

Тестирование ПО

Организация процесса тестирования

Кулаков Кирилл Александрович

Будни тестирования

- Проект большой → много тестов → ручной запуск
- Демонстрация работы / качества приложения
- Демонстрация ошибки разработчику
- Регрессионное тестирование
- Несколько проектов одновременно → необходимость переключаться
- Текучка кадров / неопределенный круг разработчиков
- Отчет о проекте начальству / заказчикам

Задача автоматизации тестирования

- Цель: сокращение издержек тестирования
 - уменьшение ручной работы
 - уменьшение человеческого фактора
 - инструмент для проверки нового кода
 - Составление отчетов руководству → онлайн режим отчета
- Недостатки
 - накладные расходы на создание тестов
 - накладные расходы на автоматизацию
 - накладные расходы на поддержку системы тестирования

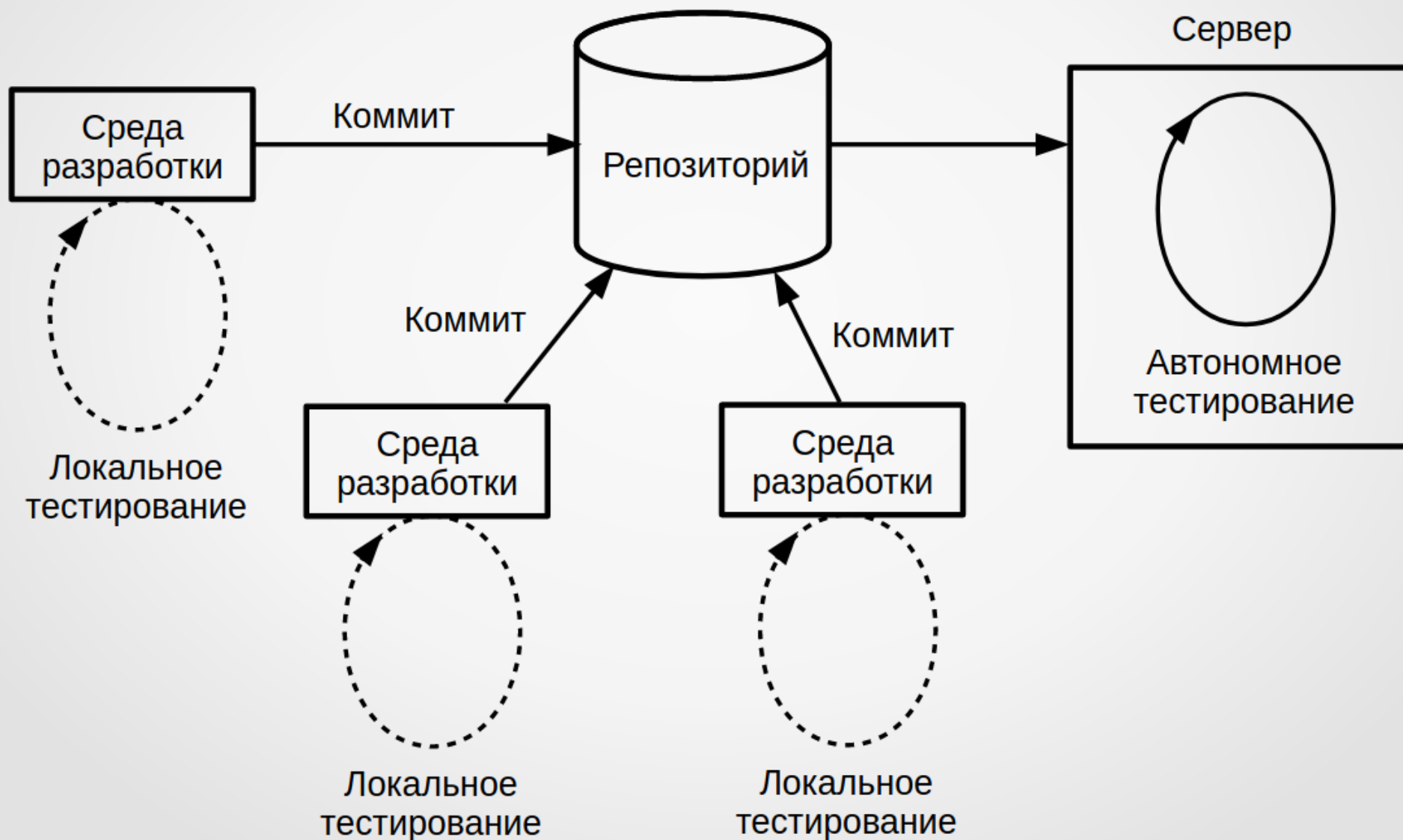
Варианты автоматизации

- Запуск тестов при необходимости
 - скрипт для ручного запуска
 - разбиение тестов на группы
- Запуск тестов после события
 - триггеры для выбранных событий (коммиты, запросы на слияния, ...)
 - выделенные ресурсы (сервера)
- Систематический запуск тестов
 - планировщик задач (ночные тесты, еженедельные тесты, ...)

Непрерывная интеграция (CI)

- практика разработки программного обеспечения
- слияния рабочих копий в общую основную ветвь разработки несколько раз в день
- выполнение частых автоматизированных сборок проекта
- Цель: скорейшее выявление потенциальных дефектов и решение интеграционных проблем.

Непрерывная интеграция



Непрерывная интеграция

- Преимущества
 - проблемы интеграции выявляются и исправляются быстро, что оказывается дешевле;
 - немедленный прогон модульных тестов для свежих изменений;
 - постоянное наличие текущей стабильной версии вместе с продуктами сборки — для тестирования, демонстрации, и т.д.
 - немедленный эффект от неполного или неработающего кода приучает разработчиков к работе в итеративном режиме с более коротким циклом.

Непрерывная интеграция

- Недостатки
 - затраты на поддержку работы непрерывной интеграции;
 - потенциальная необходимость в выделенном сервере под нужды непрерывной интеграции;
 - немедленный эффект от неполного или неработающего кода отучает разработчиков от выполнения периодических резервных включений кода в репозиторий.
 - использование ветвей для новых изменений → увеличение расхода ресурсов

Реализация автоматизации тестирования

- Инструменты:
 - qt creator (среда разработки + локальное тестирование)
 - google test (создание тестов)
 - GitHub (репозиторий кода)
 - GitHub actions (непрерывная сборка)

Создание тестов для автоматического тестирования

- Задача: обеспечить автономность запуска
 - доступность исходных данных
 - доступность окружения объекта тестирования (других модулей / программ / ...)
 - возможность получить результат работы теста
 - независимость тестов друг от друга и от истории запусков
- Выводы
 - все что нужно для теста в репозиторий
 - поиск "маркеров" в результате работы объекта тестирования
 - удаление временных файлов по завершении работы теста

GitHub Actions

- Мониторинг пушей в репозиторий (GitHub)
- Запуск скриптов сборки (.github/workflows/...) свободным обработчиком (worker) на чистой виртуальной машине
- Контроль результата по кодам возврата
- Предоставление бейджей с результатами сборки
- Документация: <https://docs.github.com/en/actions>

gtest/.github/workflows/test-action.yml

- 1) name: CI/CD GitHub Actions
- 2) on: [push]
- 3) jobs:
- 4) gtest-tests:
- 5) runs-on: ubuntu-latest
- 6) steps:
- 7) - uses: actions/checkout@v2
- 8) - run: git submodule init
- 9) - run: git submodule update
- 10) - run: sudo apt-get update -y && sudo apt-get install -y qt5-qmake qt5-default
- 11) - run: **qmake**
- 12) - run: **make**
- 13) - run: **./tests/tests**

Результат работы

← CI/CD GitHub Actions

✓ Update sonar-project.properties #34

Re-run all jobs



Summary

Jobs

✓ gtest-tests

✓ sonarcloud-check

Run details

Usage

Workflow file

gtest-tests

succeeded 3 minutes ago in 1m 4s

Search logs



- > ✓ Set up job 1s
- > ✓ Run actions/checkout@v2 1s
- > ✓ Run git submodule init 0s
- > ✓ Run git submodule update 1s
- > ✓ Run pip install --user cpp-coveralls 5s
- > ✓ Run sudo apt-get update -y && sudo apt-get install -y qt5-qmake qt5-default 34s
- > ✓ Run qmake 1s
- > ✓ Run make 18s
- ▼ ✓ Run ./tests/tests 0s

```
1 ▶ Run ./tests/tests
4 [=====] Running 9 tests from 4 test cases.
5 [-----] Global test environment set-up.
6 [-----] 2 tests from test1
7 [ RUN      ] test1.suite1
8 [      OK  ] test1.suite1 (0 ms)
9 [ RUN      ] test1.suite2
10 [      OK  ] test1.suite2 (0 ms)
11 [-----] 2 tests from test1 (0 ms total)
12
```

Результат ошибки

← CI/CD GitHub Actions

✖ Update fibonacci_test.h #35

🔄 Re-run jobs



🏠 Summary

Jobs

✖ gtest-tests

🕒 sonarcloud-check

Run details

🕒 Usage

📄 Workflow file

gtest-tests

failed 2 minutes ago in 45s

🔍 Search logs



- > ✔ Set up job 1s
- > ✔ Run actions/checkout@v2 1s
- > ✔ Run git submodule init 0s
- > ✔ Run git submodule update 2s
- > ✔ Run pip install --user cpp-coveralls 2s
- > ✔ Run sudo apt-get update -y && sudo apt-get install -y qt5-qmake qt5-default 22s
- > ✔ Run qmake 0s
- > ✔ Run make 14s
- ▼ ✖ Run ./tests/tests 0s

```
1 ▶ Run ./tests/tests
4 [=====] Running 9 tests from 4 test cases.
5 [-----] Global test environment set-up.
6 [-----] 2 tests from test1
7 [ RUN      ] test1.suite1
8 [      OK  ] test1.suite1 (0 ms)
9 [ RUN      ] test1.suite2
10 [      OK  ] test1.suite2 (0 ms)
```

Результат ошибки

🕒 Usage

📄 Workflow file

```
20 [ OK ] fibonacciTest.num0 (0 ms)
21 [ RUN ] fibonacciTest.greater2
22 fibonacci_test.h:21: Failure
23     Expected: fibonacci(5)
24     Which is: 5
25 To be equal to: 2
26 [ FAILED ] fibonacciTest.greater2 (0 ms)
27 [ RUN ] fibonacciTest.negative
28 [ OK ] fibonacciTest.negative (0 ms)
29 [ RUN ] fibonacciTest.inputFile
30 [ OK ] fibonacciTest.inputFile (0 ms)
31 [-----] 4 tests from fibonacciTest (0 ms total)
32
33 [-----] 2 tests from TestStdOut
34 [ RUN ] TestStdOut.TestStdOut
35 [ OK ] TestStdOut.TestStdOut (1 ms)
36 [ RUN ] TestStdOut.usingCapture
37 [ OK ] TestStdOut.usingCapture (0 ms)
38 [-----] 2 tests from TestStdOut (1 ms total)
39
40 [-----] Global test environment tear-down
41 [=====] 9 tests from 4 test cases ran. (1 ms total)
42 [ PASSED ] 8 tests.
43 [ FAILED ] 1 test, listed below:
44 [ FAILED ] fibonacciTest.greater2
45
46 1 FAILED TEST
47 Error: Process completed with exit code 1.
```



Coveralls

0s



Post Run actions/checkout@v2

0s

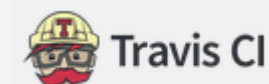


Complete job

0s

Другие среды для автоматизации

- <https://circleci.com/> - CircleCI
- <https://jenkins.io/> - Jenkins
- <https://www.jetbrains.com/teamcity/> - TeamCity
- <https://codeship.com/> - Codeship
- <https://gitlab.com/> - Gitlab
- <https://www.travis-ci.com/> - Travis CI



GitLab



Пример Gitlab CI

S shower

Project information

Репозиторий

Обсуждения 12

Запросы на слияние 1

CI/CD

Сборочные линии

Редактор

Задания

Расписания

Безопасность и комплаенс

Deployments

Infrastructure

Monitor

Аналитика

Wiki

Сниппеты

Настройки

smalt > shower > Задания > #12989

✓ пройдено

Job test запущено 6 мес. назад по Кулаков Кирилл

Search job log

1 Running with gitlab-runner 14.10.1 (f761588f)

2 on gitlab 19a5f92d

3 **Preparing the "docker" executor** 00:05

4 Using Docker executor with image php:7.3-fpm ...

5 Pulling docker image php:7.3-fpm ...

6 Using docker image sha256:fdccf4773f9eb394660c00ebbb74bc2c80cd95cec81a00704e2824a25aef652f for php:7.3-fpm with digest php@sha256:2d68e401d2d3b9f8a6572791cd7a25062450c43ff52e58391146809741ad0885 ...

8 **Preparing environment** 00:01

9 Running on runner-19a5f92d-project-6-concurrent-0 via gitlab...

11 **Getting source from Git repository** 00:03

12 **Fetching changes...**

13 Reinitialized existing Git repository in /builds/smalt/shower/.git/

14 **Checking out 628885ee as master...**

15 Removing .phpunit.result.cache

16 Removing composer.lock

17 Removing coverage.xml

18 Removing public/

19 Removing tests-junit.xml

20 Removing vendor/

21 **Skipping Git submodules setup**

23 **Restoring cache** 00:01

24 **Checking cache for default-protected...**

25 No URL provided, cache will not be downloaded from shared cache server. Inst...

test

Длительность: 4 минуты 19 секунд

Завершено: 6 мес. назад

Queued: 1 секунда

Таймаут: 1h (из project)

Runner: #1 (19a5f92d) gitlab

Покрытие: 9.25%

Артефакты задания

These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Коммит 628885ee



Merge branch 'visualization' into 'master'

✓ Pipeline #3790 for master

test

→ ✓ test

Пример Gitlab CI

 .gitlab-ci.yml  2.06 КиБ

Редактировать



Заменить

Удалить



```
1 image: php:7.3-fpm
2
3 stages:
4   - test
5   - quality
6
7 # Composer stores all downloaded packages in the vendor/ directory.
8 # Do not use the following if the vendor/ directory is committed to
9 # your git repository.
10 cache:
11   paths:
12     - vendor/
13
14 test:
15   stage: test
16   rules:
17     - if: '$CI_PIPELINE_SOURCE == "merge_request_event"' # Run code quality job in merge request pipelines
18     - if: '$CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH'      # Run code quality job in pipelines on the default branch
19     - if: '$CI_COMMIT_TAG'                               # Run code quality job in pipelines for tags artifacts:
20   script:
21     - apt-get update -y
22     - apt-get install -y git curl wget libpng-dev libzip-dev zlib1g-dev graphviz libicu-dev g++ libmagickwand-dev
23     - docker-php-ext-install gd zip exif
24 # Install Xdebug
25     - pecl channel-update pecl.php.net
26     - pecl install xdebug
27     - docker-php-ext-enable xdebug
28     - curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer
```

Классическая схема разработки

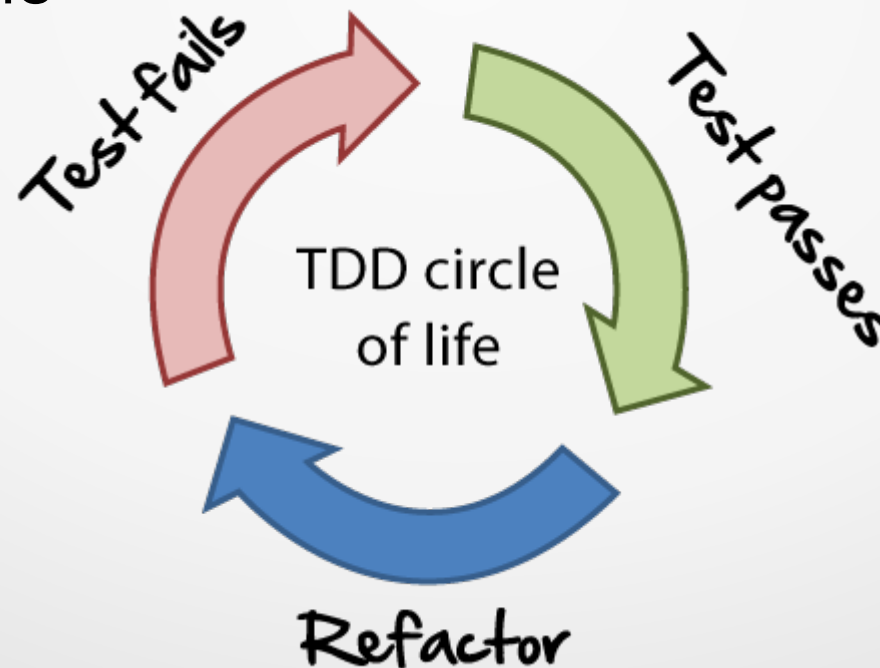
- Получем ТЗ от заказчика
- Немного думаем
- Пишем код
- Немного думаем
- Пишем еще код для предыдущего кода
- ...
- Быстренько проверяем что что-то работает
- Сдаем заказчику

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
- Проблемы демонстрации заказчику/разработчикам:
 - Что показывать?
 - Как показывать?
 - "Эффект демо"

Test-Driven Development (TDD)

- (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 объяснять работу их кода в виде тестов.
 - Преобразовывайте диаграммы последовательностей в тесты.

Виды тестов

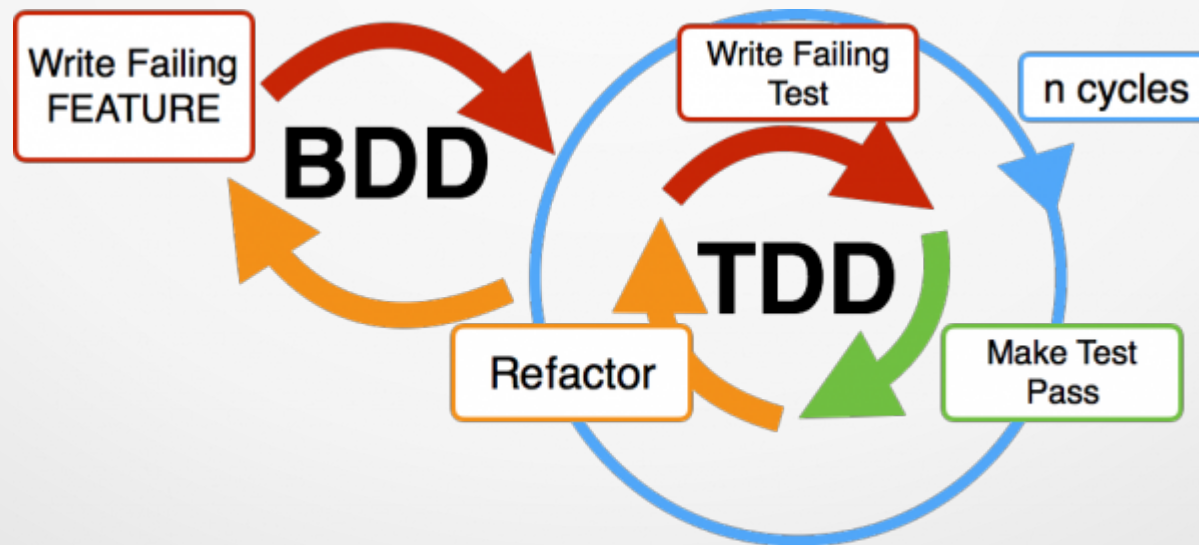
- Тест для изучения (Learning Test)
 - Перед использованием метода библиотеки, поведение которого не совсем ясно.
 - При проверки новой версии библиотеки, которую вы используете.
- Регрессионный тест (Regression Test)
 - Напишите тест прежде чем исправлять ошибку.
- Дочерний тест (Child Test)
 - Разделение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

- 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 использования практик

- 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:

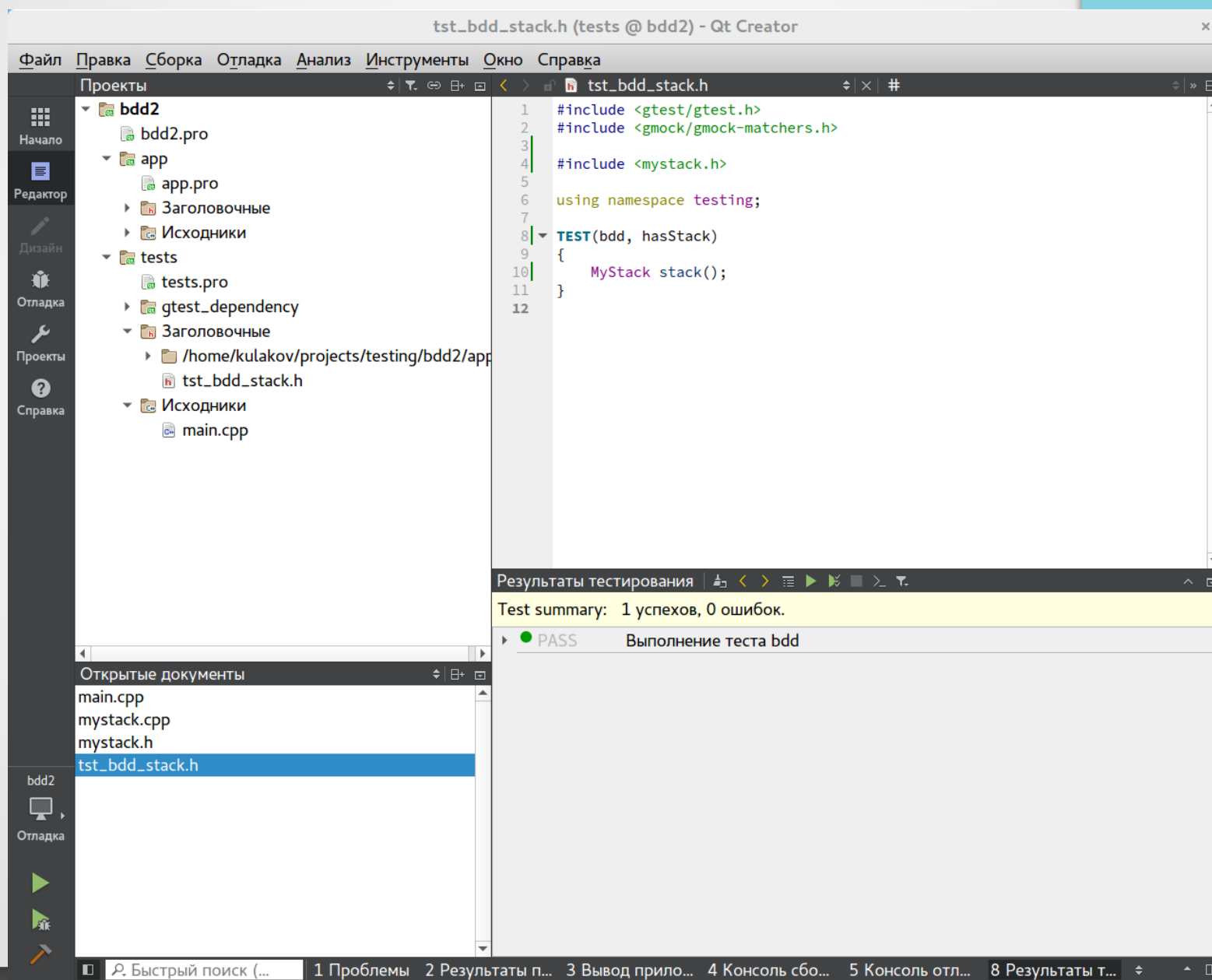
```
1 #include <gtest/gtest.h>
2 #include <gmock/gmock-matchers.h>
3
4 using namespace testing;
5
6 TEST(bdd, hasStack)
7 {
8     MyStack stack();
9 }
10
```

A red error icon is visible next to line 8. The bottom panel shows the 'Problems' (Проблемы) tab with the following error:

File	Line	Message
main.cpp	1	In file included from ../../bdd2/tests/main.cpp:1:0:
tst_bdd_stack.h		In member function 'virtual void bdd_hasStack_Test::TestBody()':
tst_bdd_stack.h	8	'MyStack' was not declared in this scope

The left sidebar shows the project structure for `bdd2`, including `bdd2.pro`, `app`, `tests` (containing `tests.pro`, `gtest_dependency`, `Заголовочные` (containing `tst_bdd_stack.h`), and `Исходники` (containing `main.cpp`)). The bottom status bar shows the search bar and tabs for 'Problems', 'Results', 'Output', 'Console', 'Debugger', and 'Results'.

Реализация. Шаг 1. Подготовка



Реализация. Шаг 2. TDD

- Добавление тестов TDD

```
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 shows the Qt Creator IDE with the project 'bdd2' open. The file explorer on the left shows the project structure, including 'tests' and 'tst_bdd_stack.h'. The main editor displays the implementation of the stack in 'tst_bdd_stack.h'. The test results panel at the bottom shows the outcome of the tests.

Code in `tst_bdd_stack.h`:

```
10 MyStack stack;
11 }
12
13 TEST(bdd, shouldThrowExceptionUponNullPush) {
14     MyStack stack;
15     ASSERT_ANY_THROW(stack.push(NULL));
16 }
17
18 TEST(bdd, shouldThrowExceptionUponPopWithoutPush) {
19     MyStack stack;
20     ASSERT_ANY_THROW(stack.pop());
21 }
22
23 TEST(bdd, shouldPopPushedValue) {
24     MyStack stack;
25     MyObj *obj = new MyObj(123);
26     stack.push(obj);
27     MyObj *ret = stack.pop();
```

Test Results:

Test summary: 1 успехов, 2 ошибок, 2 фатальных.

Result	Test Name	Location
FAIL	Выполнение теста bdd	
PASS	bdd.hasStack	tst_bdd_stack.h 8
	Выполнение заняло 0 ms.	tst_bdd_stack.h 8
FAIL	bdd.shouldThrowExceptionUponNullPush	tst_bdd_stack.h 15
	Выполнение заняло 0 ms.	tst_bdd_stack.h 13
FAIL	bdd.shouldThrowExceptionUponPopWithoutPush	tst_bdd_stack.h 20
	Выполнение заняло 0 ms.	tst_bdd_stack.h 18
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()));  
}
```