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

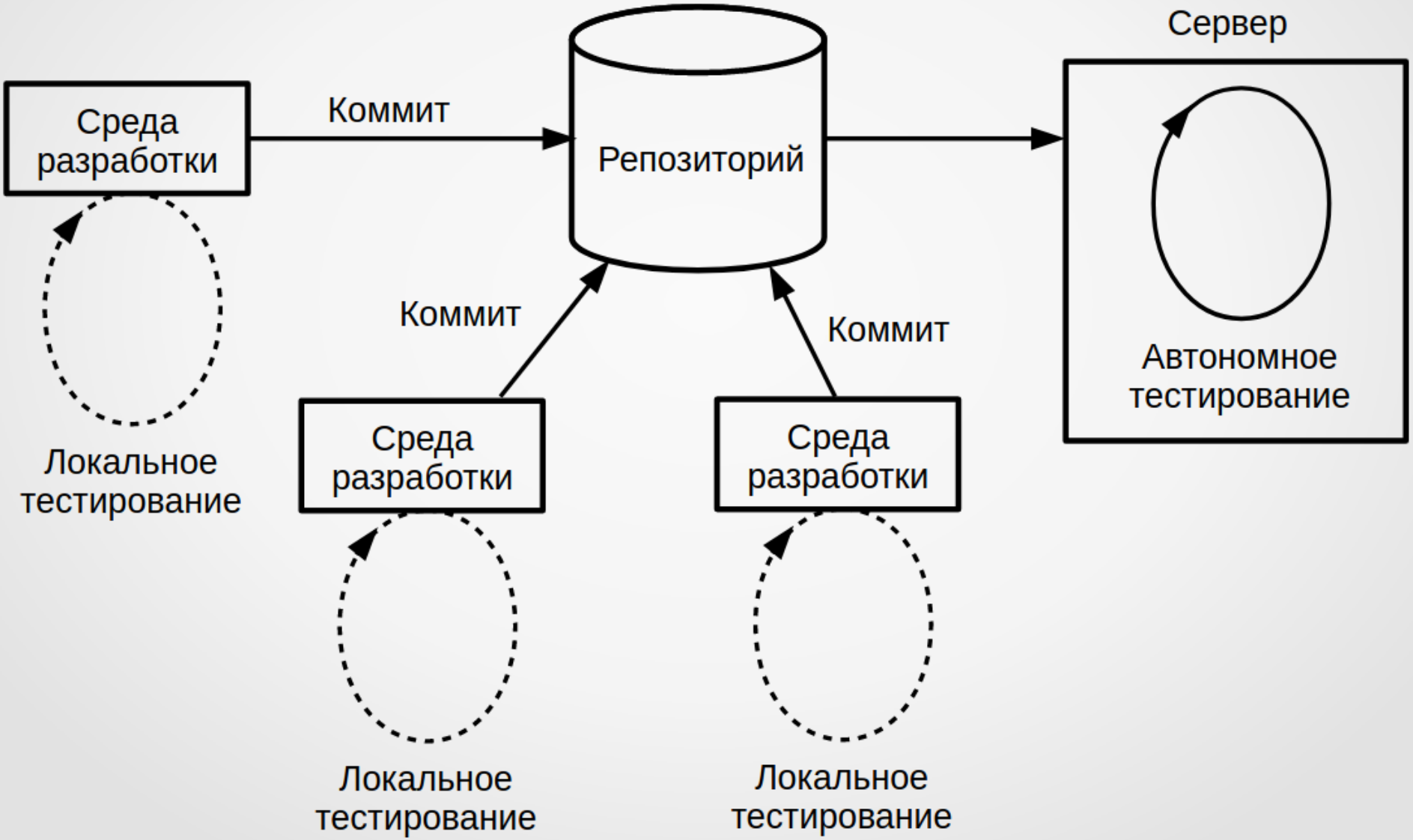
Варианты автоматизации

- 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 сборок — 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

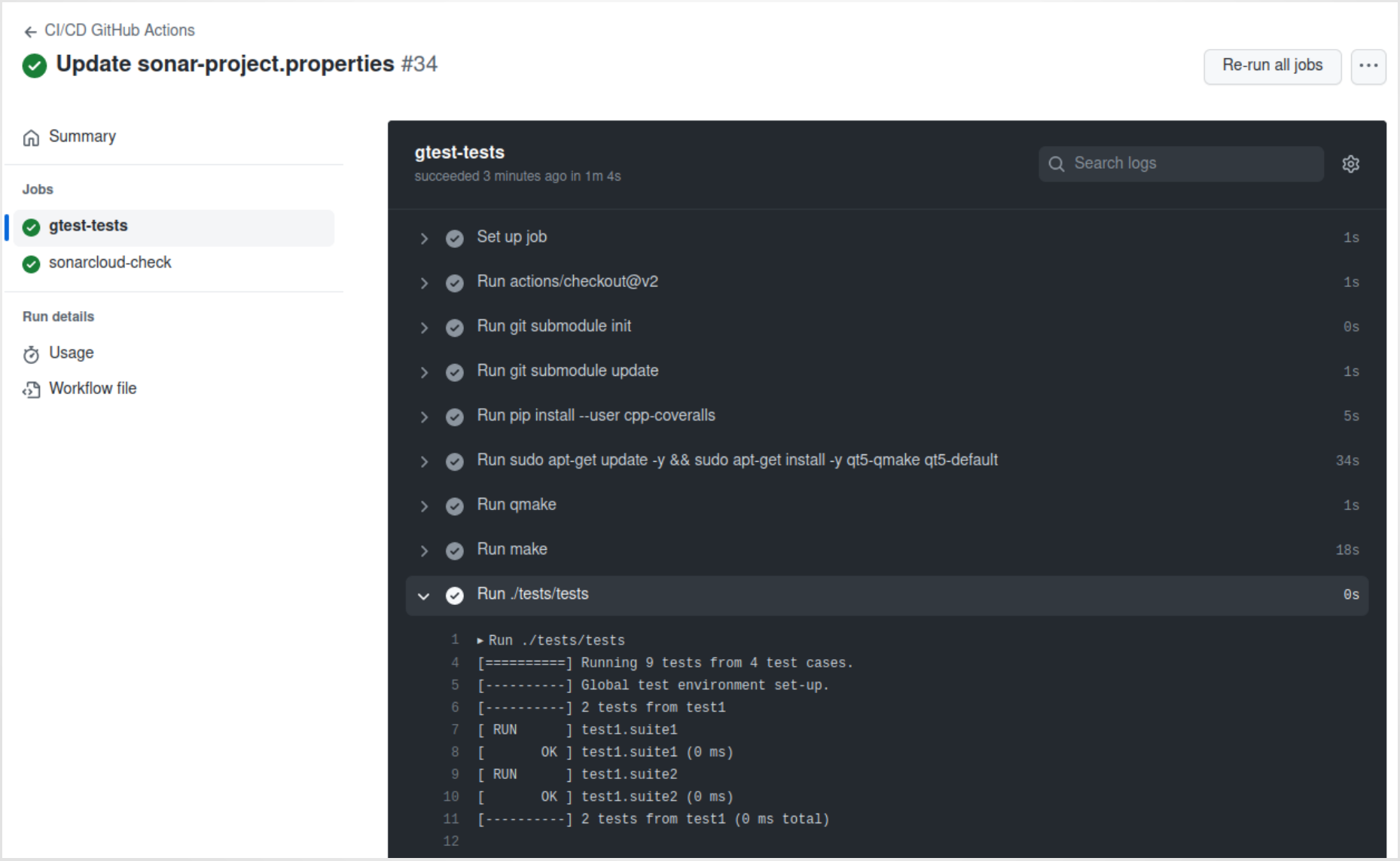
- Мониторинг пушей в репозиторий (GitHub)
- Запуск скриптов сборки (.github/workflows/…) свободным обработчиком (worker) на чистой виртуальной машине
- Контроль результата по кодам возврата
- Предоставление бейджей с результатами сборки

- Документация: https://docs.github.com/en/actions

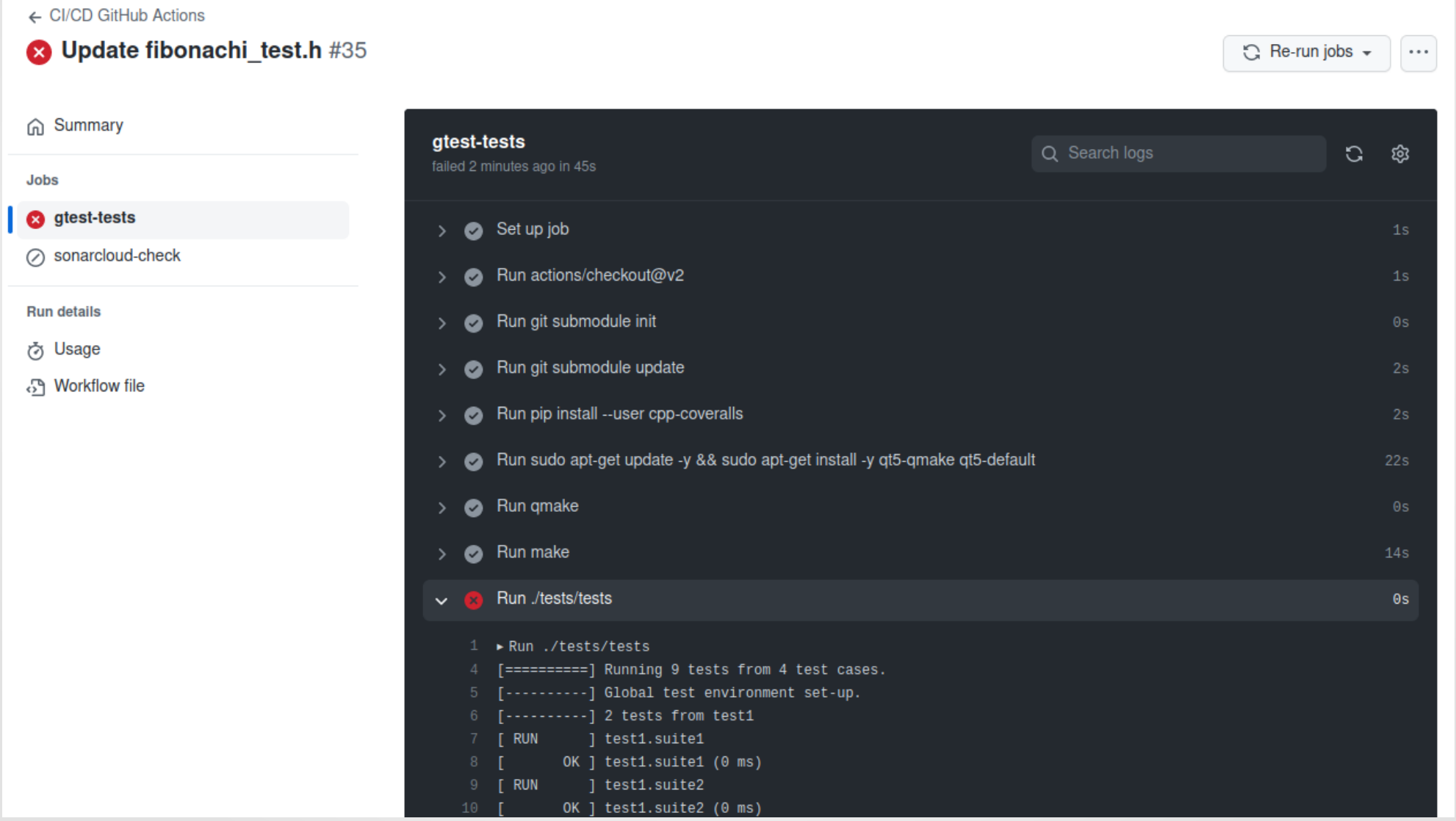
gtest/.github/workflows/test-action.yml

```
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



Результат ошибки



Результат ошибки

Usage

Workflow file

```
20  [       OK ] fibonachiTest.num0 (0 ms)
21  [ RUN      ] fibonachiTest.greather2
22  fibonachi_test.h:21: Failure
23        Expected: fibonachi(5)
24        Which is: 5
25  To be equal to: 2
26  [  FAILED  ] fibonachiTest.greather2 (0 ms)
27  [ RUN      ] fibonachiTest.negative
28  [       OK ] fibonachiTest.negative (0 ms)
29  [ RUN      ] fibonachiTest.inputFile
30  [       OK ] fibonachiTest.inputFile (0 ms)
31  [----------] 4 tests from fibonachiTest (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  ] fibonachiTest.greather2
45
46   1 FAILED TEST
47  Error: Process completed with exit code 1.
```

Coveralls 0s

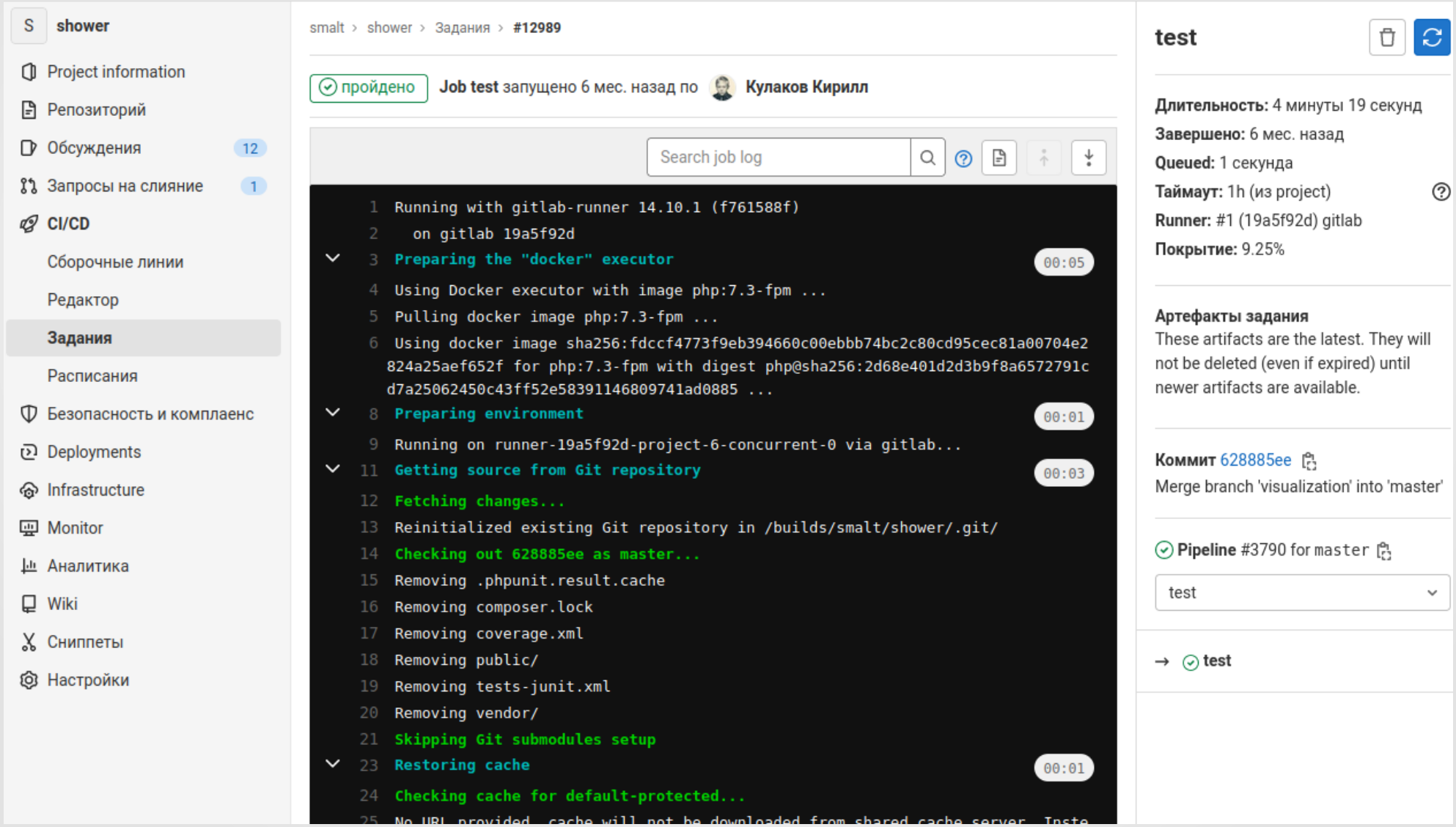
Post Run actions/checkout@v2 0s

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

Пример Gitlab CI



Пример Gitlab CI

.gitlab-ci.yml 2.06 КиБ

Редактировать | Заменить | Удалить

```yaml
image: php:7.3-fpm

stages:
  - test
  - quality

# Composer stores all downloaded packages in the vendor/ directory.
# Do not use the following if the vendor/ directory is committed to
# your git repository.
cache:
  paths:
    - vendor/

test:
  stage: test
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"' # Run code quality job in merge request pipelines
    - if: '$CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH'      # Run code quality job in pipelines on the default branc
    - if: '$CI_COMMIT_TAG'                               # Run code quality job in pipelines for tags  artifacts:
  script:
    - apt-get update -y
    - apt-get install -y git curl wget libpng-dev libzip-dev zlib1g-dev graphviz libicu-dev g++ libmagickwand-dev
    - docker-php-ext-install gd zip exif
# Install Xdebug
    - pecl channel-update pecl.php.net
    - pecl install xdebug
    - docker-php-ext-enable xdebug
    - curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer
```

Классическая схема разработки

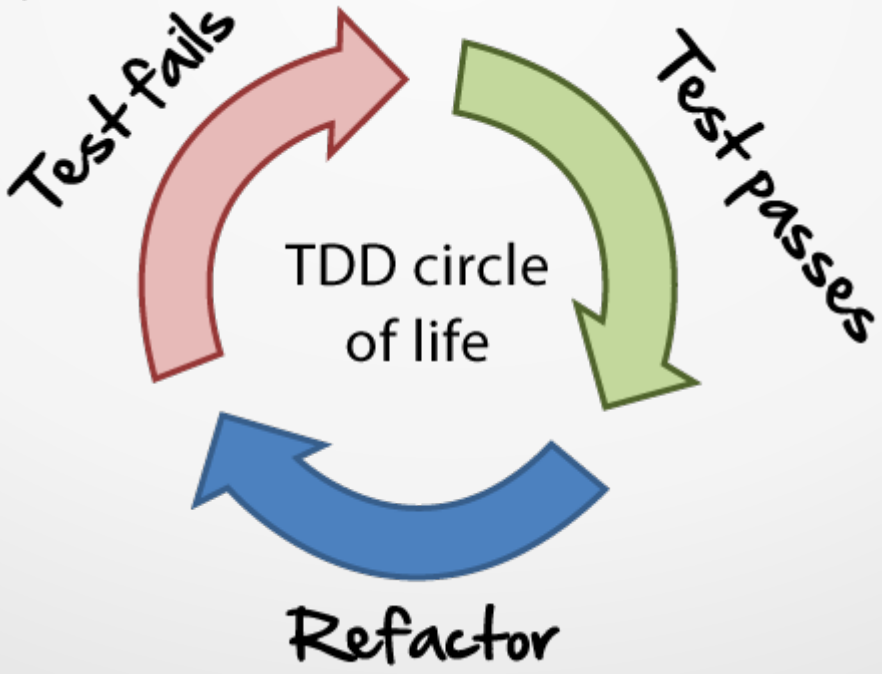
- Получем ТЗ от заказчика
- Немного думаем
- Пишем код
- Немного думаем
- Пишем еще код для предыдущего кода
- ...
- Быстренько проверяем что что-то работает
- Сдаем заказчику

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

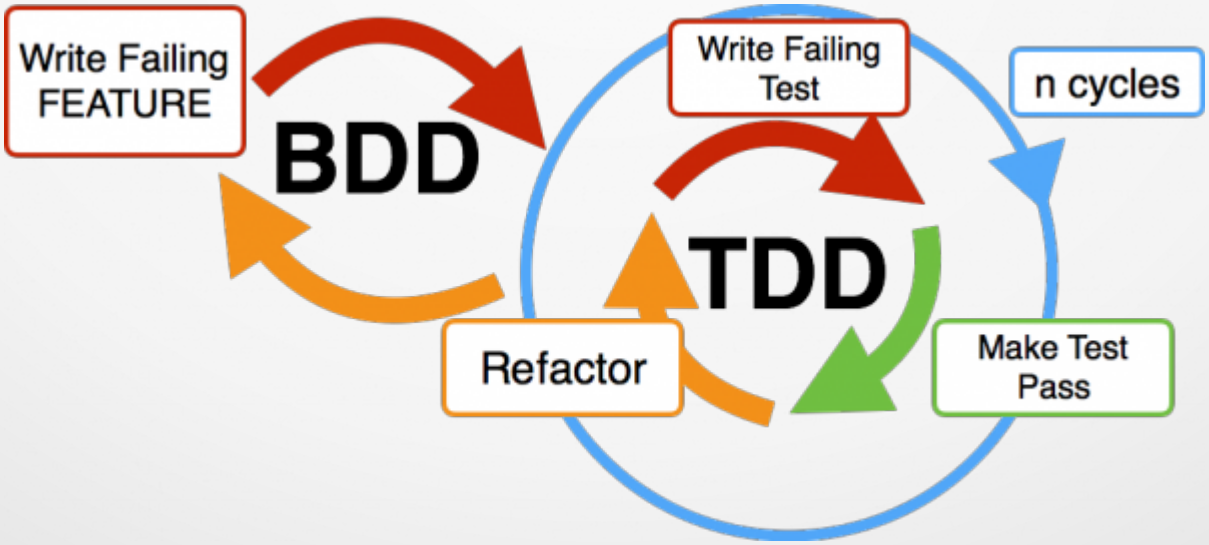
- Тест для изучения (Learning Test)
  - Перед использованием метода библиотеки, поведение которого не совсем ясно.
  - При проверки новой версии библиотеки, которую вы используете.
- Регрессионный тест (Regression Test)
  - Напишите тест прежде чем исправлять ошибку.
- Дочерний тест (Child Test)
  -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



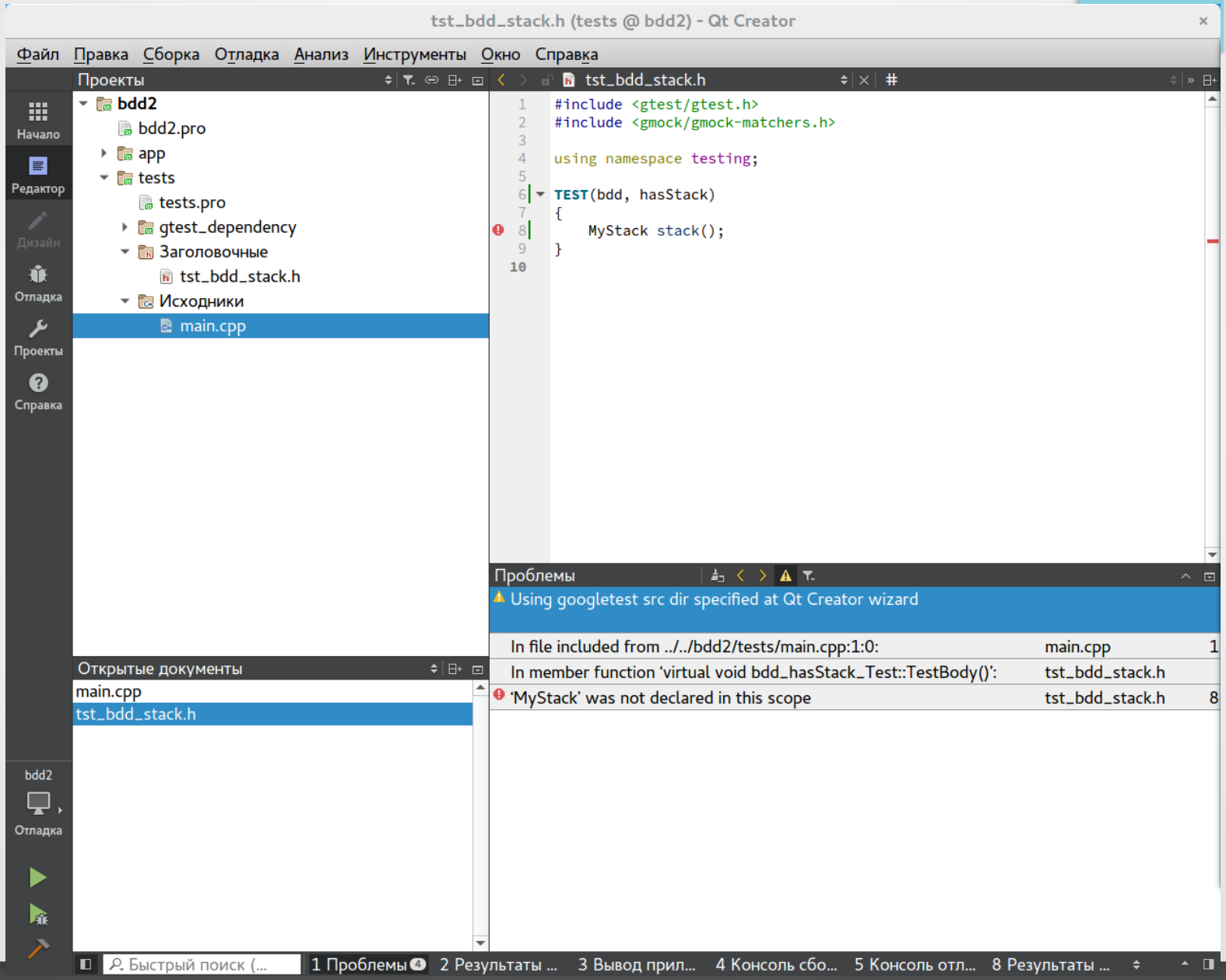
Пример использования практик

- 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

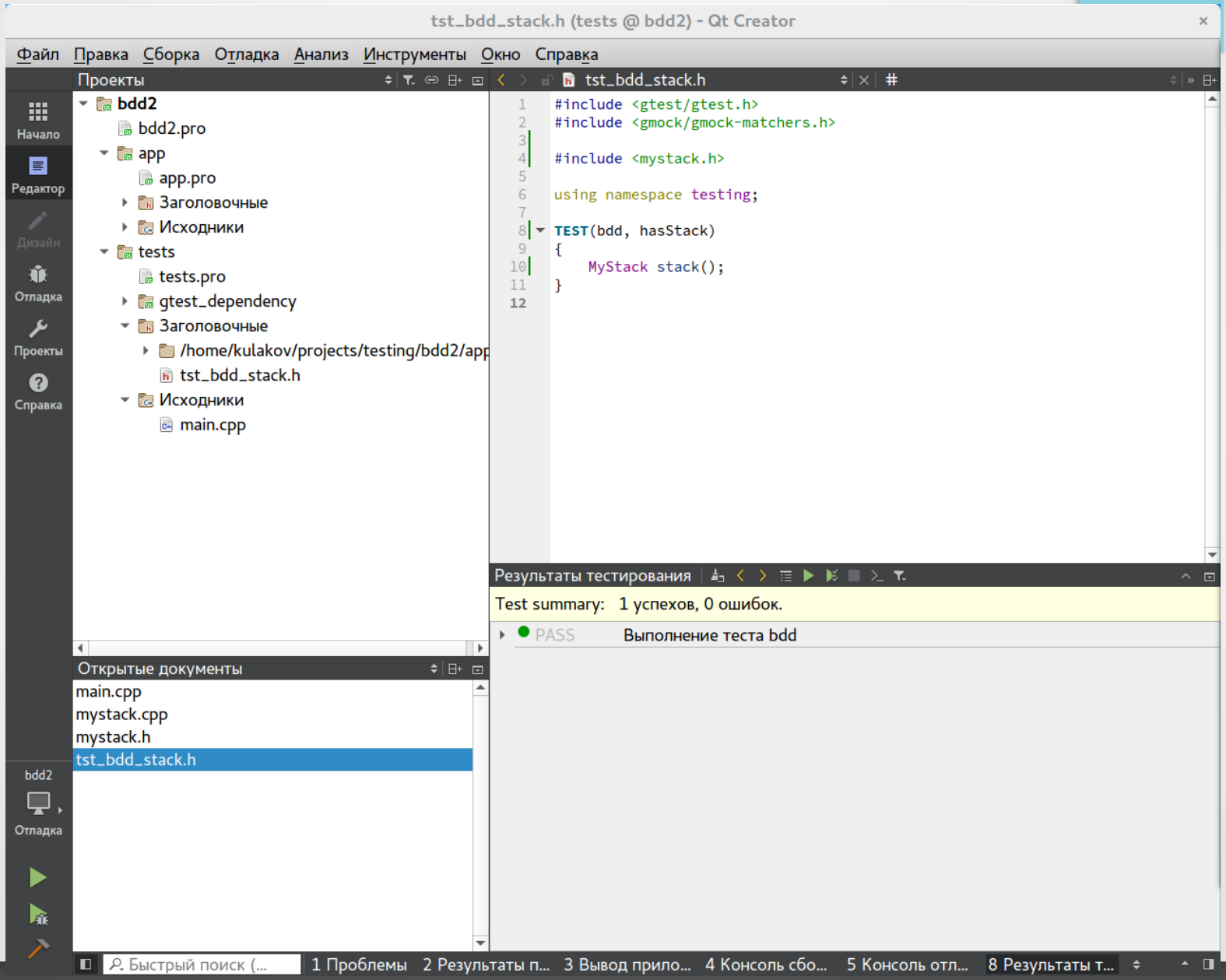
Пример использования практик

- 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



Реализация. Шаг 1. Подготовка

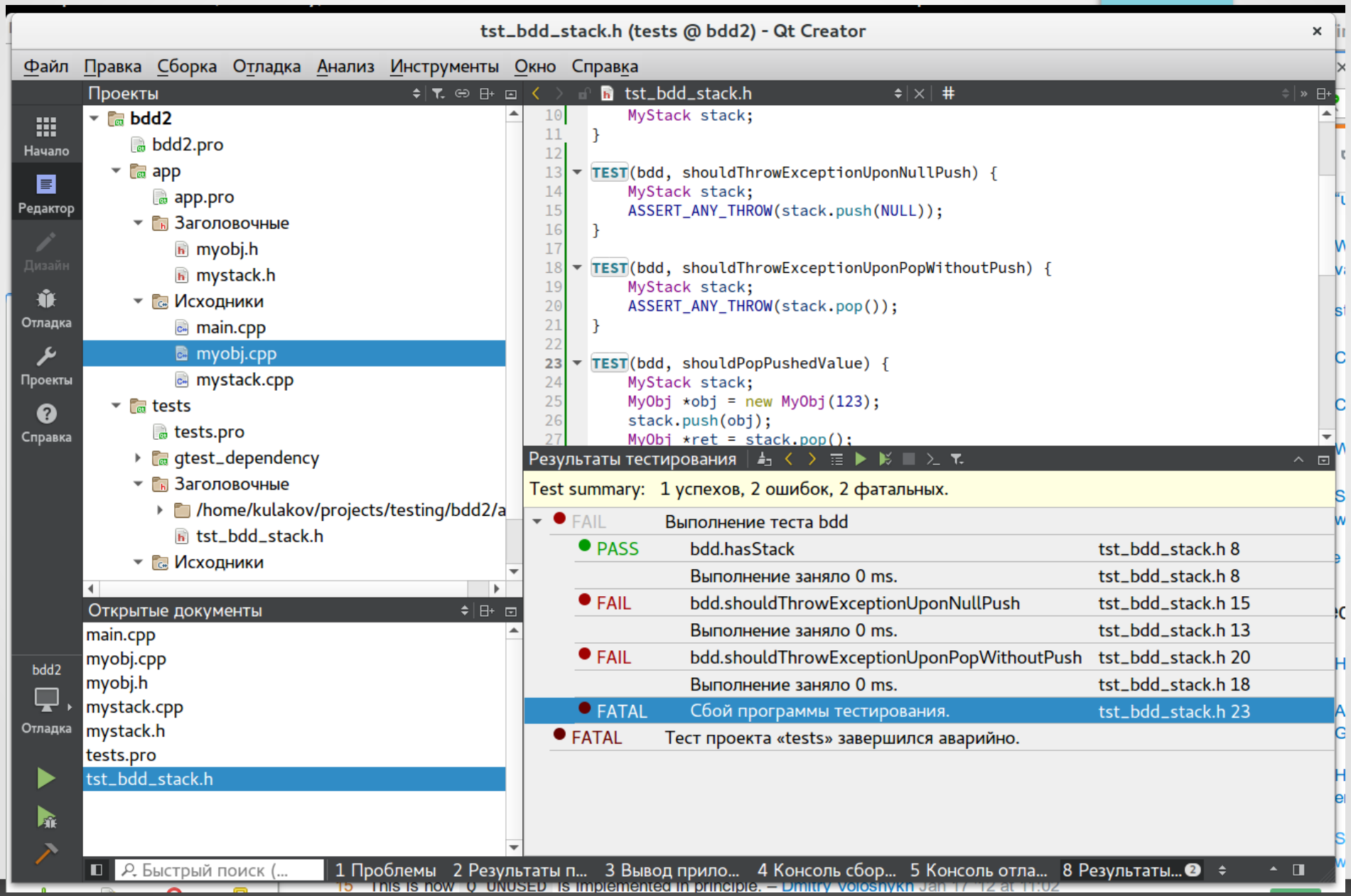


Реализация. Шаг 2. TDD

- 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



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()));
}
```