# Технология и методы программирования

Лекция 5. Тестирование

# Определения

**Тестировние ПО** —  процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом.

**Качество ПО** — это совокупность характеристик ПО, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

# Определения v2

- **Обеспечение качества** - это совокупность мероприятий, охватывающих все технологические этапы разработки, выпуска и эксплуатации программного обеспечения (ПО) информационных систем, предпринимаемых на разных стадиях жизненного цикла ПО для обеспечения требуемого уровня качества выпускаемого продукта.
- **Контроль качества** - это совокупность действий, проводимых над продуктом в процессе разработки для получения информации о его актуальном состоянии в разрезах: "готовность продукта к выпуску", "соответствие зафиксированным требованиям", "соответствие заявленному уровню качества продукта".

# Предпосылки для развития контроля качества

- экономическая целесообразность
- конкурентные преимущества
- репутационная составляющая

# Тестирование. А зачем?

- Проверка соответствия требованиям
- Обнаружение проблем на ранних этапах
- Обнаружение нетипичных сценариев использования ПО
- Повышение качества ПО

# Примеры ошибок. 1

```c
#include <stdio.h>
#include <stdlib.h>

int
get_file_size(FILE *f)
{
        int cur, sz;

        cur = ftell(f);
        fseek(f, 0, SEEK_END);
        sz = ftell(f);
        fseek(f, cur, SEEK_SET);
        return sz;
}

int
main(int argc, const char *argv[])
{
        char *mybuf = NULL;
        FILE *f;
        size_t sz;

        f = fopen(argv[1], "rb");
        sz = get_file_size(f);
        mybuf = malloc(sz);
        int rc = fread(mybuf, 1, sz, f);
        fwrite(mybuf, 1, sz, stdout);
        return 0;
}
```

# Примеры ошибок. 2

```
 1 int
 2 sum(int *arr, int arrsz)
 3 {
 4         int i;
 5         int sum;
 6         for (i = 0; i < arrsz; i++) {
 7                 sum += arr[i];
 8         }
 9         return sum;
10 }
11
```

# Откуда берутся ошибки?

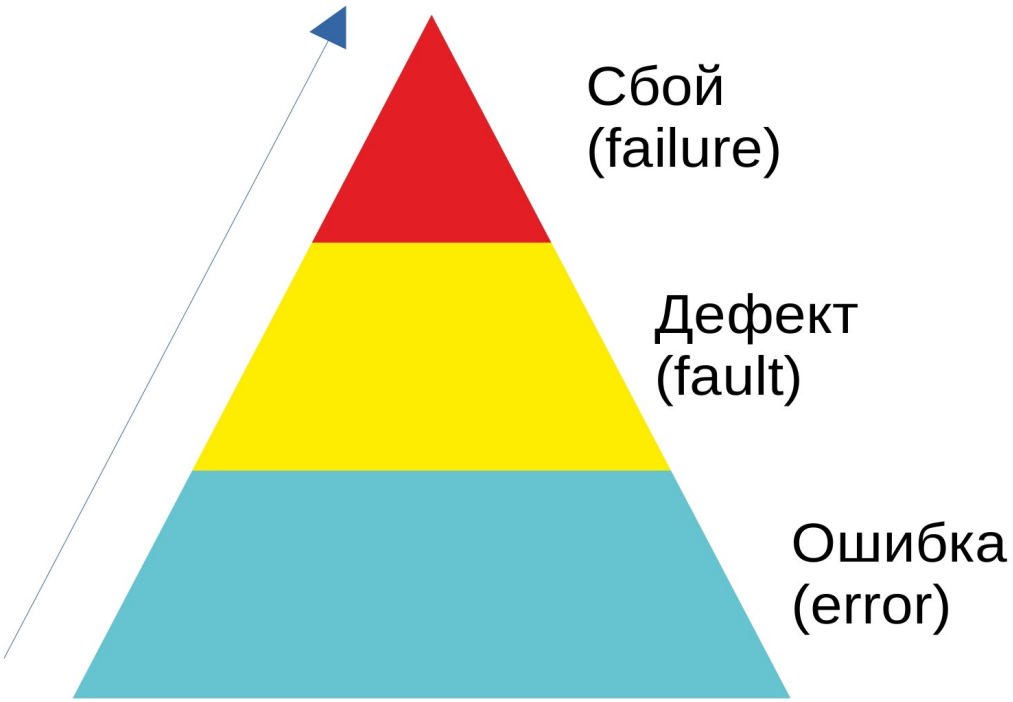
- Опечатки

```
if (i = 3) {
        //something
}

for (i = 0; i < 5; j++) {
        //something
}
```

- Ошибки в документации
- Ошибки при использовании функций
- Неверная обработка пограничных случаев
- Неверная обработка ошибок
- Добавление/удаление функционала
- ...

# Модель ошибки



Сбой
(failure)

Некорректное поведение программы, наблюдаемое «снаружи»

Дефект
(fault)

Некорректное состояние программы, которое приводит к сбою

Ошибка
(error)

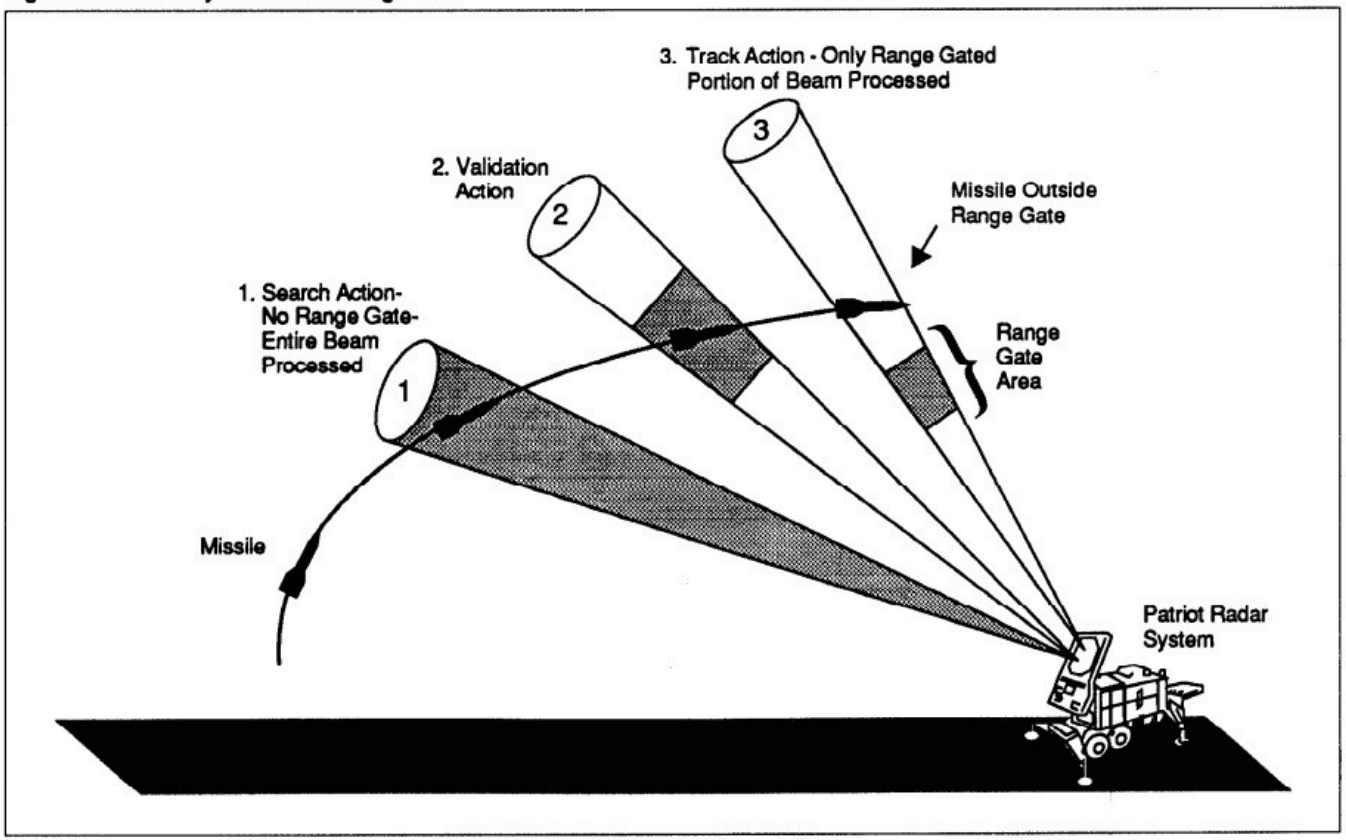
Недостаток, заложенный при разработке

# Цель тестирования

Тестирование не даёт однозначный ответ на вопрос "работает ли ПО правильно". Но может показать случаи, в которых ПО работает неправильно.

# Patriot missle floating point bug

Figure 5: Incorrectly Calculated Range Gate




Bonus: https://habr.com/en/company/pvs-studio/blog/312890/

# Therac-25

Bonus: https://habr.com/en/company/pvs-studio/blog/307788/

# Проблема полноты тестирования

- Проверить программу при всех возможных условиях работы невозможно.

  Пример:

```
int
add(int a, int b)
{
        return a + b;
}
```

  Очень затратно проверять функцию для каждого значения a и b.

# Свойства теста

Для того, чтобы найти ошибку, тест должен обеспечивать выполнение следующих пунктов:

- тест должен выполнить место в исходном коде, где присутствует программная ошибка
- при выполнении ошибки состояние программы должно испортиться с появлением дефекта (fault)
- сбой должен распространиться дальше и вызвать сбой (failure) в работе программы

# Тестирование. Схема.

# Тестирование. Уровни тестирования

# Модульное тестирование (unit testing)

Модульное тестирование – проверка корректности работы отдельных модулей программы

Модуль – это

- Набор входов
- Некий «ящик», выполняющий какие-либо преобразования
- Набор выходов

# Модульное тестирование.

**Как тестировать?**

- Подать модулю на вход какие-то данные
- Считать с выходов результат работы

**Из чего состоит?**

- Тесты (test cases)
- Наборы тестов (test suite)
- исполнитель тестов (test runner)

# Модульное тестировние. Тестовый пример (test case)

Тестовый пример состоит из

- Входных данных
- Информации о действиях
- Ожидаемых выходных данных

# Подход к тестированию

Знаем ли мы как работает то, что мы тестируем?

- **Нет** —Чёрный ящик
- **Да** —Белый ящик
- **Частично** — Серый ящик

# Python. Unittest. Объекты

- Test fixture — инициализатор начального состояния
- Test case —  тест/тестовый пример
- Test suite — набор тестов
- Test runner — «запускатор» тестов

# Python. Unittest. API

- Test fixture — TestCase.setUp TestCase.tearDown
- Test case —  TestCase
- Test suite — набор файлов с TestCase
- Test runner — TextTestRunner

# C. µnit

Минималистичный фреймворк на C (2.5k строк кода)

```c
static MunitResult
test_compare(const MunitParameter params[], void* data) {

  munit_assert_ptr_equal(data, (void*)(uintptr_t)0xdeadbeef);
  munit_assert_null(NULL);
  munit_assert_int(random_int, >=, 128);
  munit_assert_true(strstr("", "") == 0);

  return MUNIT_OK;
}


#define TEST_ITEM(func) {#func, func, NULL, NULL, MUNIT_TEST_OPTION_NONE }
static MunitTest test_suite_tests[] = {
  {
    (char*) "/example/compare",     //имя теста
    test_compare,                   //функция с тестами
    test_compare_setup,             //(опционально) указатель на функцию
                                    //инициализации
    test_compare_tear_down,         //(опционально) указатель на функцию
                                    //деинициализации
    MUNIT_TEST_OPTION_NONE,         //флаги с опциями
    NULL                            //(опционально) параметры
  },
  { (char*) "/example/rand", test_rand, NULL, NULL, MUNIT_TEST_OPTION_NONE, NULL },

  TEST_ITEM(test_foo),

  { NULL, NULL, NULL, NULL, MUNIT_TEST_OPTION_NONE, NULL }
};


static const MunitSuite test_suite = {
  test_suite_tests,     //указатель на массив с тестами
  NULL,                 //опциональный указатель на следующий MunitSuite
  1,                    //сколько раз будут запускаться каждый тест
  MUNIT_SUITE_OPTION_NONE
};

int main(int argc, char* argv[])
{
  return munit_suite_main(&test_suite, (void*) "µnit", argc, argv);
}
```

# DEMO

# The end