



KIỂM THỬ VÀ ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

INT3117

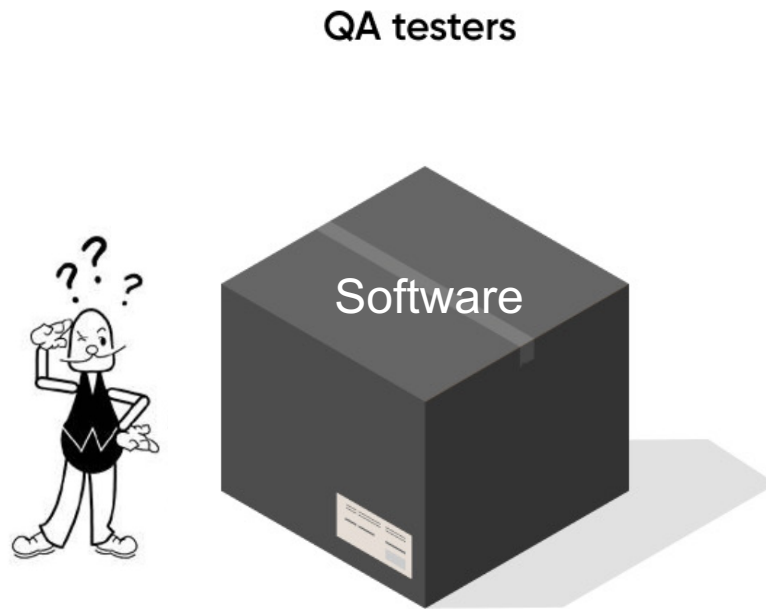
Bài giảng 02: Khái niệm cơ bản & JUnit

- Kiểm thử đơn vị là gì?
- JUnit là gì và tại sao lại sử dụng JUnit?

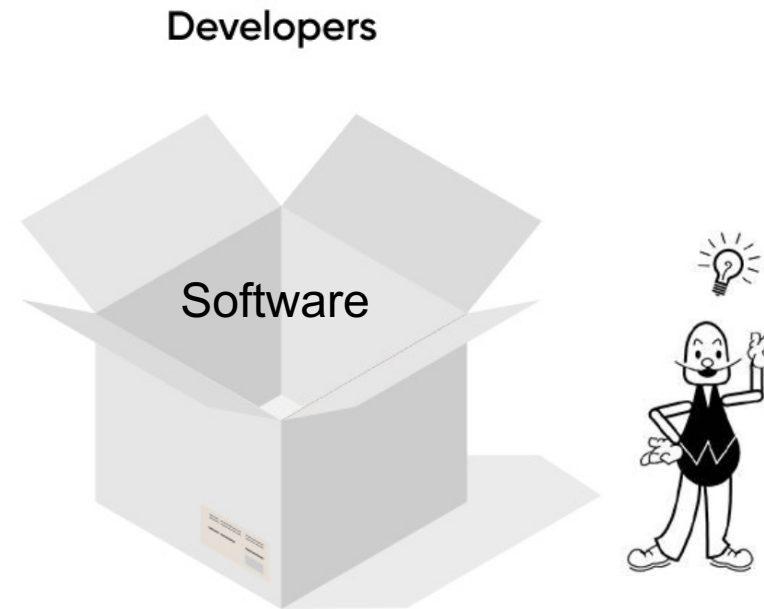
Hôm nay...

- Các khái niệm cơ bản trong kiểm thử
- Công cụ kiểm thử JUnit

KIỂM THỬ HỘP ĐEN vs. KIỂM THỬ HỘP TRẮNG



Black box - we do not
know anything



White box - we know
everything

KIỂM THỬ HỘP ĐEN vs. KIỂM THỬ HỘP TRẮNG

Hộp đen (Black-box/Chức năng/Dựa trên đặc tả)

Hộp trắng (White-box/Dựa trên code/Cấu trúc)

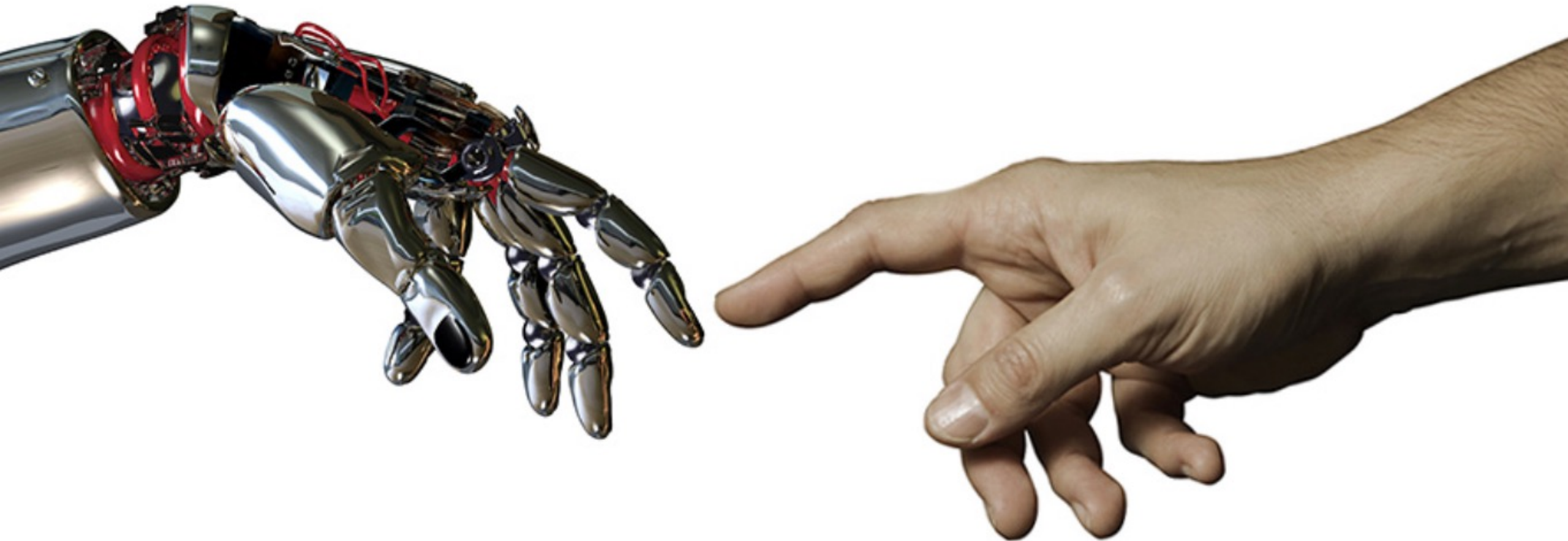
- Dựa trên yêu cầu chức năng
- Định hướng đầu vào/đầu ra
- Yếu tố nội tại của mã nguồn không được dùng để thiết kế việc kiểm thử/đánh giá chất lượng phần mềm

- Dựa trên cấu trúc của mã nguồn
- Kiểm thử/đánh giá chất lượng của phần mềm ở khía cạnh độ phủ của mã nguồn

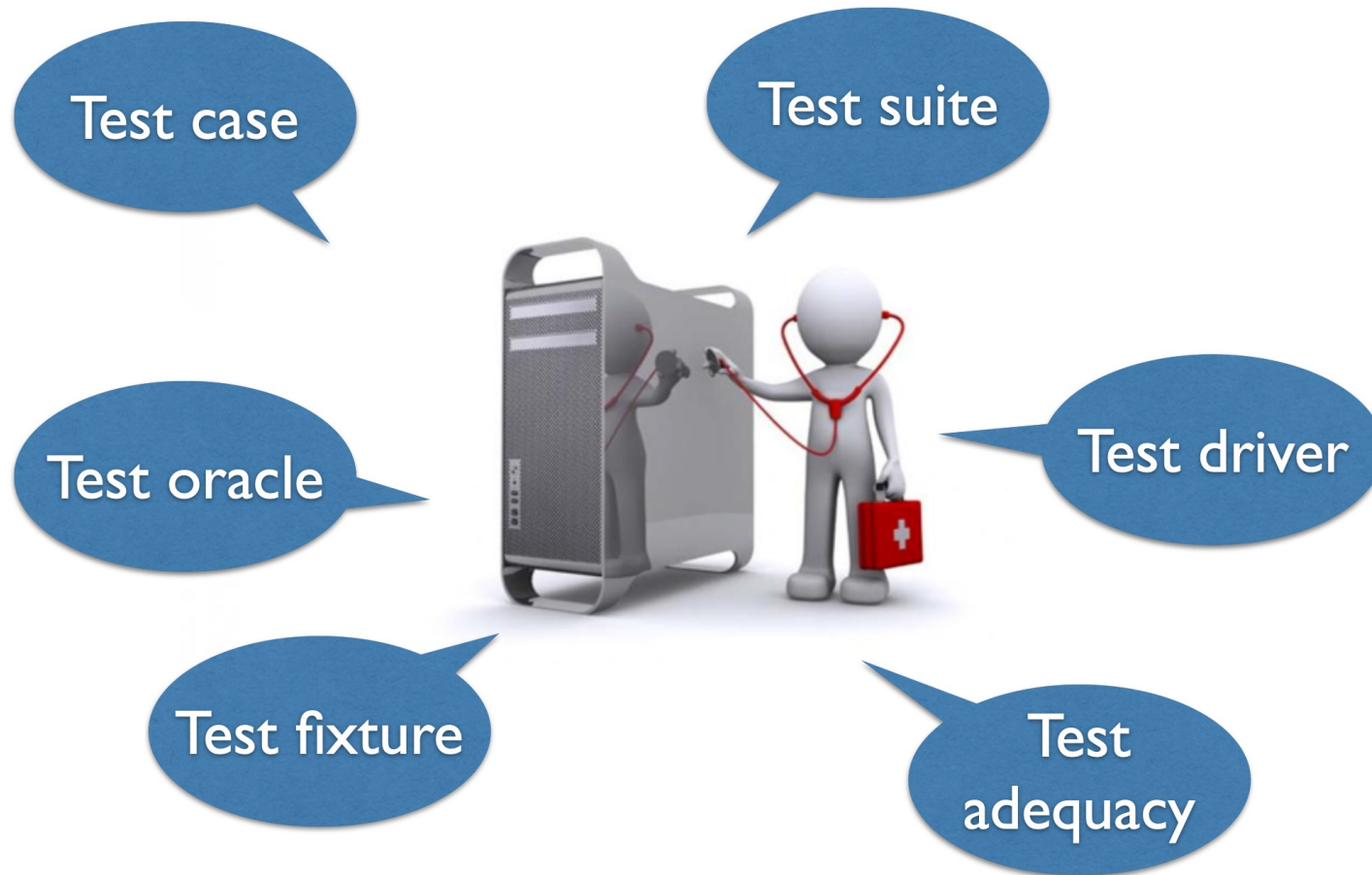


Nhiều kỹ thuật “lai” giữa Black- và White-box (Gray box)

Kiểm thử thủ công và Kiểm thử tự động



Kiểm thử: Các khái niệm



Kiểm thử: Các khái niệm

- **Cá kiểm thử (test case, test):** Một lần thực thi của *phần mềm* với đầu vào cho trước:
 - Giá trị đầu vào (inputs)
 - Giá trị đầu ra kỳ vọng (expected outputs)
 - Đôi khi, bao gồm cả các bước thực thi

```
int actual_output=sum(1,2);  
assertTrue(actual_output==3);
```


Kiểm thử: Các khái niệm

- **Test oracle:** Đầu ra kì vọng của phần mềm cho một đầu vào nhất định
 - Là một phần của các test cases
 - Là vấn đề khó nhất trong tự động kiểm thử

```
int actual_output=sum(1,2);  
assertTrue(actual_output==3);
```

Kiểm thử: Các khái niệm

- **Bối cảnh kiểm thử (test context/fixture):** Trạng thái của phần mềm – nền tảng cho việc chạy các ca kiểm thử
 - Load database với các dữ liệu cụ thể (đã biết)
 - Chuẩn bị input và cài đặt hoặc tạo ra các dữ liệu giả (mock objects)

Kiểm thử: Các khái niệm

- **Test suite:** Một tập các ca kiểm thử
 - Thường có chung thiết lập và các điều kiện tiên quyết
 - Thường chạy cùng nhau
 - Thông thường, các test suites khác nhau cho các mục đích khác nhau, VD: nền tảng, chức năng, hiệu năng

Kiểm thử: Các khái niệm

- **Test driver:** Framework dùng để load các tests hoặc một test suite
 - Xử lý các thiết lập, so sánh giữa đầu ra kì vọng và đầu ra thực tế

Kiểm thử: Các khái niệm

- **Test adequacy:**

- Chúng ta không thể luôn dùng tất cả test inputs, vậy chúng ta sử dụng những inputs nào và khi nào thì dừng lại?
- Chúng ta cần 1 chiến lược để xác định khi nào thì chúng ta test đủ nhiều

Kiểm thử: Các khái niệm

- **Test adequacy:** Một luật để đánh giá tính đầy đủ của một bộ test cho một phần hoặc cả phần mềm
 - Độ phủ kiểm thử: phủ hết các dòng lệnh, phủ hết nhánh, ...

All files

80.49% Statements 524/651 55.75% Branches 97/174 67.76% Functions 124/183 79.02% Lines 437/553

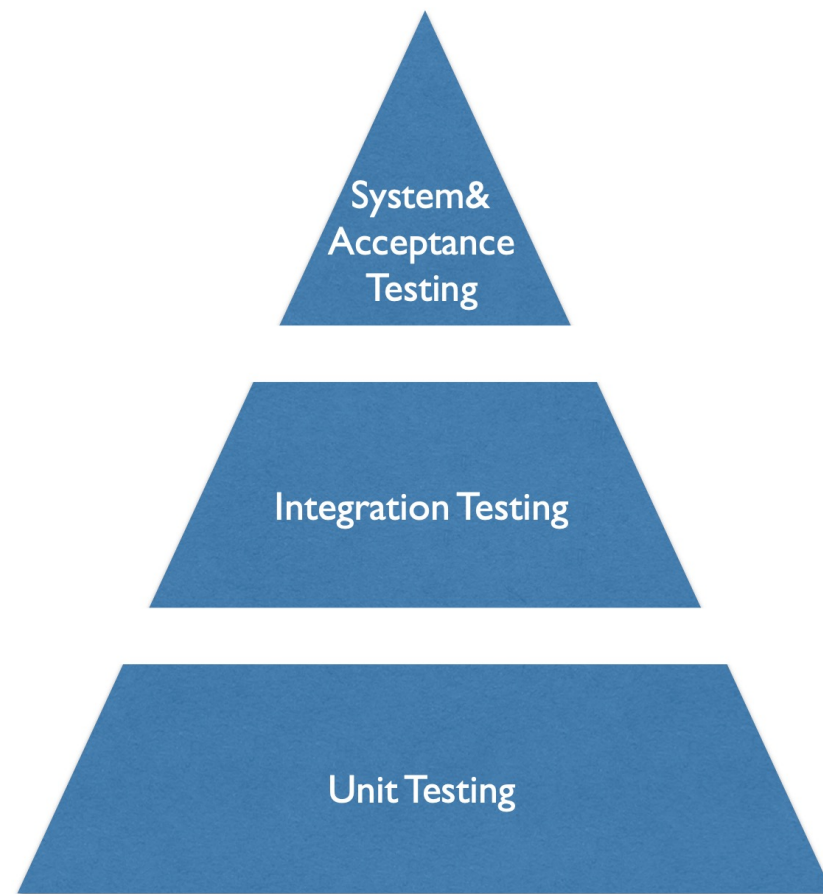
Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
src	<div></div>	100%	15/15	100%	0/0	100%	1/1	100%	15/15
src/app	<div></div>	100%	36/36	100%	0/0	100%	8/8	100%	27/27
src/app/auth	<div></div>	60.8%	76/125	25%	9/36	37.14%	13/35	58.77%	67/114
src/app/common	<div></div>	79.55%	70/88	26.92%	7/26	66.67%	12/18	78.21%	61/78
src/app/home	<div></div>	100%	14/14	100%	0/0	100%	5/5	100%	11/11
src/app/inventory	<div></div>	100%	6/6	100%	0/0	100%	3/3	100%	4/4
src/app/inventory/categories	<div></div>	100%	6/6	100%	0/0	100%	3/3	100%	4/4
src/app/inventory/inventory-home	<div></div>	100%	6/6	100%	0/0	100%	3/3	100%	4/4
src/app/user/profile	<div></div>	87.69%	57/65	93.1%	54/58	65%	13/20	88.33%	53/60
src/app/user/user	<div></div>	71.6%	58/81	55.88%	19/34	50%	11/22	74.63%	50/67
src/app/user/view-user	<div></div>	100%	18/18	83.33%	5/6	100%	4/4	100%	16/16
src/environments	<div></div>	100%	1/1	100%	0/0	100%	0/0	100%	1/1

Code coverage generated by Istanbul at Sat May 05 2018 17:05:21 GMT-0400 (EDT)

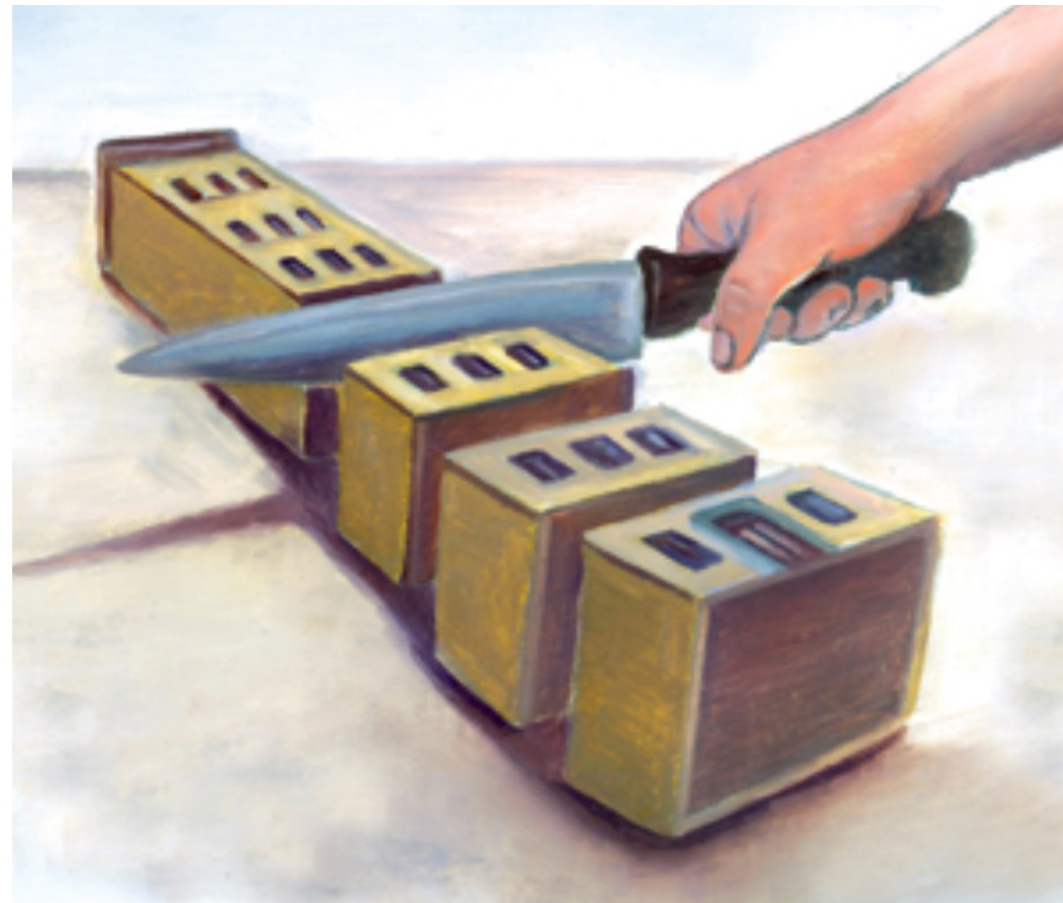
Độ mịn trong kiểm thử

- Kiểm thử đơn vị (Unit testing)
 - Kiểm thử từng module (function/method)
- Kiểm thử tích hợp (Integration testing)
 - Kiểm thử tương tác giữa các modules
- Kiểm thử hệ thống (System testing)
 - Thử toàn bộ hệ thống, được thực hiện bởi developers
- Kiểm thử chấp nhận (Acceptance testing)
 - Xác thực hệ thống với yêu cầu của khách hàng, thực hiện bởi khách hàng, không có test case một cách chính quy.

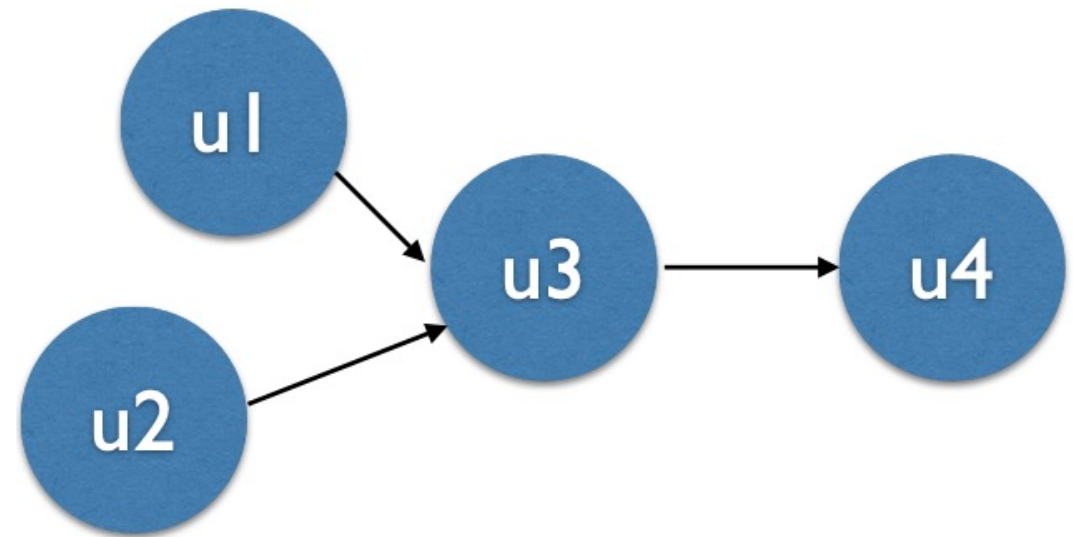


Kiểm thử đơn vị (Unit testing)

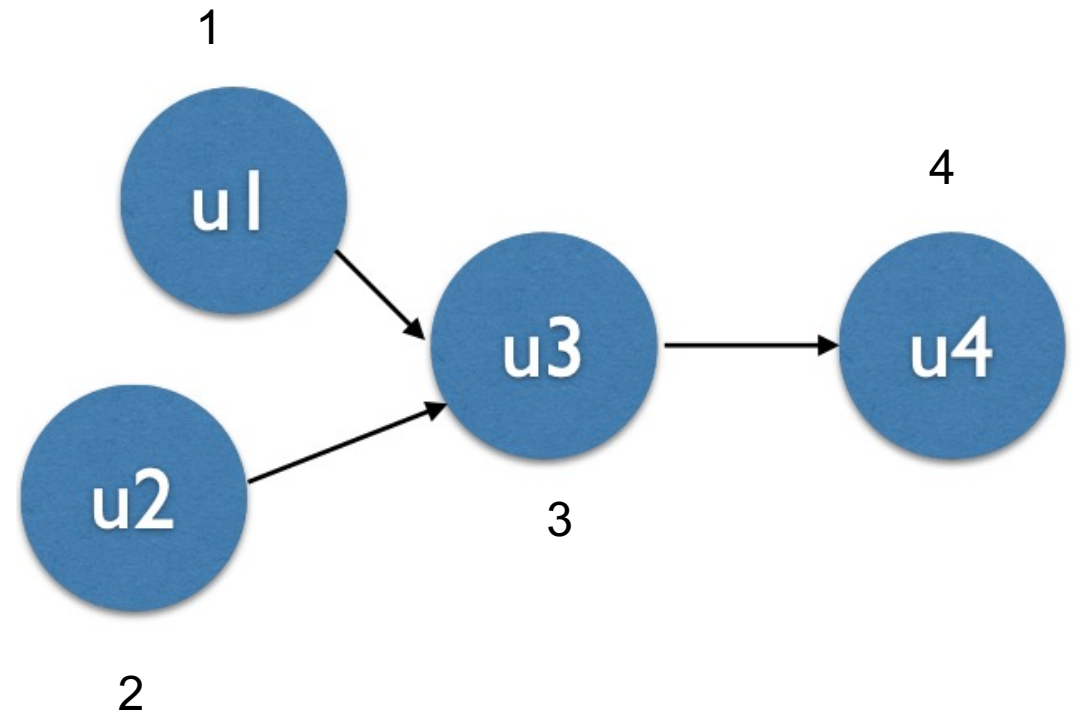
- Các vấn đề điển hình:
 - Thuật toán
 - Cấu trúc dữ liệu cục bộ
 - Điều kiện biên
 - Xử lý lỗi
- Tại sao? Chia để trị
 - Chia hệ thống thành các đơn vị
 - Debug từng đơn vị nhỏ
 - Thu hẹp phạm vi tìm lỗi



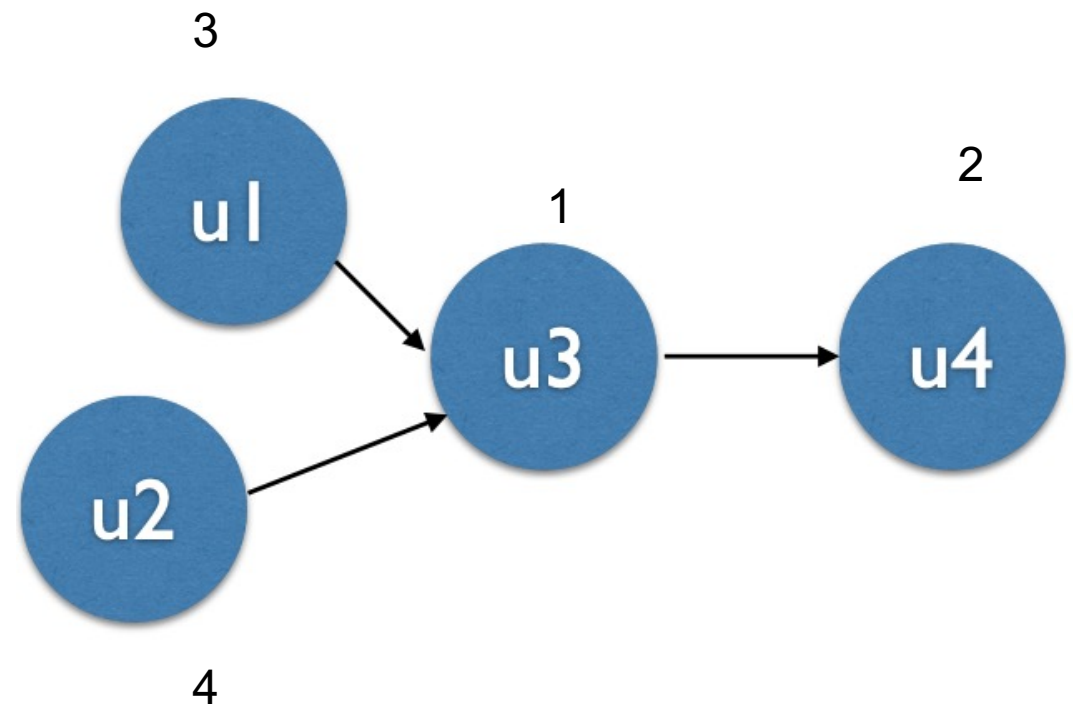
Kiểm thử đơn vị như nào?



Kiểm thử đơn vị như nào?



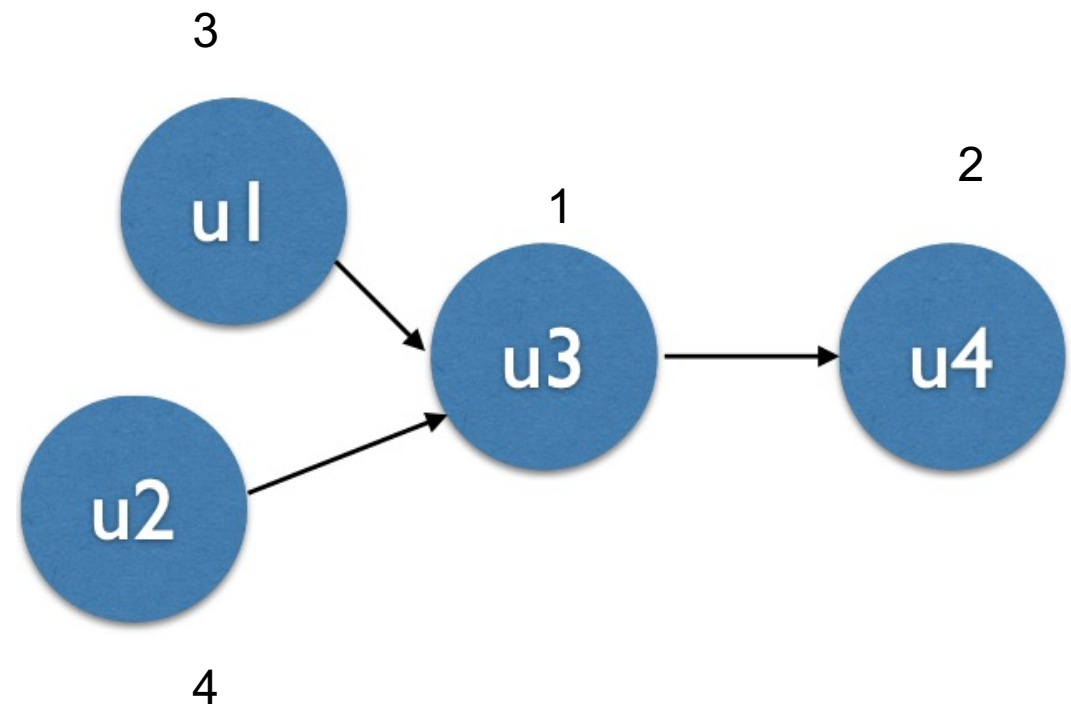
Kiểm thử đơn vị như nào?



***Theo thứ tự này làm như thế nào?
Có vấn đề gì?***

Kiểm thử đơn vị như nào?

- Mô hình hoá hệ thống theo từng lớp
 - Bắt đầu với các đơn vị không phụ thuộc vào đơn vị khác
 - Tiếp tục việc mô hình hoá với các đơn vị đã được kiểm thử
- Lợi ích
 - Tránh phải viết các đơn vị giả
 - Khi kiểm thử một đơn vị, các đơn vị sẽ được kiểm thử chỉ phụ thuộc vào các đơn vị đáng tin cậy



***Theo thứ tự này làm như thế nào?
Có vấn đề gì?***

Kiểm thử chương trình

```
public class lMath {  
  
    /** Returns an integer square root of x (discarding the fractional parts) */  
    public int isqrt(int x) {  
        int guess = 1;  
        while (guess * guess < x) {  
            guess++;  
        }  
        return guess;  
    }  
}
```

Phương pháp cổ điển

*/** A class to test the class IMath. */*

```
public class IMathTestNoJUnit {  
    /** Runs the tests. */  
    public static void main(String[] args) {  
        printTestResult(0);  
        printTestResult(1);  
        printTestResult(2);  
        printTestResult(3);  
        printTestResult(100);  
    }  
    private static void printTestResult(int arg) {  
        IMath tester=new IMath();  
        System.out.print("isqrt(" + arg + ") ==> ");  
        System.out.println(tester.isqrt(arg));  
    }  
}
```

Kết quả kiểm thử

- Phương pháp cổ điển đánh giá chương trình thế nào? Chương trình đúng hay sai?
- Có vấn đề gì với kết quả kiểm thử của phương pháp cổ điển?

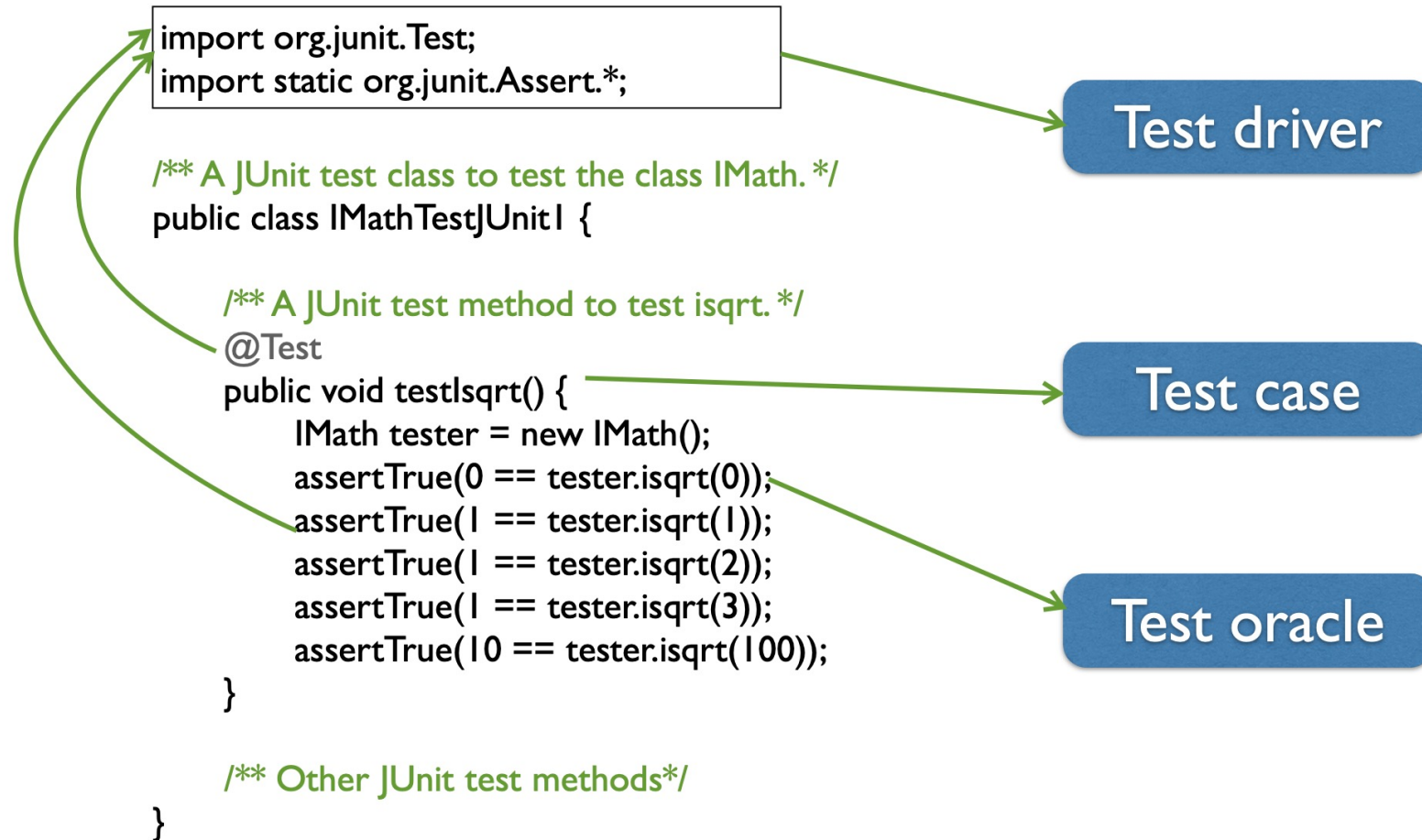
```
lsqrt(0) ==> 1  
lsqrt(1) ==> 1  
lsqrt(2) ==> 2  
lsqrt(3) ==> 2  
lsqrt(100) ==> 10
```

Solution?

- Tự động xác nhận bởi chương trình kiểm thử
 - Tự viết hoặc
 - Sử dụng công cụ hỗ trợ (JUnit)
- JUnit
 - Đơn giản, linh động, dễ sử dụng, open-source, dùng phổ biến cho Java
 - Làm tốt với test sets lớn
 - **junit.org**

JUnit

JUnit – Demo (1)



JUnit – Demo (2)

```
import org.junit.Test;
import static org.junit.Assert.*;
```

```
/** A JUnit test class to test the class IMath. */
public class IMathTestJUnit2 {
```

```
    /** A JUnit test method to test isqrt. */
```

```
    @Test
```

```
    public void testIsqrt() {
        IMath tester = new IMath();
```

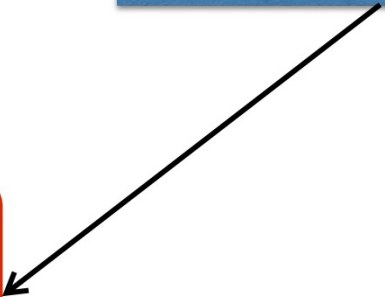
```
        assertEquals(0, tester.isqrt(0));
        assertEquals(1, tester.isqrt(1));
        assertEquals(1, tester.isqrt(2));
        assertEquals(1, tester.isqrt(3));
        assertEquals(10, tester.isqrt(100));
```

```
    }
```

```
    /** Other JUnit test methods*/
```

```
}
```

```
assertTrue(0 == tester.isqrt(0));
assertTrue(1 == tester.isqrt(1));
assertTrue(1 == tester.isqrt(2));
assertTrue(1 == tester.isqrt(3));
assertTrue(10 == tester.isqrt(100));
```



JUnit – Demo (2)

```
public class IMathTestJUnit4 {  
    private IMath tester;  
  
    @Before /** Setup method executed before each test */  
    public void setup(){  
        tester=new IMath();  
    }  
  
    @Test /** JUnit test methods to test isqrt. */  
    public void testIsqrt1() {  
        assertEquals("square root for 0 ", 0, tester.isqrt(0));  
    }  
    @Test  
    public void testIsqrt2() {  
        assertEquals("square root for 1 ", 1, tester.isqrt(1));  
    }  
    @Test  
    public void testIsqrt3() {  
        assertEquals("square root for 2 ", 1, tester.isqrt(2));  
    }  
    ...  
}
```



Test fixture

JUnit – Demo (3)

```
import org.junit.Test;
import static org.junit.Assert.*;
```

```
/** A JUnit test class to test the class IMath. */
public class IMathTestJUnit3 {
```

```
    /** A JUnit test method to test isqrt. */
```

```
    @Test
```

```
    public void testIsqrt() {
```

```
        IMath tester = new IMath();
```

```
        assertEquals("square root for 0 ", 0, tester.isqrt(0));
```

```
        assertEquals("square root for 1 ", 1, tester.isqrt(1));
```

```
        assertEquals("square root for 2 ", 1, tester.isqrt(2));
```

```
        assertEquals("square root for 3 ", 1, tester.isqrt(3));
```

```
        assertEquals("square root for 100 ", 10, tester.isqrt(100));
```

```
    }
```

```
    /** Other JUnit test methods */
```

```
}
```

JUnit – Demo (4)

@RunWith(Parameterized.class)

Indicate this is a
parameterized test class

public class IMathTestJUnitParameterized {

private IMath tester;

private int input;

private int expectedOutput;

To store input-output pairs

/** Constructor method to accept each input-output pair*/

public IMathTestJUnitParameterized(int input, int expectedOutput) {

 this.input = input;

 this.expectedOutput = expectedOutput;

}

@Before /** Set up method to create the test fixture */

public void initialize() {tester = new IMath();}

@Parameterized.Parameters /** Store input-output pairs, i.e., the test data */

public static Collection<Object[]> valuePairs() {

 return Arrays.asList(new Object[][] { { 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 1 }, { 100, 10 } });

}

@Test /** Parameterized JUnit test method*/

public void testIsqrt() {

 assertEquals("square root for " + input + " ", expectedOutput, tester.isqrt(input));

}

JUnit – Test suite

JUnit - Annotation

Annotation	Description
@Test	Identify test methods
@Test (timeout=100)	Fail if the test takes more than 100ms
@Before	Execute before each test method
@After	Execute after each test method
@BeforeClass	Execute before each test class
@AfterClass	Execute after each test class
@Ignore	Ignore the test method

Junit - Assertions

Assertion	Description
<code>fail([msg])</code>	Let the test method fail, optional msg
<code>assertTrue([msg], bool)</code>	Check that the boolean condition is true
<code>assertFalse([msg], bool)</code>	Check that the boolean condition is false
<code>assertEquals([msg], expected, actual)</code>	Check that the two values are equal
<code>assertNull([msg], obj)</code>	Check that the object is null
<code>assertNotNull([msg], obj)</code>	Check that the object is not null
<code>assertSame([msg], expected, actual)</code>	Check that both variables refer to the same object
<code>assertNotSame([msg], expected, actual)</code>	Check that variables refer to different objects

Tham khảo về JUnit

- Trang chủ: <https://www.junit.org/>
- Tutorials:
 - <https://www.vogella.com/tutorials/JUnit/article.html>
 - <https://www.tutorialspoint.com/junit/>



Buổi học sau

- Các kỹ thuật kiểm thử hộp đen
 - **Đọc trước chương 5**, Giáo trình kiểm thử phần mềm



Bài tập

- Viết các chương trình dưới đây bằng Java + Javadoc và sinh các test cases và test chương trình sử dụng JUnit (dùng IntelliJ IDEA):

DummyTel có cấu trúc tỷ lệ sau đây cho các cuộc gọi đường dài:

1. Bất kỳ cuộc gọi nào bắt đầu lúc hoặc sau 18:00 nhưng trước 08:00 được giảm 50%.
2. Bất kỳ cuộc gọi nào bắt đầu lúc hoặc sau 08:00 nhưng trước 18:00 được tính giá đầy đủ.
3. Tất cả các cuộc gọi đều phải chịu thuế 5%.
4. Tỷ lệ thông thường cho một cuộc gọi là 0,50 đồng/phút.
5. Bất kỳ cuộc gọi nào dài hơn 60 phút đều được giảm giá 15% trên chi phí (sau khi trừ đi bất kỳ khoản giảm giá nào khác nhưng trước khi cộng thuế).

** Chương trình đọc **thời gian bắt đầu** cuộc gọi dựa trên đồng hồ 24 giờ và **thời lượng** của cuộc gọi. Chương trả về **tổng chi phí rỗng** (sau khi trừ các khoản chiết khấu và cộng thuế). Chương trình sẽ giả sử chỉ các giá trị số nguyên được nhập vào, thời lượng không âm và thời gian bắt đầu biểu thị thời gian đồng hồ thực. Kết quả được làm tròn đến phần trăm gần nhất.