть и возвращает его
- Что будет делать pop() если в стек еще ничего не было добавлено? → должна выдаваться ошибка
- Что будет, если выполнить команду push() null? → должна выдаваться ошибка

# Реализация. Шаг 1. Подготовка

The screenshot shows the Qt Creator IDE interface. The main editor displays the file `tst_bdd_stack.h` with the following code:

```
1 #include <gtest/gtest.h>
2 #include <gmock/gmock-matchers.h>
3
4 using namespace testing;
5
6 TEST(bdd, hasStack)
7 {
8     MyStack stack();
9 }
10
```

A red error icon is visible on line 8. The **Проблемы** (Problems) panel at the bottom right shows the following error:

Message	File	Line
Using googletest src dir specified at Qt Creator wizard		
In file included from ../../bdd2/tests/main.cpp:1:0:	main.cpp	1
In member function 'virtual void bdd_hasStack_Test::TestBody()':	tst_bdd_stack.h	
'MyStack' was not declared in this scope	tst_bdd_stack.h	8

The **Открытые документы** (Open Documents) panel at the bottom left shows the following files:

- main.cpp
- tst\_bdd\_stack.h

The **Проекты** (Projects) panel on the left shows the project structure:

- bdd2
  - bdd2.pro
  - app
    - tests
      - tests.pro
      - gtest\_dependency
      - Заголовочные
        - tst\_bdd\_stack.h
      - Исходники
        - main.cpp

# Реализация. Шаг 1. Подготовка

The screenshot displays the Qt Creator IDE interface. The main editor window shows the file `tst_bdd_stack.h` with the following C++ code:

```
1 #include <gtest/gtest.h>
2 #include <gmock/gmock-matchers.h>
3
4 #include <mystack.h>
5
6 using namespace testing;
7
8 TEST(bdd, hasStack)
9 {
10     MyStack stack();
11 }
12
```

The left sidebar shows the project structure for `bdd2`, including subdirectories `app`, `tests`, and `Исходники`. The `tests` directory contains `tests.pro`, `gtest_dependency`, and `Заголовочные` files. The `Исходники` directory contains `main.cpp`.

The bottom status bar shows the test results for the `bdd` test:

```
Результаты тестирования
Test summary: 1 успехов, 0 ошибок.
● PASS Выполнение теста bdd
```

The bottom taskbar shows the following tabs: 1 Проблемы, 2 Результаты п..., 3 Вывод прило..., 4 Консоль сбо..., 5 Консоль отл..., 8 Результаты Т...

# Реализация. Шаг 2. BDD

- Добавление тестов BDD

```
TEST(bdd, hasStack) { ... }
```

```
TEST(bdd, shouldThrowExceptionUponNullPush) { ... }
```

```
TEST(bdd, shouldThrowExceptionUponPopWithoutPush) { ... }
```

```
TEST(bdd, shouldPopPushedValue) { ... }
```

```
TEST(bdd, shouldPopSecondPushedValueFirst) { ... }
```

```
TEST(bdd, shouldLeaveValueOnStackAfterPeek) {
```

```
    MyStack stack;
```

```
    MyObj *obj1 = new MyObj(123);
```

```
    MyObj *obj2 = new MyObj(321);
```

```
    stack.push(obj1);
```

```
    stack.push(obj2);
```

```
    MyObj *ret = stack.pop();
```

```
    ASSERT_TRUE(obj2->equals(stack.peek()));
```

```
    ASSERT_TRUE(obj2->equals(ret));
```

```
    ASSERT_TRUE(obj1->equals(stack.peek()));
```

```
}
```



# Реализация. Шаг 3. Запуск

The screenshot displays the Qt Creator IDE with the following components:

- Project Explorer:** Shows a project named 'bdd2' with sub-projects 'app' and 'tests'. The file 'tst\_bdd\_stack.h' is selected under 'tests/Исходники'.
- Editor:** Displays the code for 'tst\_bdd\_stack.h', including a 'MyStack' class and three test functions: 'shouldThrowExceptionUponNullPush', 'shouldThrowExceptionUponPopWithoutPush', and 'shouldPopPushedValue'.
- Test Results Panel:** Shows the output of the test run. The summary indicates 1 success, 2 errors, and 2 fatal errors.

**Test Results Summary:**

Result	Test Name	File	Line
PASS	bdd.hasStack	tst_bdd_stack.h	8
FAIL	bdd.shouldThrowExceptionUponNullPush	tst_bdd_stack.h	15
FAIL	bdd.shouldThrowExceptionUponPopWithoutPush	tst_bdd_stack.h	20
FATAL	Сбой программы тестирования.	tst_bdd_stack.h	23
FATAL	Тест проекта «tests» завершился аварийно.		

# Реализация. Шаг 4. Исправление

- Сценарии с исключениями (3 успеха, 3 ошибки, 1 фатальная)

```
void MyStack::push(MyObj *val) {  
    if (val == NULL) {  
        throw std::invalid_argument("Argument val is null");  
    }  
}
```

```
MyObj *MyStack::pop() {  
    if (startItem == NULL) {  
        throw std::invalid_argument("Stack is empty");  
    }  
    return NULL;  
}
```

# Реализация. Шаг 4. Исправление

- Сценарии с добавлением/удалением элементов (5 успехов, 1 ошибка, 1 фатальная)

```
void MyStack::push(MyObj *val) {  
    if (val == NULL) {  
        throw std::invalid_argument("Argument val is null");  
    }  
  
    StackItem *new_item = (StackItem *)malloc(sizeof(StackItem));  
    new_item->next = startItem;  
    new_item->val = val;  
    startItem = new_item;  
}
```

# Реализация. Шаг 4. Исправление

```
MyObj *MyStack::peek() {  
    if (startItem == NULL) {  
        throw std::invalid_argument("Stack is empty");  
    }  
    return startItem->val;  
}
```

```
MyObj *MyStack::pop() {  
    if (startItem == NULL) {  
        throw std::invalid_argument("Stack is empty");  
    }  
    StackItem *ret_item = startItem;  
    startItem = startItem->next;  
    MyObj *ret = ret_item->val;  
    free(ret_item);  
    return ret;  
}
```

# Реализация. Шаг 5. Завершение

- Ошибка в тесте

```
TEST(bdd, shouldLeaveValueOnStackAfterPeep) {  
    MyStack stack;  
    MyObj *obj1 = new MyObj(123);  
    MyObj *obj2 = new MyObj(321);  
    stack.push(obj1);  
    stack.push(obj2);  
    ASSERT_TRUE(obj2->equals(stack.peek()));  
    MyObj *ret = stack.pop();  
    ASSERT_TRUE(obj2->equals(ret));  
    ASSERT_TRUE(obj1->equals(stack.peek()));  
}
```