Тестирование и отладка ПС (часть 2)

Отвагин Алексей Владимирович,
доцент каф. ЭВМ, к.т.н., а. 505-5

Содержание

- Интеграционное тестирование
- Автоматизация тестирования

Характеристика интеграционного тестирования

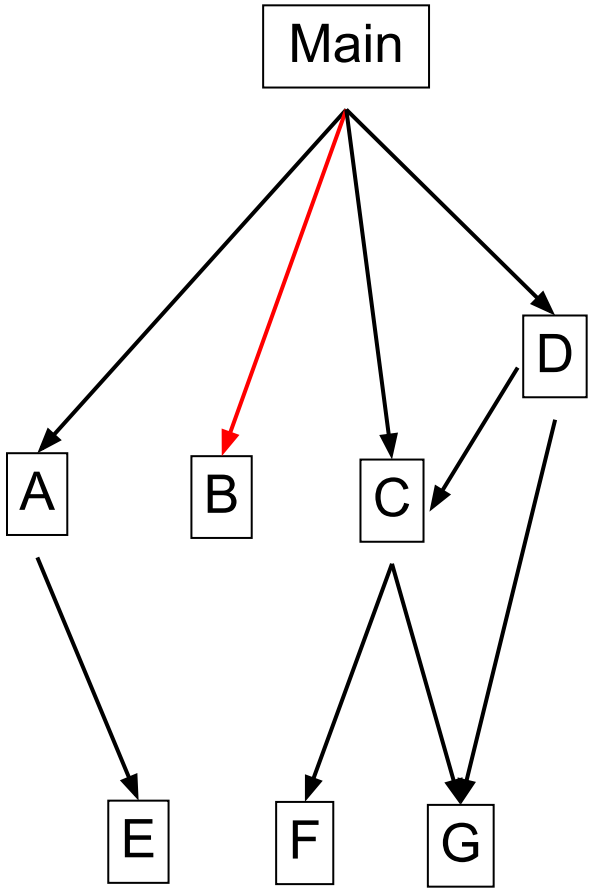
Действия: проверка взаимодействия модулей посредством интерфейсов

Цель: выявление ошибок в интерфейсе или кооперации модулей

Условие: модули уже протестированы

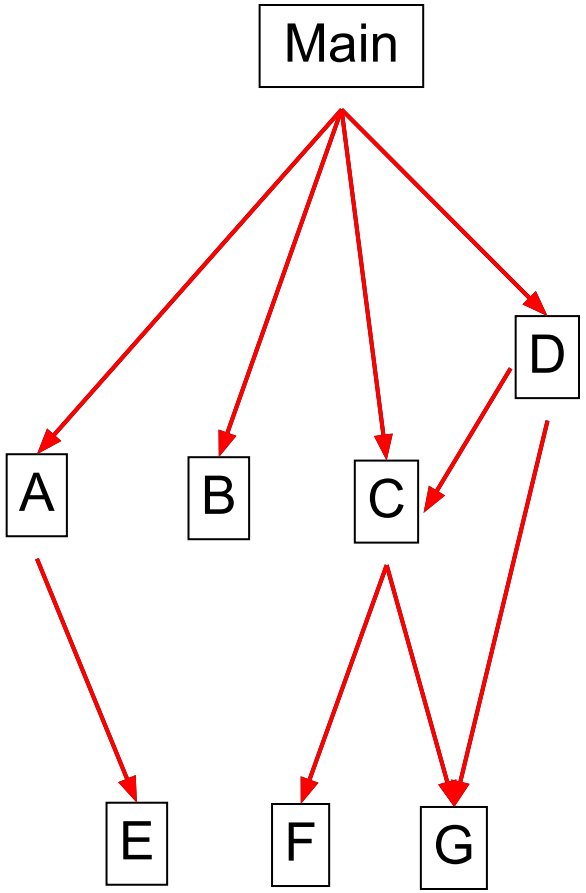
Схема интеграционного тестирования

- ПС представлено композицией модулей
- Модули уже протестированы автономно
- Тестируются интерфейсы
  - Системное поведение не оценивается
  - Создается граф вызовов процедур
- Для тестирования модуля P необходимо
  - Все субмодули (потомки) реализованы (реальный код или имитация)
  - Реализован один или все родители модуля (реальный код или драйвер)



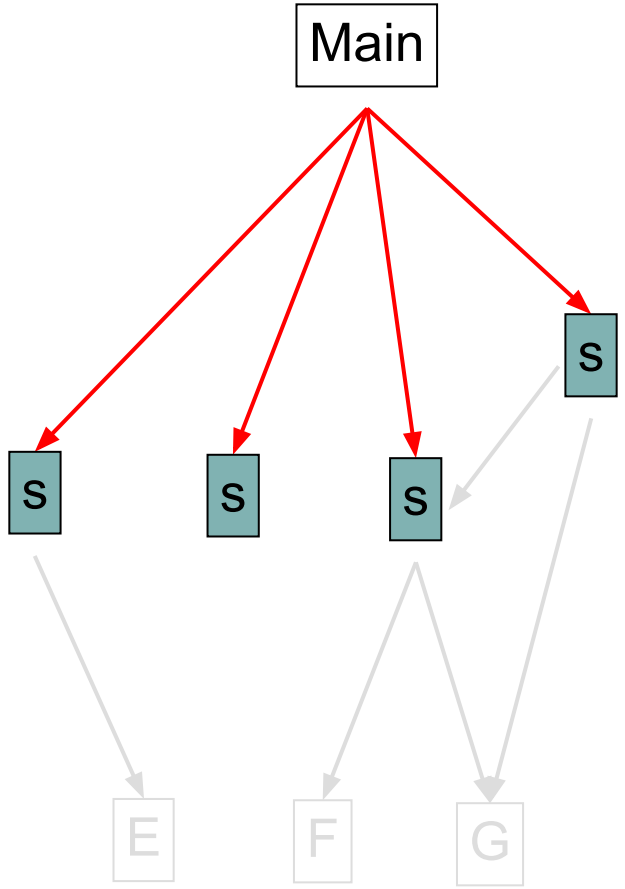
Метод большого взрыва

- Сразу собрать все модули в одну программу
- Надежда на результаты автономных тестов
- Требуется небольшое количество тестов
- Тяжело локализовать ошибки



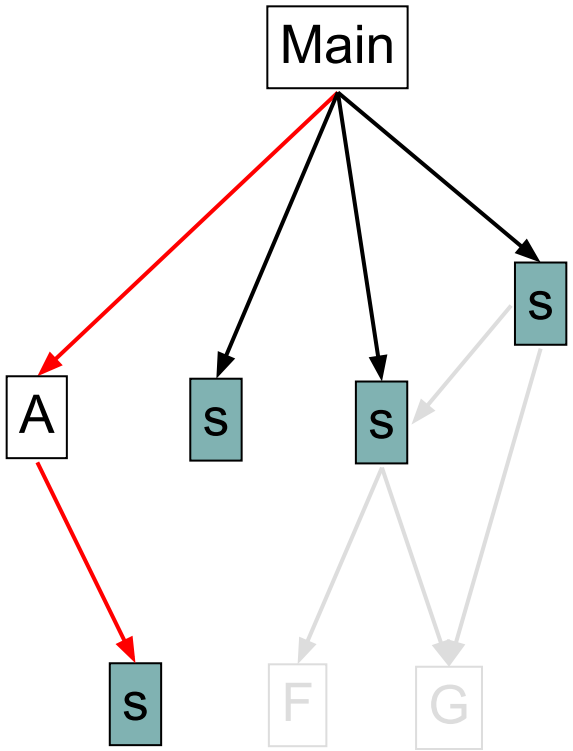
Нисходящий метод

- Начинаем с основного модуля: Main
- Все модули, к которым обращается Main, заменены заглушкой:
  - Простая заглушка
  - Имитатор поведения
  - Легко программируется
    - Для проверки взаимодействия: фиктивный возврат
    - Проверка вычислений: имитатор функции



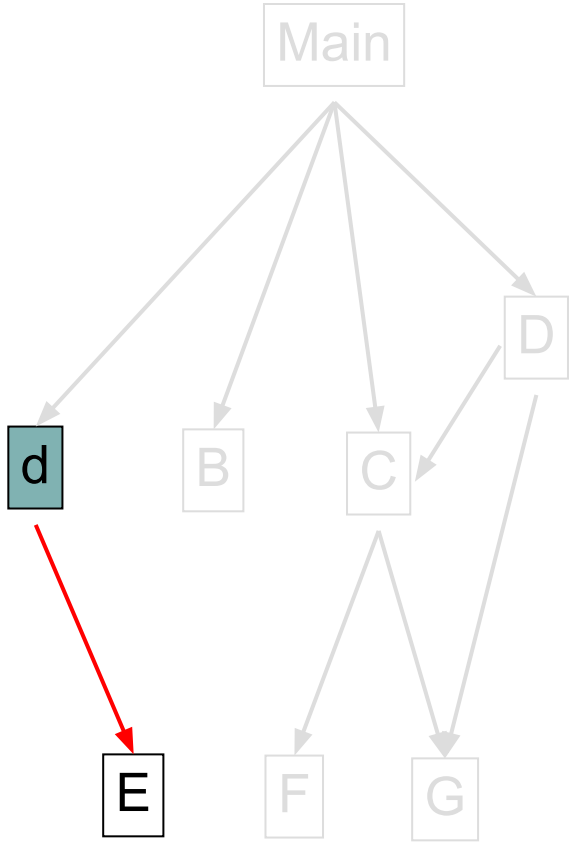
Нисходящий метод (продолжение)

- Следующая итерация: модуль A
- Требуются новые заглушки (количество и качество заглушек могут меняться)
- На каждый модуль требуется новый сеанс тестирования
- Легко локализовать ошибки



Восходящий метод

- Начинаем с модуля нижнего уровня: E
- Заменяем вызывающий модуль драйвером:
  - Имитирует вызовы реального модуля
  - Сложно программировать:
    - Ограничения по времени
    - Трудности реализации сложных стратегий
- Требуется много драйверов
  - Драйвер может изменять поведение
- Требуется много сеансов тестирования
- Легко локализовать ошибки



Сэндвич

- Смесь восходящего и нисходящего подходов
- Требует заглушек и драйверов
- Тестируем интерфейсы между уже протестированными участками или подграфами
- Количество сеансов тестирования среднее
- Труднее локализовать ошибки

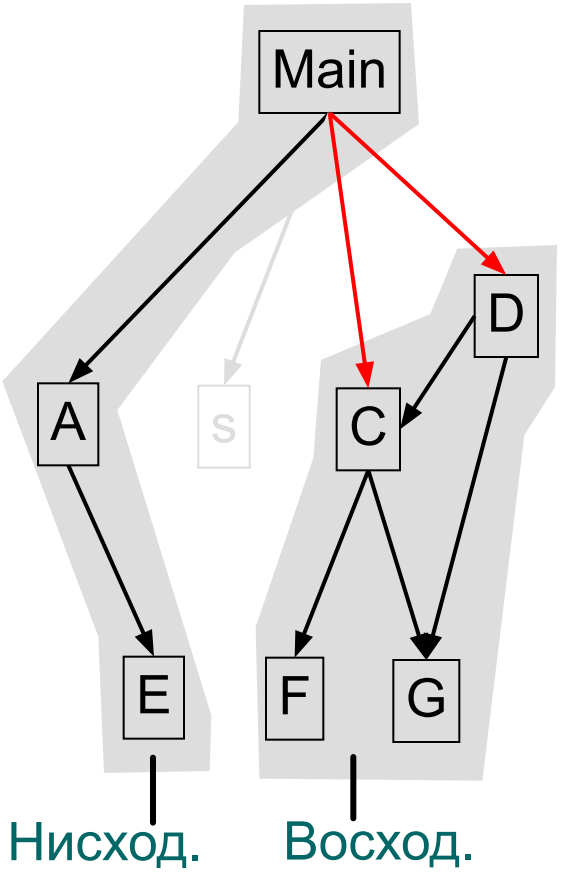
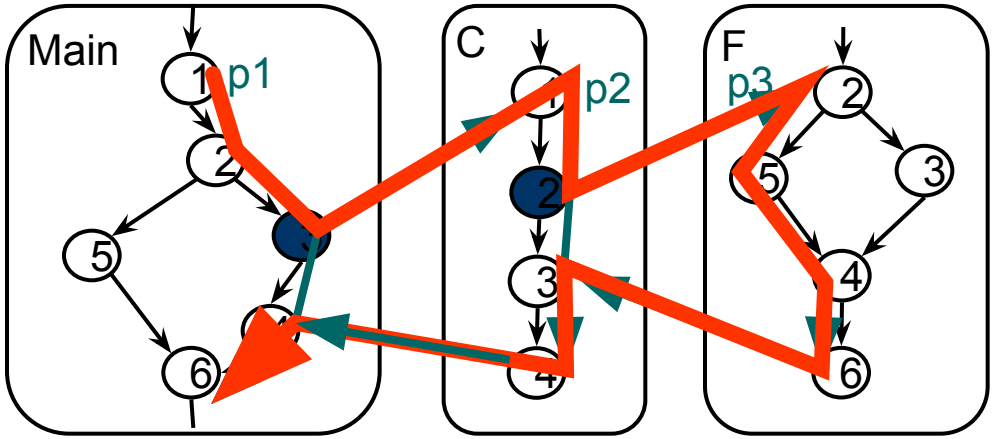
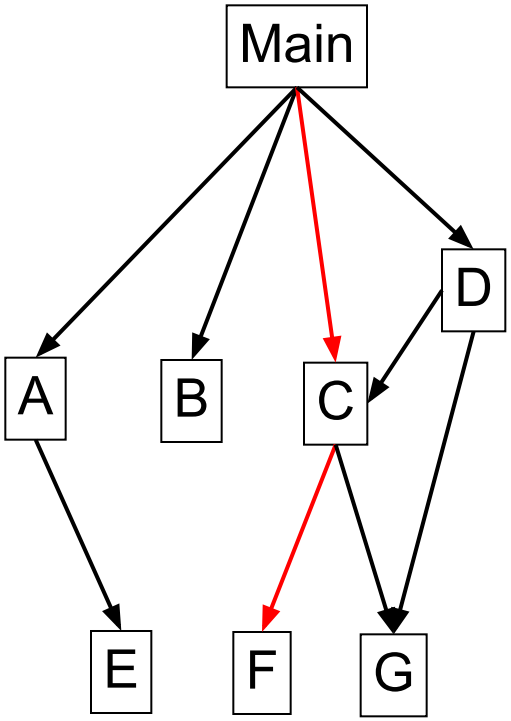


Схема интеграционного теста

- Входные значения/действия + настройка драйвера/заглушки/модуля + ожидаемый вывод (вызовы, действия, результаты)
- В созданном графе структуры программы:
  Путь между модулями: p1/p2/p3
- Структура вызовов и возвратов:
  Main < C < F > C > Main





Автоматизация тестирования

- Подходы к тестированию могут быть обобщены
- Развитие open-source привело к созданию множества доступных средств автоматизации
- Созданы среды для многих популярных языков JUnit, Cunit и др.

Модульное тестирование (unit testing)

- Модуль – наименьшая единица кода, разрабатываемая одним программистом
- Модульные тесты пишутся на том же языке, что и тестируемый код
- Модульные тесты пишутся теми же самыми программистами, которые написали код
- Среда тестирования собирает тесты в наборы и позволяет их пакетное выполнение

Неисправности и ошибки

- Неисправность (failure) – наличие в коде устранимой погрешности, выявленной assert()
- Ошибка (error) – наличие в коде погрешности, вызывающей системную ошибку или исключение

Основы JUnit

- Класс, который тестируется – класс продукта
- Класс, который тестирует – класс теста
- Каждому классу продукта соответствует собственный класс теста
- Классы тестов объединяются в набор (suite)

Пример Largest

- Пример определяет наибольшее число в списке значений
- Например, [7, 8, 9] ☐ 9

```
public class Largest {
/**
 * Return the largest element in a list.
 *
 * @param list A list of integers
 * @return The largest number in the given list
 */
public static int Largest(int[] list)
{
        int index, max=Integer.MAX_VALUE;
        for (index = 0; index < list.length-1; index++) {
                if (list[index] > max) {
                        max = list[index];
                }
        }
        return max;
}
}
```

Тест для Largest

```
import junit.framework.*;
public class TestLargest extends TestCase {
    public TestLargest(String name) { super(name);
    }
public void testSimple() {
    assertEquals(9, Largest.largest(new int[] {7,8,9}));
    }
}
```

Интерфейс JUnit

- Существует три пользовательских интерфейса для JUnit TestRunner:
  - *TextUI*: Обеспечивает текстовый вывод в stdout.
  - *AwtUI*: Обеспечивает вывод на основе графического интерфейса пользователя, используя AWT из Java.
  - *SwingUI*: Обеспечивает вывод на основе графического интерфейса пользователя, используя комплект компонентов графического интерфейса пользователя Swing из Java.
- **java junit.*USERINTERFACE*.TextRunner *classfile***

Вывод для Largest

There was 1 failure:

1) testSimple(TestLargest)junit.framework.AssertionFailedError:

expected:<9> but was:<2147483647>

at TestLargest.testSimple(TestLargest.java:11)

- Причина:

```
int index, max=Integer.MAX_VALUE;
```

Тест порядка в списке Largest

```
import junit.framework.*;
public class TestLargest extends TestCase {
    public TestLargest(String name) {    super(name);
    }
public void testSimple() {
    assertEquals(9, Largest.largest(new int[] {7,8,9}));
    }
}
public void testOrder() {
    assertEquals(9, Largest.largest(new int[] {9,8,7}));
    assertEquals(9, Largest.largest(new int[] {7,9,8}));
    assertEquals(9, Largest.largest(new int[] {7,8,9}));
}
```

Вывод для Largest

There was 1 failure:

1) testOrder(TestLargest)junit.framework.AssertionFailedError:

expected:<9> but was:<8>

at TestLargest.testOrder(TestLargest.java:10))

- Причина:

```
for (index = 0; index < list.length-1; index++) {
```

Тесты разных списков Largest

```
public void testDups() {
assertEquals(9, Largest.largest(new int[] {9,7,9,8}));
}

public void testOne() {
assertEquals(1, Largest.largest(new int[] {1}));
}

public void testNegative() {
int [] negList = new int[] {-9, -8, -7};
assertEquals(-7, Largest.largest(negList));
}
```

Вывод для Largest

There was 1 failure:

1) testNegative(TestLargest)junit.framework.AssertionFailedError:

expected:<-7> but was:<0>

at TestLargest.testNegative(TestLargest.java:16)

- Причина:

```
int index, max=0;
```

Набор тестов Largest

```
import junit.framework.*;
public class TestLargest extends TestCase {
        public TestLargest(String name) {
        super(name);
        }

protected void setUp() {
}

protected void tearDown() {
}

public static Test suite() {
        TestSuite suite = new TestSuite();
        suite.addTest(new TestLargest ("testSimple"));
        suite.addTest(new TestLargest ("testOrder"));
        suite.addTest(new TestLargest ("testDups"));
        suite.addTest(new TestLargest ("testOne"));
        suite.addTest(new TestLargest ("testNegative"));
        }
}
```

Содержание тестов

- Каждый тест содержит совокупность методов assert(), определенных в классе Assert
- Методы assert() полиморфны и определяют **атомы** тестирования

Виды assert
assertEquals([String], expected, actual)
assertNull([String], Object)
assertSame([String], expected, actual)
assertTrue([String], boolean)
fail([String])

Пример BinString

- Пример вычисляет сумму кодов символов в строке и возвращает ее двоичное представление в виде строки
- Например, “” = 0 = “0”

  “d” = 100 = “1100100”

  “Add” = 265 = “100001001”

Исходный текст программы

```
public class BinString {
    public BinString() {}
    public String convert(String s) {
    return binarise(sum(s));
    }
    public int sum(String s) {
    if(s=="") return 0;
    if(s.length()==1) return ((int)(s.charAt(0)));
    return ((int)(s.charAt(0)))+sum(s.substring(1));
    }
    public String binarise(int x) {
    if(x==0) return "";
    if(x%2==1) return "1"+binarise(x/2);
    return "0"+binarise(x/2);
    }}
```

Таблица соответствия функций

BinString.java	BinStringTest**.java**
BinString()	BinStringTest()
sum()	testSum()
binarize()	testBinarize()
convert()	testConvert()

Тесты для примера BinString

```
import junit.framework.*;
     public class BinStringTest extends TestCase {
     private BinString binString;

     public BinStringTest(String name) {
     super(name);
     }

     protected void setUp() {
     binString = new BinString();
     }

     public void testSum () {
     int expected = 0;
     assertEquals(expected, binString.sum(""));
     expected = 100;
     assertEquals(expected, binString.sum("d"));
     expected = 265;
     assertEquals(expected, binString.sum("Add"));
     }
```

Тесты для примера BinString (2)

```
    public void testBinarise () {
    String expected = "101";
    assertEquals(expected, binString.binarise(5));
    expected = "11111100";
    assertEquals(expected, binString.binarise(252));
    }
    public void testConvert() {
    String expected = "1000001";
    assertEquals(expected, binString.convert("A"));
    }
}
```