

Технологии разработки программного обеспечения

2020 / 2021, 1 курс, 2 семестр

Пудов Сергей Григорьевич

Лекция 7

- ▶ Тестирование приложений
 - ▶ Основы
 - ▶ Уровни зрелости
 - ▶ Типы тестирования
 - ▶ Уровни тестирования
 - ▶ Невозможно тестировать все
- ▶ Тестирование классов эквивалентности
- ▶ Тестирование граничных значений
- ▶ Ctest
- ▶ Code coverage

Основы тестирования

Lee Copeland, «A Practitioner's Guide to Software Test Design»

Тестирование программного обеспечения (Software Testing) - проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. *[IEEE Guide to Software Engineering Body of Knowledge, SWEBOOK, 2004]* В более широком смысле, **тестирование** - это одна из техник контроля качества, включающая в себя активности по планированию работ (**Test Management**), проектированию тестов (**Test Design**), выполнению тестирования (**Test Execution**) и анализу полученных результатов (**Test Analysis**).

Уровни зрелости

Пять уровней зрелости (Software Testing Maturity Model):

▶ **1. хаотический**

Процесс тестирования программного обеспечения имеет хаотический характер, что отличает большинство начинающих компаний. Процесс тестирования не определен как выделенная активность и не отделен от процесса отладки кода. Тестирование выполняется по факту создания кода и построения или сборки системы. Цель тестирования — показать, что приложение работает.

▶ **2. фаза разработки**

Тестирование программного обеспечения отделено от кодирования и выделяется как следующая фаза. Главная цель тестирования — показать, что приложение соответствует требованиям. Имеются базовые подходы и практики тестирования.

▶ **3. интегрированный**

Процесс тестирования интегрирован в жизненный цикл разработки программного обеспечения. Цели тестирования базируются на требованиях. Имеется организация тестирования, а само тестирование выделено в профессиональную деятельность.

Уровни зрелости

► 4. *управление и измерение*

Тестирование является измеряемым и контролируемым процессом. Процессы критических *осмотров* (review) проектных артефактов (тестовые планы и сценарии, сообщения об ошибках, итоговые отчеты о состоянии версии и т.д.) относятся к тестовым активностям. Продукт тестируется на соответствие таким качественным метрикам, как надежность, удобство, сопровождаемость.

► 5. *оптимизация процесса, предотвращение ошибок и контроль качества*

Тестирование является определенным и управляемым процессом. Стоимость тестирования наравне с показателями эффективности может быть определена. Тестирование как процесс поддается изменениям, которые однозначно положительно на него влияют. Внедрены и используются практики предотвращения ошибок и контроля качества. Автоматизированное тестирование применяется как основной подход в тестировании. Проектирование тестов, анализ полученных результатов, обработка описаний ошибок, а также метрик, связанных с тестированием, осуществляется при помощи соответствующих инструментальных средств.

Типы тестирования

▶ **Black box**

Стратегия, основанная на требованиях и спецификациях. Не требует знания реализации.

▶ **White box**

Стратегия, основанная на знании реализации тестируемого продукта. Требует навыков программирования.

Уровни тестирования

- ▶ Модульное тестирование (Unit testing). Unit – «наименьший» фрагмент создаваемого кода. C++ / Java – class, C – функция
- ▶ Интеграционное тестирование – тестирование подсистемы или целой системы. Выявляет ошибки связей между частями кода.
- ▶ Системное тестирование
 - ▶ Функциональное тестирование
 - ▶ Юзабилити тестирование
 - ▶ Тестирование безопасности
 - ▶ Тестирование локализации
 - ▶ Тестирование производительности
 - ▶ ...
- ▶ Приемочное тестирование

Невозможно тестировать все!

```
int blench( int j)
{
    j -= 1;    // Error! Should be      j += 1;
    j /= 30000;
    return j;
}
int: -32768 <= j <= 32767
```

Только 4 входных значения из 65536 позволят найти ошибку.

Классы эквивалентности

- ▶ Пример:

- ▶ 0 – 16 не нанимать
- ▶ 16 – 18 частичная занятость
- ▶ 18 – 55 полная занятость
- ▶ 55 – 99 не нанимать

- ▶ Полное покрытие: тестовые данные от 0 до 100

- ▶ Покрытие классов эквивалентности: 4 теста

- ▶ 10 - не нанимать
- ▶ 17 – частичная занятость
- ▶ 40 – полный рабочий день
- ▶ 80 – не нанимать

Классы эквивалентности

Ожидания:

- ▶ Если один набор данных из класса эквивалентности обнаруживает ошибку, то и все остальные наборы данных из этого же класса эквивалентности приведут к ошибке
- ▶ Если один набор данных из класса эквивалентности НЕ обнаруживает ошибку, то ни один другой набор данных из этого же класса эквивалентности вероятно не приведет к ошибке.

Резюме:

- ▶ Тестирование классов эквивалентности – техника, использующаяся для сокращения количества тестов до управляемого числа с сохранением разумного покрытия функционала тестами.

Граничные значения

16 лет – что является правильным ответом ?

▶ Пример:

- ▶ 0 – 15 не нанимать
- ▶ 16 – 17 частичная занятость
- ▶ 18 – 54 полная занятость
- ▶ 55 – 99 не нанимать

Наборы данных:

$\{-1, 0, 1\}$

$\{14, 15, 16\}$

$\{17, 18, 19\}$

$\{54, 55, 56\}$

$\{98, 99, 100\}$

Библиотека Ctest

```
#include <sum.h>
```

```
#include <ctest.h>
```

```
CTEST(arithmetic_suite, simle_sum)
```

```
{
```

```
    // Given
```

```
    const int a = 1;
```

```
    const int b = 2;
```

```
    //When
```

```
    const int result = sum(a, b);
```

```
    //Then
```

```
    const int expected = 3;
```

```
    ASSERT_EQUAL(expected, result);
```

```
}
```

Библиотека Ctest

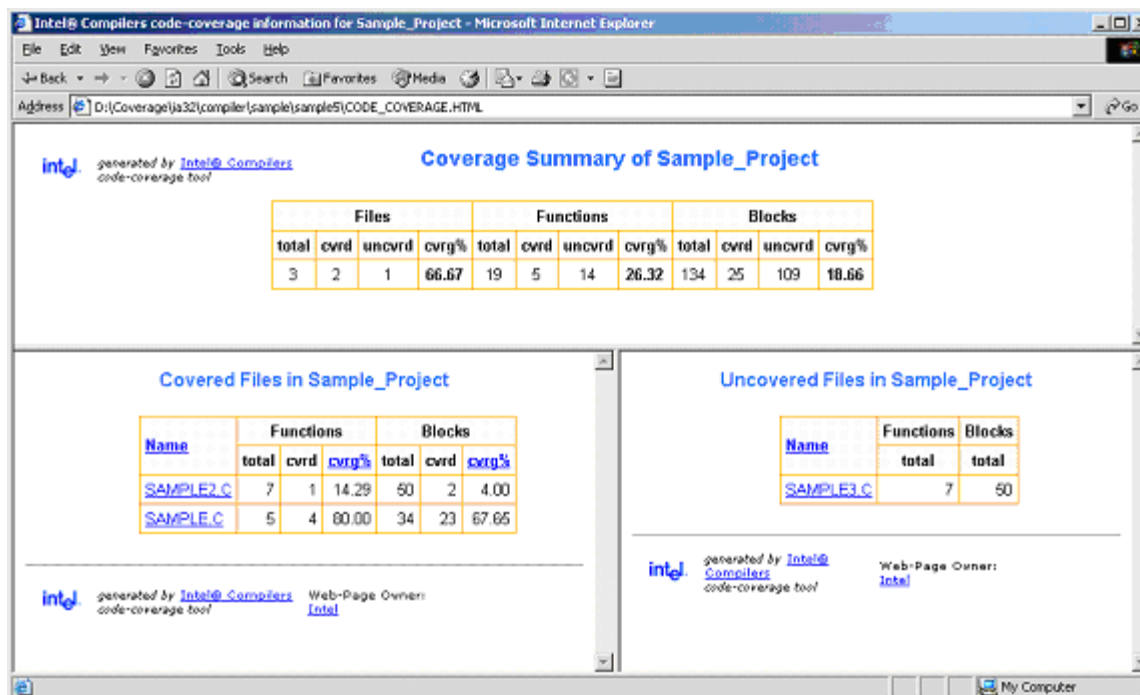
<https://github.com/bvdberg/ctest>

```
$ make test
make[1]: Nothing to be done for `all'.
TEST 1/27 suite1:test1 [OK]
TEST 2/27 suite1:test2 [FAIL]
  ERR: mytests.c:12 expected 1, got 2
TEST 3/27 suite2:test1 [FAIL]
  ERR: mytests.c:16 expected 'foo', got 'bar'
TEST 4/27 suite3:test3 [OK]
TEST 5/27 memtest:test1 [OK]
  LOG: memtest_setup() data=0x107f488a0 buffer=0x0
  LOG: __ctest_memtest_test1_run() data=0x107f488a0 buffer=0x7fed79800000
  LOG: memtest_tearardown() data=0x107f488a0 buffer=0x7fed79800000
TEST 6/27 memtest:test3 [SKIPPED]
TEST 7/27 memtest:test2 [FAIL]
  LOG: memtest_setup() data=0x107f488a0 buffer=0x7fed79800000
  LOG: __ctest_memtest_test2_run() data=0x107f488a0 buffer=0x7fed79800000
  ERR: mytests.c:53 shouldn't come here
TEST 8/27 fail:test1 [FAIL]
  ERR: mytests.c:61 shouldn't come here
TEST 9/27 weaklinkage:test1 [OK]
  LOG: __ctest_weaklinkage_test1_run()
TEST 10/27 weaklinkage:test2 [OK]
  LOG: __ctest_weaklinkage_test2_run()
TEST 11/27 nosetup:test1 [OK]
  LOG: __ctest_nosetup_test1_run()
  LOG: nosetup_tearardown()
TEST 12/27 ctest:test_assert_str [FAIL]
  ERR: mytests.c:98 expected 'foo', got 'bar'
TEST 13/27 ctest:test_assert_equal [FAIL]
  ERR: mytests.c:103 expected 123, got 456
TEST 14/27 ctest:test_assert_not_equal [FAIL]
  ERR: mytests.c:108 should not be 123
TEST 15/27 ctest:test_assert_null [FAIL]
  ERR: mytests.c:114 should be NULL
TEST 16/27 ctest:test_assert_not_null_const [OK]
TEST 17/27 ctest:test_assert_not_null [FAIL]
```

Поккрытие кода

- ▶ Поккрытие кода — мера, используемая при тестировании программного обеспечения. Она показывает процент, насколько исходный код программы был протестирован.
- ▶ Существует несколько различных способов измерения поккрытия, основные из них:
 - ▶ *поккрытие операторов* — каждая ли строка исходного кода была выполнена и протестирована;
 - ▶ *поккрытие условий* — каждая ли точка решения (вычисления истинно ли или ложно выражение) была выполнена и протестирована;
 - ▶ *поккрытие путей* — все ли возможные пути через заданную часть кода были выполнены и протестированы;
 - ▶ *поккрытие функций* — каждая ли функция программы была выполнена;
 - ▶ *поккрытие вход/выход* — все ли вызовы функций и возвраты из них были выполнены.
 - ▶ *поккрытие значений параметров* — все ли типовые и граничные значения параметров были проверены.

Покрытие кода (Intel compiler)



Продолжение в следующей лекции...

