Software testing; week 8

Before deploying the application, you have to test it.

Test framework : JUnit

Testing for Java

## What is testing, in fact ?

Make sure the system behaves the way we want it to. Find bugs. You want the system to output exactly what you’re testing.

eg : Ariane 5 flight.

When we lost it, we had some conditions.

The problem of this flight was in the software

eg2 : Northeast blackout. Costed 6 Billion$. The main source of the problem was a variable that was improperly initialized.

The cost of testing represents 48% of the global cost to implement in your software.

There is testing and verification :

* **verification :** you can do it in a small app, when you can prove that your system won’t contain any bugs. But you can’t be 100% sure that your system won’t contain any bugs, bc you can’t check all the possible values (too many things to check)
* **testing : B method** (<https://fr.wikipedia.org/wiki/M%C3%A9thode_B>) → Ligne 14; proof that a program is correct.

⇒ finite number of values of parameters. You can test more values than while verifying your program.

In general, it’s impossible to test all possible values of the current complex system.

A software code isn’t right if it’s not tested. In Microsoft, you have a tester for each developer. *So, in fact, it is a very important job*, *and even internship if you want in fact.*

Even more than ½ startups fail because their softwares don’t have the appropriate corrections and testings.

How do we test?

* **Static testing** : you test your application without executing it. You only view its source code.
* **Dynamic testing** : the most used software testing technique. You test your app with input values and watch the behaviour of your program.

With what do we test?

* a specification : specifies the expected results.

You have to identify the expected result.

You can specify pre&post condition.

public class Account {

double balance;

pre : amount >= balance;

post : balance = balance + amount;

public void withdraw(double amount) {

balance = amount;

}

}

While testing, you have to compare the expected result with the obtained result. If they don’t match, it means that your test failed. (you don’t say)

Testing tells you there is an error but it doesn’t find the error in question

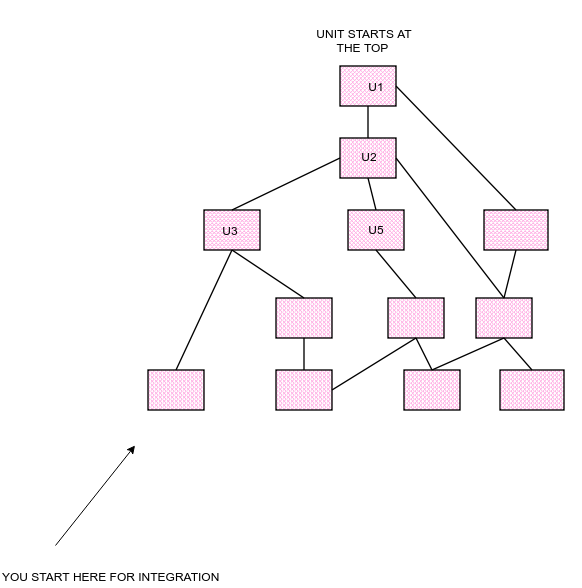
Phases in testing :

* **unit testing :** first step. You test each function & each class independently, separately.
  + if there are 100 classes, you test them all individually.
* **Integration and system testing :** you test the system as a whole, you don’t test things separately anymore. The program calls many/all functions.
* in which order should you test and integrate your functions?
* bottom-up approach : you start by testing the branches in the tree(from bottom), and you go up.
* **acceptance testing :** you test the different functionalities of the program.

JUnit :

define and executes tests and tests suites

write and debug code



* no regressing testing : checking that the introduced modifications do not introduce new errors.

To test your expected result :

* you have to check visually the display of your result ⇒ you can have some errors. + it’s difficult to check all the values needed (conventional)

⇒ you need to write a JUnit program that’ll do all the checking in your place

no conventional, but systematic.

* there is some additional statements as “assertTrue”/

Testing your code :

you have :

class Calculator {

public int add(int a, int b)

{return a+b+1;}

class CaclulatorTest extends TestCase {

@Test

public void testAdd() {

Calculator c= new Calculator();

assertTrue(c.add(3,5) == 8);

}

}

⇒ fail

Par contre == 9 ⇒ success