FACULTY OF ENGINEERING
AND ARCHITECTURE

SE 3 : TESTING

Bart Dhoedt

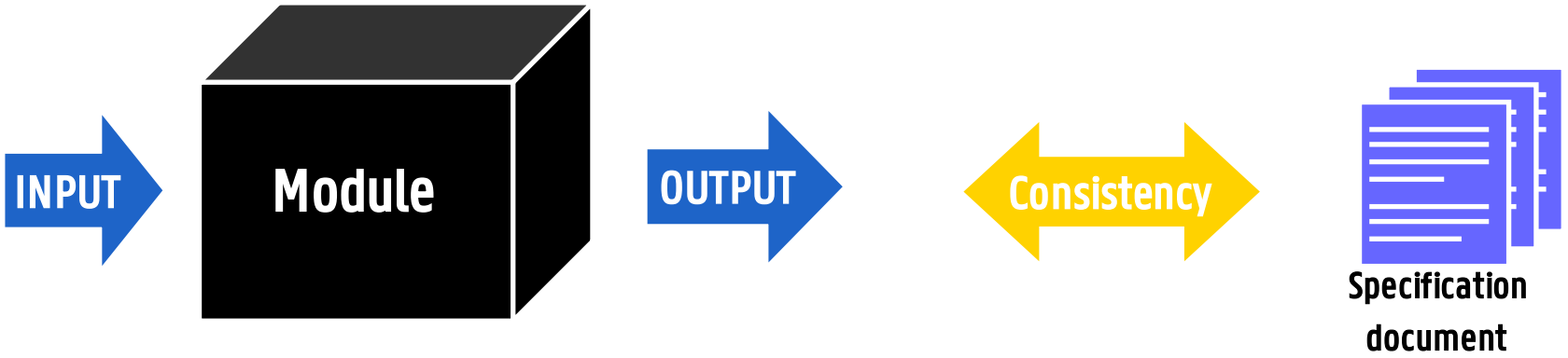
Academiejaar 2017-2018

1. TYPES OF TESTS

BLACK-BOX TESTING

= test to specifications

- ignore the code itself
- based on specification document
- = input/output testing
- after informal testing by programmer
- problem : combinatorial explosion of test cases !



BLACK-BOX TESTING

Techniques

- add test cases to detect NEW possible error
- keep track of test cases for regression testing

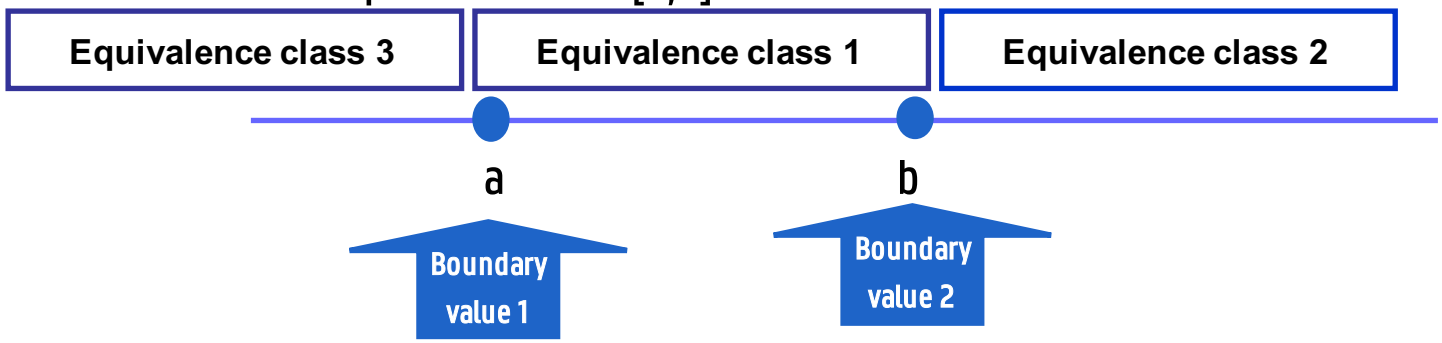
Equivalence and boundary analysis

- Partition input space into equivalence classes
- Take (at least) 1 test case for each class
- Test “boundary” values

BLACK-BOX TESTING

Examples

1. Numerical input in interval [a,b]



2. Character input belonging to predefined set

Equivalence class 1	Character belongs to set
Equivalence class 2	Character does not belong to set

BLACK-BOX TESTING

A product reads in two positive real numbers *a* and *b*, where *a* in [0,100] and *b* in]10,200[. A third real number *c* is read, with *c*>0.
In addition a command is read. There are 10 possible commands.

How many test cases are necessary for black box testing ?

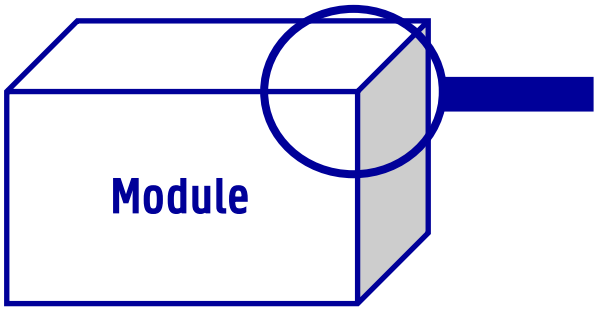
WHITE-BOX TESTING (GLASS BOX TESTING)

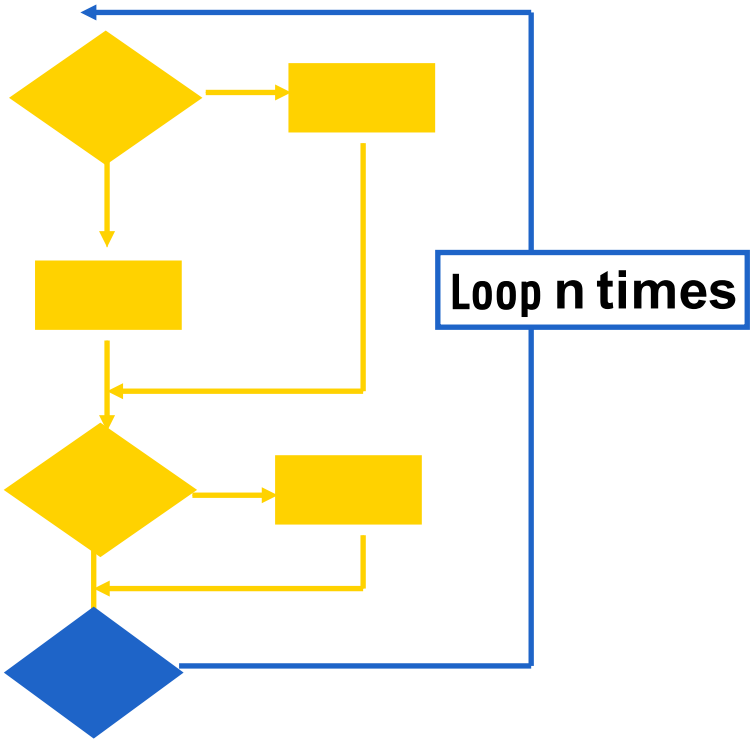
= test to code

- Test every possible path through the module
- Combinatorial problems !
- Does not reveal all possible problems !

4 possible paths

4^n possible paths

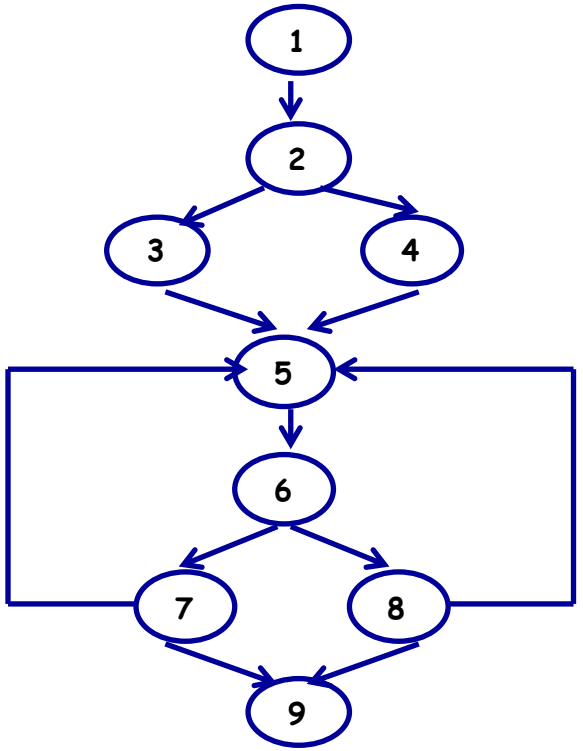




CONTROL FLOW GRAPH

- nodes are statements or groups of statements executed in sequence
- edges : control transfer between statement groups

```
public static int f(int n) {
1:      int a=0;
2:      if(n%2==0) {
3:              a=1;
        } else {
4:              a=2;
        }
5:      while(a<n) {
6:              if(a%2==0) {
7:                      a=2*a+1;
                }
                else {
8:                      a*=2;
                }
        }
9:      return a;
}
```



COVERAGE TECHNIQUES

- statement coverage
  = node coverage in control graph
- branch coverage
  = edge coverage in control graph
- multiple condition coverage
  = extended version of branch coverage
- cyclomatic coverage
  = limited set (base set) of path coverage
- all paths coverage
  = cover all paths in control graph

JUNIT

= test framework for unit testing

"Never in the field of software development was so much owed by so many to so few lines of code". [M. Fowler]

"The jewel on the crown of XP" (philosophy : code a bit – test a bit)

[freely downloadable from http://www.junit.org/]

"Testrunner" :

- runs test suites (= set of test cases, can be composed of other suites)
- reports results (textual or graphical UI)
  - test case gives **FAILURES** == produces the wrong result
    - wrong return value
    - not throwing expected exception
  - test case gives **ERRORS** == throws unexpected exceptions
- each TestCase has at least 1 call to assert-method

  Assert.assertTrue(<actual boolean result>);

  Assert.assertEquals(<actual result>,<expected result>);

JUNIT

- Annotations
    - method to test : @Test
    - fixtures
        - setup method : @Before
        - teardown method : @After
    - expensive resources (e.g. database connection)
        - construct : @BeforeClass (static)
        - cleanup : @AfterClass (static)
- testing exceptions

```
@Test(expected=IndexOutOfBoundsException.class)
public void empty() {
        ArrayList l=new ArrayList();
        l.get(-1);
}
```

-Timout (prevent endless loop)

```
@Test(timeout = 100)
```

-Ignoring test case : @Ignore

- obsolete test cases
- Tests take too long ...

TEST RUN LOGIC

```
import org.junit.*;
public class Test2 {
        @BeforeClass
        public static void setUpBeforeClass() throws Exception {System.out.println("setUpBeforeClass");}
        @AfterClass
        public static void tearDownAfterClass() throws Exception {System.out.println("tearDownAfterClass");}
        @Before
        public void setUp() throws Exception {System.out.println("setUp");}
        @After
        public void tearDown() throws Exception {System.out.println("tearDown");}
        @Test
        public void test1() {System.out.println("test1");}
        @Test
        public void test2() {System.out.println("test2");}
}
```

```
setUpBeforeClass
setUp
test1
tearDown
setUp
test2
tearDown
tearDownAfterClass
```

ASSERTIONS

import static org.junit.Assert.*;

fail(String)
Let the method fail, might be usable to check
that a certain part of the code is not reached.

assertTrue(true);
True

assertEquals([String message], expected, actual)
Test if the values are the same.
Note: for arrays the reference is checked not the content of the arrays

assertEquals([String message], expected, actual, tolerance)
Usage for float and double; the tolerance are the number
of decimals which must be the same

assertNull([message], object)
Checks if the object is null

assertNotNull([message], object)
Check if the object is not null

assertSame([String], expected, actual)
Check if both variables refer to the same object

assertNotSame([String], expected, actual)
Check that both variables refer not to the same object

assertTrue([message], boolean condition)
Check if the boolean condition is true.

JUNIT-TESTING

Annotate test with @Test

Test cases spread over multiple classes

- modular testing (building incremental test sets)
- reusing part of the test code

```
import org.junit.*;
import org.junit.runner.*;
import org.junit.runners.*;
@RunWith(Suite.class)
@Suite.SuiteClasses({
        A.class,
        B.class,
        C.class,
})

public class TestSuite {

}
```

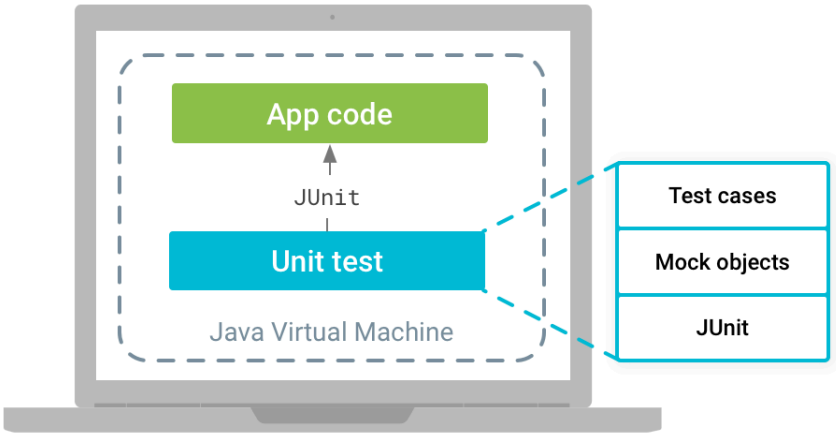
2. TESTING IN ANDROID

- POJO-TESTS

- INSTRUMENTED TESTS

TYPES OF TESTS : LOCAL TESTING

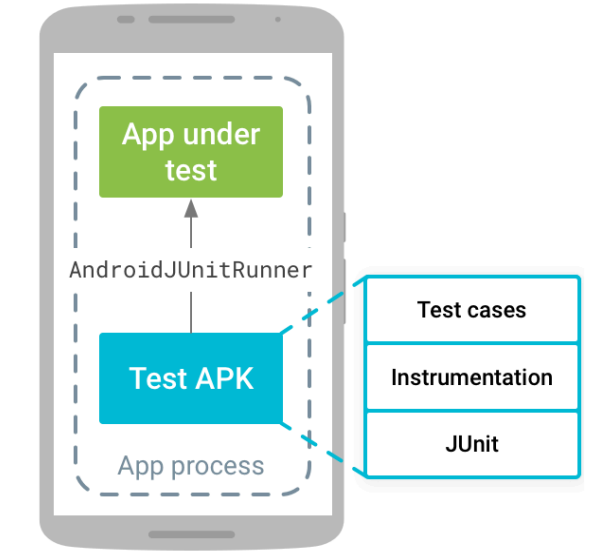
- runs on standard JVM, without using Android dependencies
- Android package replaced by Exception-throwing methods
- used to reveal logical errors fast
    - avoid complexity of hardware setup
    - avoid overhead of emulation



Local unit test
`src/test/java/`

TYPES OF TESTS : INSTRUMENTED TEST

- code runs on actual device or emulator
- uses “real” Android software
- test for situations that can not be tested through mocking
    - response time
    - interaction with sensor hardware
    - …



Instrumented test
`src/androidTest/java/`

TYPES OF TESTS : USER INTERFACE TESTS

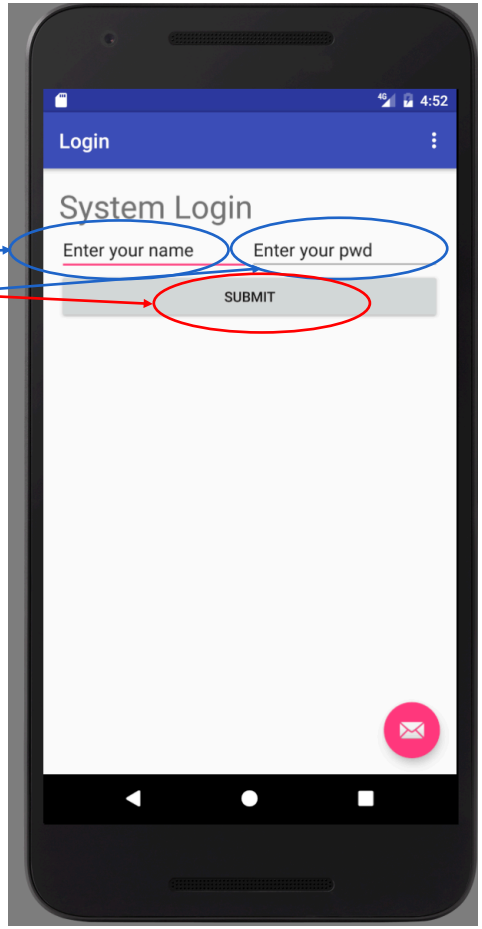
- mimic user interaction programmatically
- intercept interaction to UI-widgets
- Espresso-library : test scenario
    - find the view(s)
    - perform some actions on (number of) view(s)
    - check the result

SIMPLE EXAMPLE : LOGIN

```
public class MainActivity extends AppCompatActivity {

    public void checkLogin(View view){
        String login = ((EditText) findViewById(R.id.editText)).getText().toString();
        String pwd = ((EditText) findViewById(R.id.editText2)).getText().toString();
        TextView v = (TextView) findViewById(R.id.textView);
        Rule[] r = new Rule[]{new CapitalLoginCheck(),
                new CapitalPWDCheck(),
                new DBchecker()};
        LoginChecker l = new LoginChecker(r);
        v.setText(l.check(login, pwd));
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
                ... standard Android ...
    }
}
```



SIMPLE EXAMPLE : LOGIN

```
public class LoginChecker {
    private Rule[] rule;
    public LoginChecker(Rule[] r){
        this.rule = r;
    }
    public String check(String login, String pwd){
        for(Rule r:rule){
            String result = r.isValid(login, pwd);
            if(!(result.equals("OK"))) {
                return result;
            }
        }
        return "OK";
    }

}
```

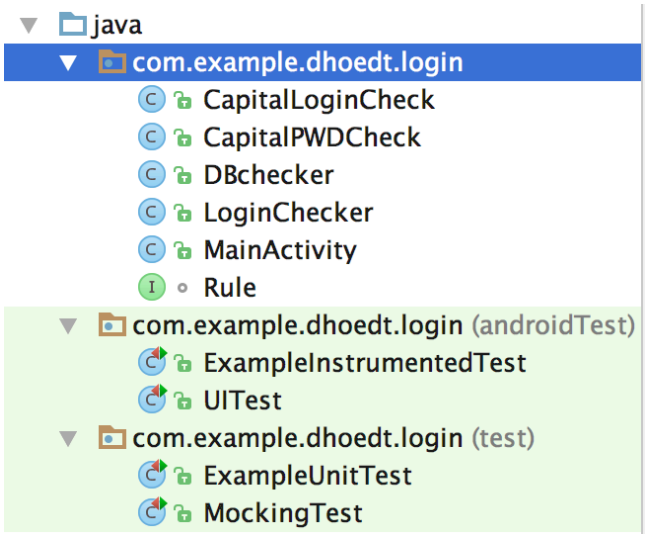
```
interface Rule {
   String isValid(String login, String pwd);
}
public class CapitalLoginCheck implements Rule {
   public String isValid(String login, String pwd){
      if(login.equals(login.toUpperCase())) {
         return "check CAPS lock in login field !";
      } else {
         return "OK";
      }
   }
}
```

```
public class CapitalPWDCheck implements Rule {
   public String isValid(String login, String pwd){
      if(pwd.equals(pwd.toUpperCase())) {
         return "check CAPS lock in pwd field !";
      } else {
         return "OK";
      }
   }
}
```

```
public class DBchecker implements Rule {
   public String isValid(String login, String pwd){
      // something complex, Android dependent ...
      return "OK";
   }
}
```

1. LOCAL UNIT TESTS (NO MOCKING)

```
public class ExampleUnitTest {
    @Test
    public void capitalLoginCheck() throws Exception {
        CapitalLoginCheck capitalLogin = new CapitalLoginCheck();
        assertEquals(capitalLogin.isValid("AAAAAA","BBBB"),"check CAPS lock in login field !");
        assertEquals(capitalLogin.isValid("AAAAAa", "BBBB"),"OK");
    }
    @Test
    public void capitalPWDCheck() throws Exception {
        CapitalPWDCheck capitalPwd = new CapitalPWDCheck();
        assertEquals(capitalPwd.isValid("AAAAAA","BBBB"),"check CAPS lock in pwd field !");
        assertEquals(capitalPwd.isValid("AAAAA", "BBBBb"),"OK");
    }
    @Test
    public void addition_isCorrect() throws Exception {
        assertEquals(4, 2 + 2);
    }
}
```



2. INSTRUMENTED TESTS (NO UI TESTING)

```
import android.content.Context;
import android.support.test.InstrumentationRegistry;
import android.support.test.runner.AndroidJUnit4;
import org.junit.Test;
import org.junit.runner.RunWith;
import static org.junit.Assert.*;

@RunWith(AndroidJUnit4.class)
public class ExampleInstrumentedTest {
    @Test
    public void capitalLoginCheck() throws Exception {
        CapitalLoginCheck capitalLogin = new CapitalLoginCheck();
        assertEquals(capitalLogin.isValid("AAAAAA","BBBB"),"check CAPS lock in login field !");
        assertEquals(capitalLogin.isValid("AAAAAa", "BBBB"),"OK");
    }
    @Test
    public void capitalPWDCheck() throws Exception {
        CapitalPWDCheck capitalPwd = new CapitalPWDCheck();
        assertEquals(capitalPwd.isValid("AAAAAA","BBBB"),"check CAPS lock in pwd field !");
        assertEquals(capitalPwd.isValid("AAAAA", "BBBBb"),"OK");
    }
}
```

2. INSTRUMENTED TESTS (UI TESTING - ESPRESSO)

```
@RunWith(AndroidJUnit4.class)
@LargeTest
public class UITest {
  @Rule
  public ActivityTestRule<MainActivity> mActivityRule = new ActivityTestRule<>(MainActivity.class);
  @Test
  public void testValidUser() throws Exception {
    onView(withId(R.id.editText)).perform(clearText(), typeText("Bart"), closeSoftKeyboard());
    onView(withId(R.id.editText2)).perform(clearText(), typeText("Pass"), closeSoftKeyboard());
    onView(withId(R.id.button)).perform(click());
    onView(withId(R.id.textView)).check(matches(withText("OK")));
  }
  @Test
  public void testCapitalLogin() throws Exception {
    onView(withId(R.id.editText)).perform(clearText(), typeText("BBBBBB"), closeSoftKeyboard());
    onView(withId(R.id.editText2)).perform(clearText(), typeText("PPPPP"), closeSoftKeyboard());
    onView(withId(R.id.button)).perform(click());
    onView(withId(R.id.textView)).check(matches(withText("check CAPS lock in login field !")));
  }
}
```