Tests & Continuous Integration

J.-M. Bruel – jbruel@gmail.com – Version 21.2 2021-11-23

Table of Contents

1. **Why** testing?
    1.1. To deliver the **good product**
    1.2. If it works for **1** doesn't necessarily for **100**
    1.3. **Murphy**'s law
    1.4. Different **OS** or different **terms**
    1.5. To give **the best**
2. A concrete example of **mandatory** test
3. Concrete example of mandatory **documentation**
4. Tests' Typology
5. JUnit etc.
    5.1. What to test?
    5.2. Assertions
    5.3. Tests Strategies
    5.4. Tests order
    5.5. What about graphical interfaces?
    5.6. Tests' coverage
6. Concrete application
    6.1. From `To Be Done` to `On going`
    6.2. Create a specific branch (if new *feature*)
    6.3. Write a failing test, then make it pass
    6.4. Merge and do integration testing
    6.5. Commit & Push in `devs`
    6.6. From `On going` to `Review`

To go further...

1. Why testing?

> *A majority of the production failures (77%) can be reproduced by a unit test.*
>
> — *Yuan et al. OSDI 2014*

Figure 1. A recent tweet...

See https://blog.acolyer.org/2016/10/06/simple-testing-can-prevent-most-critical-failures/amp/

1.1. To deliver the good product

Figure 2. A product does what it's supposed to do

1.2. If it works for 1 doesn't necessarily for 100

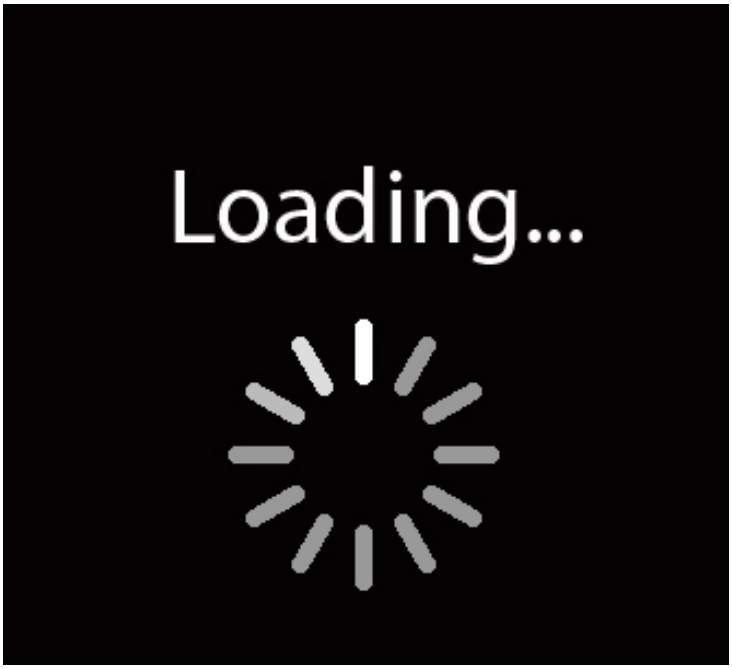


Figure 3. Scalability issue

1.3. Murphy's law

> *Everything that can go wrong will eventually go wrong.*
>
> *— Edward A. Murphy Jr.*

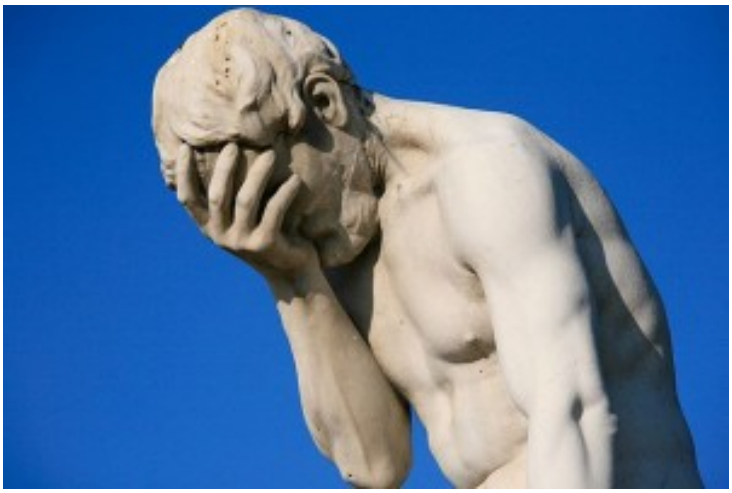


Figure 4. Murphy's law

1.4. Different OS or different terms

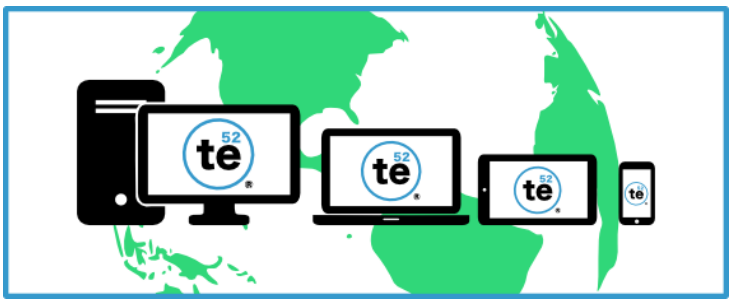


Figure 5. Diversity

1.5. To give the best

Figure 6. Doing the best

2. A concrete example of mandatory test

Asciidoctor Contribution

(https://github.com/asciidoctor/asciidoctor/blob/master/CONTRIBUTING.adoc)

Figure 7. Around a beer with Dan Allen, Denver, Colorado #ILoveMyJob

1. Fork the repository.
2. Run `bundle` to install development dependencies.
3. Create a topic branch
4. Add tests for your unimplemented feature or bug fix. (See [writing-and-executing-tests])
5. Run `bundle exec rake` to run the tests. If your tests pass, return to step 4.
6. Implement your feature or bug fix.
7. Run `bundle exec rake` to run the tests. If your tests fail, return to step 6.
8. Add documentation for your feature or bug fix.

9. If your changes are not 100% documented, go back to step 8.
10. Add, commit, and push your changes.
11. Submit a pull request.

3. Concrete example of mandatory documentation

Figure 8. After a running with Gaël Blondel, Saint-Malo #ILoveMyJob

> *[...] an Eclipse project is providing extensible frameworks and applications accessible via documented APIs.*
>
> — *Eclipse Development Process*

4. Tests' Typology

Table 1. Difference between Verification & Validation (source https://www.tutorialspoint.com/software_testing/software_testing_quick_guide.htm)

Verification	Validation
Is the product **good**?	Is it **the good** product?
Are you building it right?	Are you building the right thing?
Mostly done by **developers**	Mostly done by **client**
Comes first	After verification most of the time

5. JUnit etc.

5.1. What to test?

Exceptions	`@Test (expected = Exception.class)`
Execution time	`@Test(timeout=100)`
Specific environment	`System.getProperty("os.name").contains("Linux"));`

5.2. Assertions

`fail([message])`	Force the test to fail
`assertTrue([message,] condition)`	Condition is true
`assertFalse([message,] condition)`	Condition is false
`assertEquals([message,] expected, actual)`	Two values are equal

`assertNull([message,] object)`	null object
`assertSame([message,] expected, actual)`	identical objects (same réf.)

5.3. Tests Strategies

Considering `int add(int,int);` from class `myClass`.

(source : http://stackoverflow.com/questions/8751553/how-to-write-a-unit-test)

JAVA
```
//for normal addition
@Test
public void testAdd1Plus1() {
  int x  = 1 ; int y = 1;
  assertEquals(2, myClass.add(x,y));
}
```

Examples:

- *overflow*
- `null` parameters
- negative parameters
- ...

(source : http://stackoverflow.com/questions/8751553/how-to-write-a-unit-test)

JAVA
```
//if you are using 0 as default for null, make sure your class works in
that case.
@Test
public void testAdd1Plus1() {
  int y = 1;
  assertEquals(0, myClass.add(null,y));
}
```

5.4. Tests order

None!!

> *JUnit assumes that all test methods can be executed in an arbitrary order. Well-written test code should not assume any order, i.e., tests should not depend on other tests.*
>
> — *JUnit manual*

5.5. What about graphical interfaces?

Example of the `Robot` library:

(source : http://stackoverflow.com/questions/16411823/junit-tests-for-gui-in-java)

JAVA

```
Robot bot = new Robot();
bot.mouseMove(10,10);
bot.mousePress(InputEvent.BUTTON1_MASK);
//add time between press and release or the input event system may
//not think it is a click
try{Thread.sleep(250);}catch(InterruptedException e){}
bot.mouseRelease(InputEvent.BUTTON1_MASK);
```

Example of the `swingcoder` Eclipse (http://www.eclipse.org) plugin:

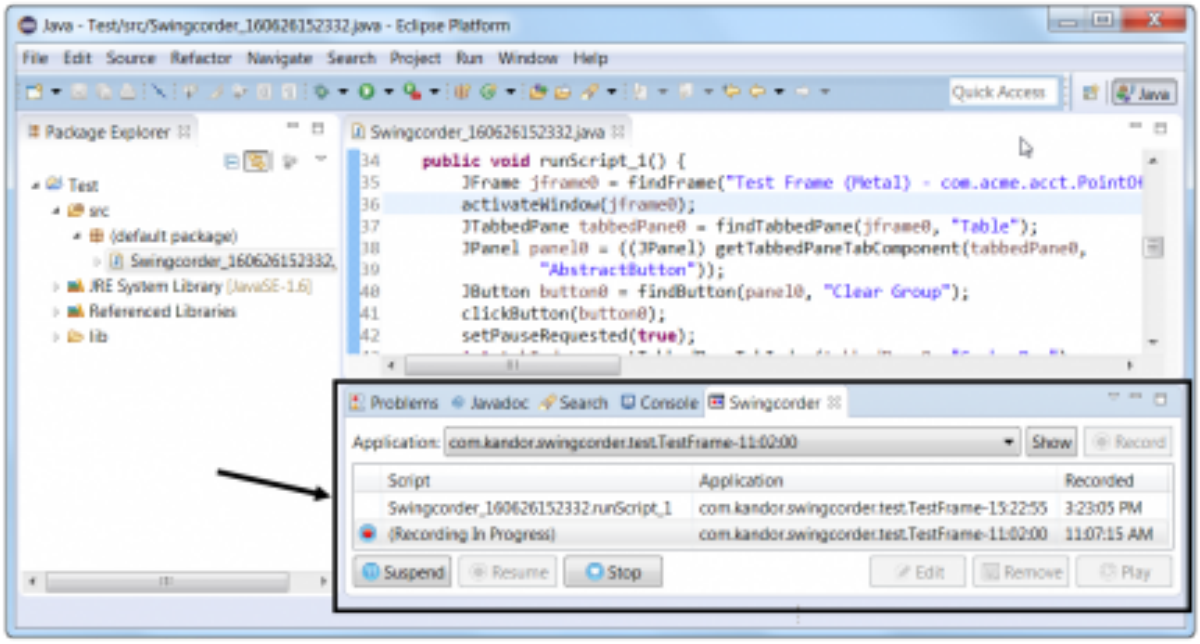


Figure 9. Simulation d'utilisation d'interface (source https://marketplace.eclipse.org/content/swingcorder)

5.6. Tests' coverage

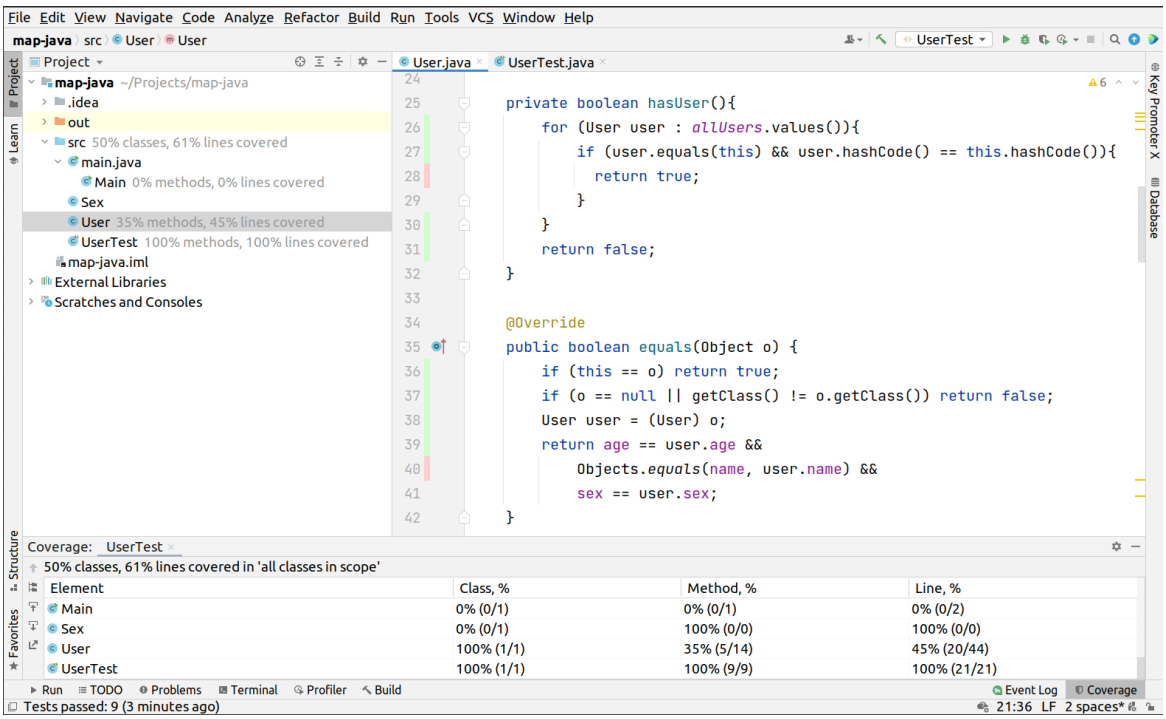
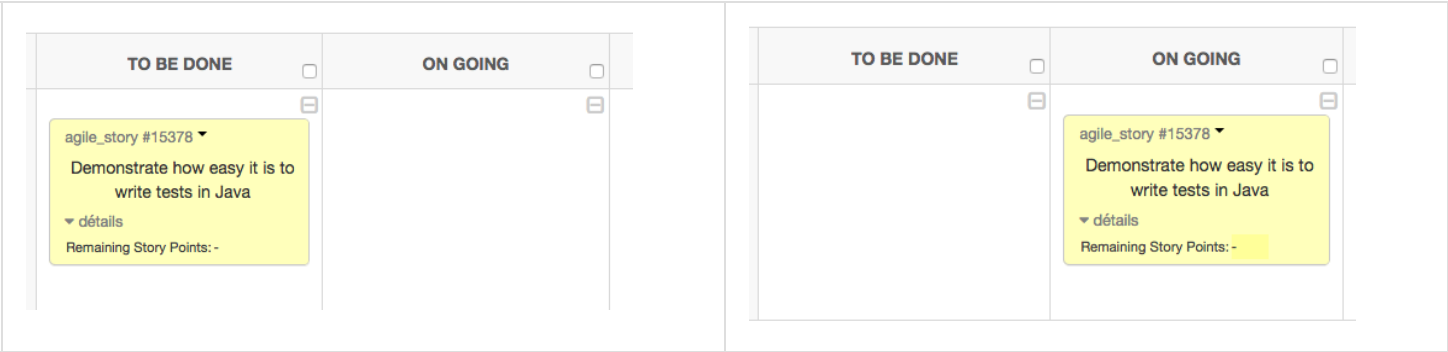


Figure 10. Tests' coverage (IntelJ IDEA: https://www.jetbrains.com/idea/)

6. Concrete application

6.1. From `To Be Done` to `On going`

Table 2. Update your kanban



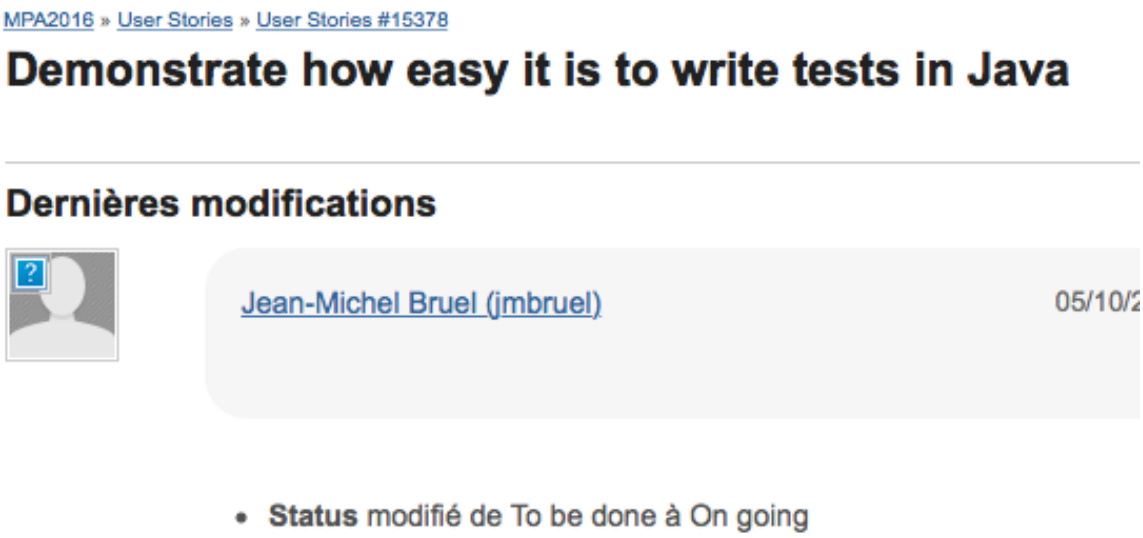


Figure 11. Confirmation

6.2. Create a specific branch (if new *feature*)

```shell
bruel (master) $ git checkout -b US-15378
Switched to a new branch 'US-15378'
bruel (US-15378) $
```

6.3. Write a failing test, then make it pass

6.4. Merge and do integration testing

```shell
bruel (US-15378) $ git commit -am "Adding push feature. Tests OK"
[US-15378 78f3242] Adding push feature. Tests OK
 1 file changed, 2 insertions(+), 3 deletions(-)
bruel (US-15378) $ git checkout devs
Switched to branch 'devs'
bruel (devs) $ git merge US-15378
```

6.5. Commit & Push in `devs`

```shell
bruel (devs) $ git commit -am "..."
...
bruel (devs) $ git push origin devs
...
bruel (devs) $ git branch -D US-15378
Deleted branch US-15378 (was f392a73).
```

6.6. From `On going` to `Review`

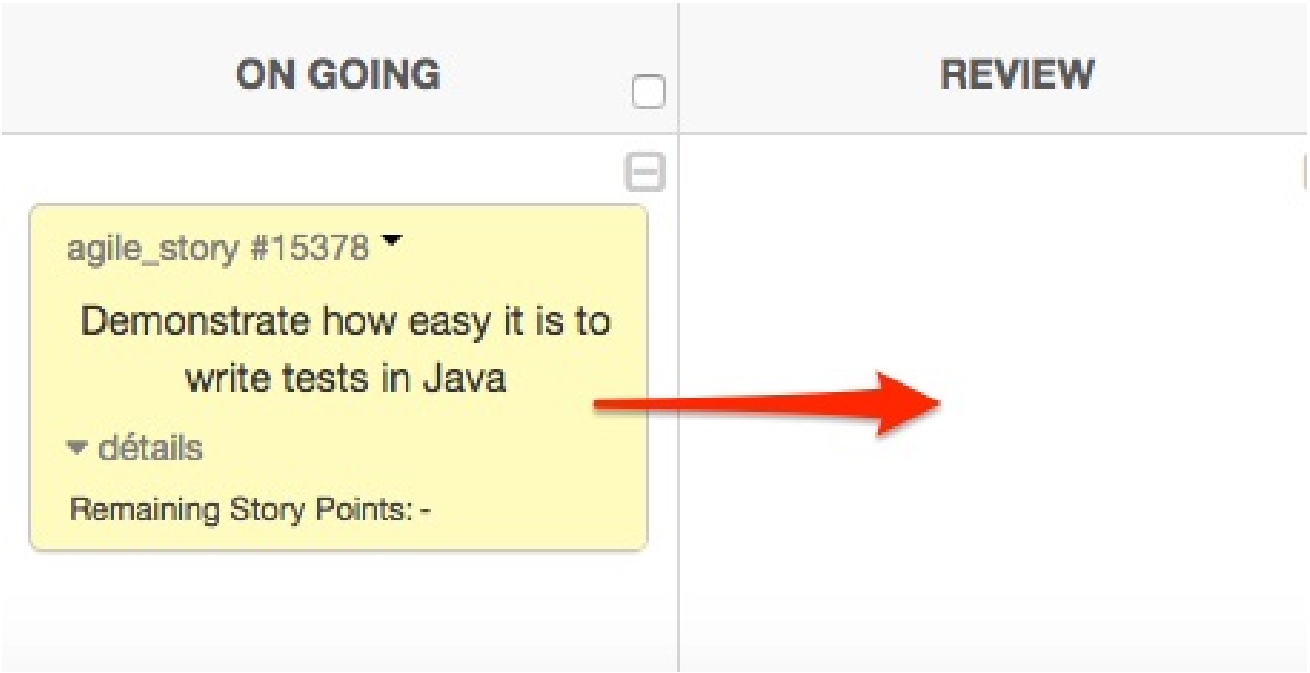


Figure 12. Update your kanban

To go further…

- http://rpouiller.developpez.com/tutoriels/java/tests-unitaires-junit4/

- https://jmbruel.github.io/teaching/topics/agile.html#_les_tests
- http://www.vogella.com/tutorials/JUnit/article.html
- http://junit.org
- http://stackoverflow.com/questions/8751553/how-to-write-a-unit-test
- https://www.quora.com/How-do-you-get-developers-to-love-testing-their-code

Version 21.2 2021-11-23
Last updated 2021-11-23 17:40:23 +0100