TMar, executable specification and automated acceptance tests

Version 1.5, July 2014

Tmar – Introduction

- Tmar is a companion for standards java/groovy testing Framework (JUnit, TestNG, Spock)
- It brings the capacity to share executable specifications, test descriptions and test results between functional team and developers
- Originally Inspired by Fitness, we tried to keep its utilization as well as its integration as simple as possible.
- Tmar comes from the frustration to have to change our actual testing framework to be able to have external test descriptions shared with our functional team members

Tmar – first simple example (Junit/TestNG)

TestDemo.multiply_test1.tmar

```
Multiplication test

/each
[ number 1 | number 2 | result ? |
|     2    |     2    |     4    |
|     2    |     3    |     6    |
```

TestDemo.groovy

```
class TestDemo extends Tmar4JUnit {
        @Test
        def multiply() {
        eachIteration('multiply_test1'){ tmar ->
        tmar.result = tmar.number1 * tmar.number2
        }
        }
}
```

TestDemo.multiply_test1.html

OK - Multiplication test
2 iteration(s) with 0 in error - 26/10/13 00:51:06 - TestDemo.multiply_test1

EACH

number1	number2	result
2	2	4
2	3	6

Tmar – first simple example (Spock)

TestDemo.multiply_test1.tmar

```
Multiplication test

/each
[ number 1 | number 2 | result ? |
|     2    |     2    |     4    |
|     2    |     3    |     6    |
```

TestDemo.groovy (with Spock)

```
class TestDemo extends Tmar4Spock {

  def multiply() {
    when:
      tmar.result = tmar.number1 * tmar.number2
    then:
      tmar.asserts()
    where:
      tmar << getData('multiply_test1')
  }
}
```

TestDemo.multiply_test1.html

OK - Multiplication test
2 iteration(s) with 0 in error - 26/10/13 00:51:06 - TestDemo.multiply_test1

EACH

number1	number2	result
2	2	4
2	3	6

Tmar – first simple example

Simple Tmar description file

By convention Tmar test description file name is <ClassTestName>.<testName>.tmar

TestDemo.multiply_test1.tmar

```
Multiplication test

/each
[ number 1 | number 2 | result ? |
|     2    |    2     |     4    |
|     2    |    3     |     6    |
```

Iterations description
All Tmar statements begin with «**/**»

Test title (description first line)

Iteration table header, start with «**[**» and have a «**|**» between each column

Iteration table values
Columns are delimited by a «**|**»
The test method will be fired for each line of the **/each** table

Columns name, used as field name for each Iteration.
Camel case convention is applied when fields are given to the test method

number 1 → number1
First Name → firstName

When a column name is followed by «**?**» Tmar will compare the value of this column with the value returned by the test method.

If there is no value in the column Tmar will display the value returned by the test method.

Tmar – first simple example

Simple JUnit or TestNG - Tmar test class

Test class extends **Tmar4JUnit** or **Tmar4TestNG**

@Test
JUnit or TestNG annotation for Test method

TestDemo.groovy

```
class TestDemo extends Tmar4JUnit {
    @Test
    def multiply() {
        eachIteration('multiply_test1'){
        tmar.result = tmar.number1 * tmar.number2
        }
    }
}
```

1

2

3

eachIteration() brings values of each iteration in a Tmar object from the Tmar description

tmar (iteration 1)

number1 = 2
Number2 = 2
Result = ?

For each iteration the business rules assign values to the fields expected by the test description (here : result)

tmar (iteration 1)

Result = 4
Expected = 4

OK

Tmar compare the expected values of the iteration with the assigned values

TestDemo.multiply_test1.tmar

```
Multiplication test

/each
[ number 1 | number 2 | result ? |
|     2    |     2    |     4    |
|     2    |     3    |     6    |
```

Tmar – first simple example

Simple Spock - Tmar test class

Test class extends **Tmar4Spock** which extends **Spock specification**.

TestDemo.groovy

```
class TestDemo extends Tmar4Spock {

  def multiply() {
    when:
      tmar.result = tmar.number1 * tmar.number2
    then:
      tmar.asserts()
    where:
      tmar << getData('multiply_test1')
  }
}
```

Test method

tmar (iteration 1)

number1 = 2
Number2 = 2
Result = ?

*For each iteration the **when** block is fired. The business rules assign values to the fields expected by the test description (here : result)*

tmar (iteration 1)

Result = 4
Expected = 4

OK

*After the execution of the when block the **then** block is fired.*
A call to the asserts() method of the Tmar superclass makes Tmar compare the expected values of the iteration with the values assigned in the when block

*First, in the **where** block, tmar << getData('multiply_test1') brings values of each iteration in a Tmar object from the Tmar description*

TestDemo.multiply_test1.tmar

```
Multiplication test

/each
[ number 1 | number 2 | result ? |
|    2     |    2     |    4     |
|    2     |    3     |    6     |
```

Tmar – use of text

- Tmar allows for text in your description in order to comment or explain your test case

Multiplication test

This text explain the purpose of the test
...

```
/each
[ number 1 | number 2 | result ? |
|     2      |      2     |      4     |
```

It's also possible to have text between table's lines to explain the iteration

```
|     2      |      3     |      6     |
```

Tmar – /inline statement

- To be even more expressive Tmar supports the **/inline** statement.
  It allows to spread the fields of an iteration in a text bloc
  - each /inline and each in tabulation can be freely mixed in the same description
- It also supports text fields value longer than one line of text using “”” “”””

```
Multiplication test

/each
[ number 1 | number 2 | result ? | text 1 |
|    2     |    2     |     4    | four   |
|    2     |    3     |     6    | six    |

/each
/inline
If we have [number 1] 2 | and [ number 2 ] 4 |
the multiplication will have for [ result ? ] 8 | and [text 1| height |

/inline
The same operation can be done with [number 1] 2 | and [ number 2 ] 5|
In this case the multiplication will have for [ result ? ] 10 |
[text 1] """
This text doesn't fit in one
line of text
"""
```

Tmar – values type & comparisons

- By default Tmar automatically manages four types of data
  - Text (default) , integer, decimal and boolean (yes, no, true, false)
  - This means that Tmar will try to automatically convert the input data before passing them to the test method (we will see later how to override this behavior)
  - It's possible to enforce the string type by using " " (example "003")

```
Multiplication test

/each
[ name      |  age ?  | code   | married | Size   |
| Jenny     |   27    | "001"  |  yes    |  1.65  |
| peter     |   35    | "002"  |  no     |  1.80  |
```

String | Integer | String | Boolean | Double

- Values Comparison
  - As we have seen previously, a trailing "?" into the column name instructs Tmar to consider the column as a result column. For each Iteration the value in the column will be compared to the returned value computed by the test method
  - If there is a single value in the column, Tmar will test equality against the returned value
  - It's also possible to use the following expressions (where "?" is the returned value)
    - | ? < 27 | ? <= 27 | ? > 27 | ? >= 27

Tmar – Assertion error

- Tmar displays assertion errors in the html report file and in the IDE's output console :

EACH

number 1	number 2	result
2	7	14 < 13

```
Assertion failed:

assert  result < max_expected // iteration : 6
        |      | |
        14     | 13
               false
```

- If you use the Tmar Eclipse Plugin, the error is also directly highlighted in the Tmar description

Tmar – second example

```
Multiplication test

/each
[ number 1 | number 2 | result ? |
|    2     |     2    |     4    |
|    2     |     3    |     6    |

It's possible to spread the fields of an iteration in a
text with the statement /inline

/each
/inline
If we have [number 1] 2 | and [ number 2 ] 4 |
the multiplication will have as [ result ? ] 8 |

/inline
The same operation can be done with [number 1] 2 | and [
number 2 ] 5 |
In this case the multiplication will have as [ result ? ]
10 |

/each
It's also possible to have some text between iterations
lines, For example :

This line is testing that 2 * 6 >= 12

[ number 1 | number 2 | result ?   |
|    2     |     6    |   ? >=  12 |

and this one that 2 * 7 < 14, it will raise an error.
As you can see it's not necessary to repeat the header
before each raw

|    2     |     7    |   ? < 14   |
```

1 ERROR(S) - Multiplication test
6 iteration(s) with 1 in error - 26/10/13 02:03:37 - TestDemo.multiply_test1

EACH

number 1	number 2	result
2	2	4
2	3	6

It's possible to spread the fields of an iteration in a text with the statement inline

EACH inline

If we have number1 : **2** and number2 : **4**
The multiplication will have for result : **8**

EACH inline

The same operation can be done with number1 : **2** and number2 : **5**
In this case the multiplication will have for result : **10**

It's also possible to have some text between iterations lines, For example :

This line is testing than 2 * 6 >= 12

EACH

number 1	number 2	result
2	6	12

and this one than 2 * 7 < 14, it will raise an error.
As you can see it's not necessary to repeat header before each raw

EACH

number 1	number 2	result
2	7	14 < 14

Tmar – Improved html presentation report

A line started with "_" is considered as a subtitle. It will also be an anchor used by an index summary in the html report

```
_bulleted list

* item 1
* item 2
* item 3

#image groovy-logo.png

_align column table

/table align

[    code    |  label                  |    number  |
|    "001"   | first label             |          1 |
|    "002"   | second label longer     |          2 |
```

Lines started with "*" are considered as list item

Image insertion
Key word image is introduced by #

Center aligned **Left** aligned **Right** aligned

Tmar considers the column header to determine column alignment

If the column header right white space is twice as big as the left white space the column is left align

If the left white space is twice as big as the right white space the column is right align, otherwise the column is centered

Tmar also support Markdown formating langage to improve report presentation (more detail at the end of the document)

bulleted list

- item 1
- item 2
- item 3

align column table

TABLE align

code	label	number
001	first label	1
002	second label longer	2

Tmar – images, documents and href

- Images can be added to your Tmar pages
  - #image <image name>

```
#image myImage.jpeg
```

  - The image has to be in a **/images** subdirectory of the Tmar description directory tree

- Tmar also let you add some href links
  - #url <link>,<label>;

```
#url http://www.jspresso.org/,Jspresso site;
```

- And with a similar syntax Tmar let you add links to documents
  - #document <document name>,<label>;

```
#document spec.ppt,Specifications;
```

  - The linked document has to be in a **/documents** subdirectory of the Tmar description directory tree

Tmar – execution in sequence **with Spock**

- Using the **/each** statement, <u>each iteration are run separately and **in parallel**</u>
- It's sometimes necessary to ensure that iterations are run in order. This is especially true if you want to use the result of the previous iteration
- To do so, you can use the **/sequence** statement instead of the **/each** one
- In that case getData() will return an Object with a list of iterations and you will have to use the tmar.hasNext() method to iterate in.

TestDemo.accumulation_test1.tmar

```
accumulation of values

/sequence
[ number 1 | number 2 | result ? | accumulated ? |
|     2      |     2      |      4      |      4           |
|     2      |     3      |      6      |     10           |
|     2      |     4      |      8      |     18           |
```

OK - Accumulation values
3 iteration(s) with 0 in error - 04/11/13 15:16:19

SEQUENCE

number 1	number 2	result	accumulated
2	2	4	4
2	3	6	10
2	4	8	18

TestDemo.groovy

```
class TestDemo extends Tmar4spock {

  def accumulated() {
    Integer cumul = 0
    when:
      while (tmar.hasNext()) {
        def total = tmar.number1 * tmar.number2
        tmar.result = total
        cumul = cumul + total
        tmar.accumulated = cumul
      }
    then:
      tmar.asserts()
    where:
      tmar << getData('accumulation_test1')
  }
}
```

Tmar – Dealing with scenario (Junit/TestNG)

- The easiest way to deal with a scenario is to use a **switch** statement on an iteration field.
- This lets you run different business rules depending of this field value

TestDemo.test_account_operations.tmar

```
Bank account operations tests

/each
[operation |  amount |  balance ? |
| deposit  |    2000 |      2000  |
| deposit  |     500 |      2500  |
| withdraw |     700 |      1800  |
```

OK - Bank account operations tests
3 iteration(s) with 0 in error - 04/11/13 14:51:49

SEQUENCE

operation	amount	balance
deposit	2000	2000
deposit	500	2500
withdraw	700	1800

TestDemo.groovy

```
class TestDemo extends Tmar 4JUnit {

  def accountOperations() {
    Integer balance = 0
    eachIteration( 'test_account_operations' ){ tmar ->
        switch (tmar.operation) {
        case 'deposit' :
            balance = balance + tmar.amount
            break
        case 'withdraw' :
            balance = balance - tmar.amount
            break
        }
        tmar.balance = balance
    }
  }
}
```

Tmar – Dealing with scenario (Spock)

- The easiest way to deal with a scenario is to use a **/sequence** with a **switch** statement on an iteration field.
- This lets you run different business rules depending of this field value

TestDemo.test_account_operations.tmar

```
Bank account operations tests

/sequence
[operation |  amount |  balance ? |
| deposit  |    2000 |      2000  |
| deposit  |     500 |      2500  |
| withdraw |     700 |      1800  |
```

OK - Bank account operations tests
3 iteration(s) with 0 in error - 04/11/13 14:51:49

SEQUENCE

operation	amount	balance
deposit	2000	2000
deposit	500	2500
withdraw	700	1800

TestDemo.groovy

```
class TestDemo extends Tmar {

   def accountOperations() {
      Integer balance = 0
      when:
         while (tmar.hasNext()) {
            switch (tmar.operation) {
               case 'deposit' :
                  balance = balance + tmar.amount
                  break
               case 'withdraw' :
                  balance = balance - tmar.amount
                  break
            }
            tmar.balance = balance
         }
      then:
         tmar.asserts()
      where:
         tmar << getData('test_account_operations')
   }
}
```

Tmar – get current iteration information

- Tmar provides three methods for this purpose :
  - **getCurrentIterationNumber()**
  - **getIterationHeader()** which returns a map of fields where key = fieldName and value = Boolean value which is true if the field is a result field (? In the column description)
  - **getIterationValues()** which returns a map of fields where key = fieldName and value is the value of the field from the Tmar description
    - Only non-result fields are present in this Map

TestDemo.accumulation_test1.tmar

```
accumulation of values

/sequence
[ number 1 | number 2 | result ? | accumulated ? |
|     2     |     2     |     4     |     4          |
|     2     |     3     |     6     |    10          |
|     2     |     4     |     8     |    18          |
```

```
tmar.getIterationHeader() == ['number1':false, 'number2':false,
'result':true, 'accumulated':true]
```

```
tmar.getIterationValues() == ['number1':2, 'number2':2] (first iteration)
tmar.getIterationValues() == ['number1':2, 'number2':3] (second iteration)
```

Tmar – Table support

- Tmar allows you to declare a table of data in your description in order to pass contextual values to your test
- A table is introduce by the **/table** statement followed by the table name
- Header and values follow he same rules as /each table

```
/table countries
[ code  |   name          |
|  FR   |  France         |
|  US   |  United States  |
|  BR   |  Brazil         |
```

- In your test code you can obtain table values from the Tmar object of the current iteration
  - Example :

```
List countryCodes = tmar.getTableColumn("countries","code")
```

Tmar – Table support

- The following methods are available on the tmar object

```
tmar.getListFromTable ('countries') == [['code':'FR', 'name':'France'],
  ['code':'US', 'name':'United States'],['code':'BR',name:'Brazil']]
```

```
tmar.getTable ('countries') == ['code':['FR','US','BR'],
                                'name':['France','United States','Brazil']
```

```
tmar.getTableLine('countries',0) == ['code':'FR','name':'France']
```

```
tmar.getTableColumn('countries','code') == ['FR','US','BR']
```

```
tmar.getTableValue('countries',0,'name') == 'France'
```

```
tmar.getTableHeader('countries') == ['code','name']
```

```
tmar.getTableSize('countries') == 3
```

Tmar – Map support

- In addition to table, Tmar allows you to declare map in your description to give context value to your test
- A map is introduce by the **/map** statement followed by the map's name
- Header and values follow he same rules than table, the difference is than the first column is a single key

```
/map countries
[ code  |   name         |
|  FR   |  France        |
|  US   |  United States |
|  BR   |  Brazil        |
```

Tmar – Map support

- The following methods are available on tmar object

```
tmar.getListFromMap('countries') == [['code':'FR', 'name':'France'],
          ['code':'US', 'name':'United States'],['code':'BR',name:'Brazil']]
```

```
tmar.getMap('countries') == ['FR':['code':'FR', 'name':'France'],
'US':['code':'US','name':'United States'],'BR':['code':'BR',name:'Brazil']]
```

```
tmar.getMapsAsTable('countries') == ['code':['FR','US','BR'],
                                     'name':['France','United States','Brazil']
```

```
tmar.getMapLine('countries','FR') == ['code':'FR','name':'France']
```

```
tmar.getMapColumn('countries','name') == ['FR':'France,'US':'United States',
                                          'BR':'Brazil']
```

```
tmar.getMapColumnAsTable('countries','name') == ['France,'United States',
                                                 'Brazil']
```

```
tmar.getMapValue('countries','BR','name') == 'Brazil'
```

```
tmar.getMapHeader('countries') == ['code','name']
```

```
tmar.getMapSize('countries') == 3
```

Tmar – Table & list comparison

- In some cases it's interesting to check the validity of a list returned by a test method
- Tmar has four operators to compare a returned list with a table content
  - "**?**" is the returned list and **@tableName** the table content
  - With **in** and **orderedIn**, the result list has to be a subset of the referenced table
  - With **match** and **orderedMatch**, both must have the same number of lines

```
/each
[ case        |        result ?              |
|  any        |   ? in @countries            |
|  ordered    |   ? orderedIn @countries     |
|  all        |   ? match @countries         |
|  allOrdered |   ? orderedMatch @countries |
```

  - The "?" can be put at the right of the expression. In that case, with **in** and **orderedIn** operators, the referenced table has to be a subset of the result list

```
/each
[ case        |        result ?              |
|  any        |   @countries in ?            |
|  ordered    |   @countries orderedIn ?       |
```

Tmar – Table & list comparison examples

```
/table countries
[ code  |   name          |
|  FR   |  France         |
|  US   |  United States  |
|  BR   |  Brazil         |
```

```
Returned list
[ code  |   name          |
|  BR   |  Brazil         |
|  FR   |  France         |
```

? in @countries — ok

? orderedIn @countries — ko

? match @countries — ko

? orderedMatch @countries — ko

Tmar – Table & list comparison examples

```
/table countries
[ code  |   name          |
|  FR   |  France         |
|  US   |  United States  |
|  BR   |  Brazil         |
```

```
Returned list
[ code  |   name          |
|  FR   |  France         |
|  BR   |  Brazil         |
```

? in @countries ok

? orderedIn @countries ok

? match @countries ko

? orderedMatch @countries ko

Tmar – Table & list comparison examples

```
/table countries
[ code   |    name          |
|  FR    |  France          |
|  US    |  United States   |
|  BR    |  Brazil          |
```

```
Returned list
[ code   |    name          |
|  FR    |  France          |
|  BR    |  Brazil          |
|  US    |  United States   |
```

? in @countries ok

? orderedIn @countries ko

? match @countries ok

? orderedMatch @countries ko

Tmar – Table & list comparison examples

```
/table countries
[ code  |   name          |
|  FR   |  France         |
|  US   |  United States  |
|  BR   |  Brazil         |
```

```
Returned list
[ code  |   name          |
|  FR   |  France         |
|  US   |  United States  |
|  BR   |  Brazil         |
```

? in @countries ok

? orderedIn @countries ok

? match @countries ok

? orderedMatch @countries ok

Tmar – Table & list comparison examples

```
/table countries
[ code  |   name          |
|  FR   |  France         |
|  US   |  United States  |
|  BR   |  Brazil         |
```

```
Returned list
[ code  |   name          |
|  FR   |  France         |
|  BR   |  Brazil         |
```

? in @countries ok

@countries in ? ko

Tmar – Table & list comparison examples

TestDemo.testCountries.tmar

```
Test matching countries

/table countries
[ code  |   name            |
|  FR   |  France           |
|  US   |  United States    |
|  BR   |  Brazil           |


/each
[ case        |       result ?            |
|  any        |  ? in @countries          |
|  allOrdered |  ? orderedMatch @countries |
```

TestDemo.groovy

```
def matchCountries() {
   when:
     tmar.result =  [[ 'code','name'],
                     ['BR','Brazil'],
                     ['FR','France']]
   then:
     tmar.asserts()
   where:
     tmar << getData('testCountries')
}
```

EACH

case	result
any	**result in @countries** ***all reconciled lines*** code | name FR | France BR | Brazil
allOrdered	**result orderedMatch @countries** ***result on all @countries lines*** code | name FR | France US | United States BR | Brazil ***unreconciled lines in result*** code | name BR | Brazil FR | France

Reconciled line

Unreconciled line

Unordered reconciled line

Tmar – Table & list comparison, returned list format

- For maximum flexibility, Tmar allows for three multi-dimensional formats as return value :
  - List of List, with header columns name in the first list and lines values in the following

```
tmar.result = [['code','name'],
               ['FR','France'],
               ['US','United States'],
               ['BR','Brazil]]
```

  - List of Map, with a Map of fields name and value for each line

```
tmar.result = [['code':'FR','name':'France'],
               ['code':'US','name':'United States'],
               ['code':'BR','name':'Brazil]]
```

  - Map of List with a Map of columns with the list of all the column's values

```
tmar.result = ['code':['FR','US','BR'],
               'name':['France','United States','Brazil']]
```

Tmar – Table & list comparison, null and empty

- If a test return a null object the result will always be an error
- If the test return an empty object the result will be :
  - an error for the “macth” and “orderedMatch” operators
  - A success for the “in” and “orderedIn” operators

Returned	Expression	result
null	? in @countries	ko
null	? orderedIn @countries	ko
null	? match @countries	ko
null	? orderedMatch @countries	ko
[ ]	? in @countries	ok
[ ]	? orderedIn @countries	ok
[ ]	? match @countries	ko
[ ]	? orderedMatch @countries	ko

Tmar – include files

- Tmar "includes" let you share Tmar descriptions between Tmar files
- Include files are introduced by the **/include** statement and Tmar supports nested include files
  - For example :

TestDemo.multiply_test1.tmar

```
Multiplication test

/include countries

/each
[ number 1 | number 2 | result ? |
|   2    |   2    |    4    |
|   2    |   3    |    6    |
```

countries.tmar

```
/table countries
[ code   |    name          |
|  FR    |   France         |
|  US    |   United States  |
|  BR    |   Brazil         |
```

```
Multiplication test

/table countries
[ code   |    name          |
|  FR    |   France         |
|  US    |   United States  |
|  BR    |   Brazil         |

/each
[ number 1 | number 2 | result ? |
|     2    |     2    |     4    |
|     2    |     3    |     6    |
```

Tmar – Share context

- Tmar lets you load a Tmar description file at the class level
  - By convention this file will be named **<className>.tmar** and has to be in an includes subdirectory
- All tables declared in this file will be seen from all iterations of all test cases of this class
- To load this description file you have to call the method **loadSharedContext**() in the method **setupTest**() for JUnit/TestNG **or setupSpec**() for Spock

```
class TestDemo extends Tmar4JUnit {

  void setupTest() {
    def mySharedContext = loadSharedContext()
  }
  test methods ...
}
```

- The returned object mySharedContext has all the needed methods to get data from the shared context tables (getTable, getColumns, getValues… see table and Map support)
  - Those methods are useful if you need to initialize your own shared objects at this point with data coming from the Tmar shared context

Tmar – Iteration's context

- Each iteration may have a different execution context
- Fields in /each or /sequence iteration may be of different types. For example, the following description is valid

```
Bank account operations tests

/sequence
[operation |  amount |  balance ? |
| deposit  |    2000 |     2000   |
| deposit  |     500 |     2500   |
| withdraw |     700 |     1800   |

/sequence
[ operation    |    account    |    date    | balance ? |
| information  |  "9700990991" | 01/01/2013 |     900   |
```

- Is that case tmar.getIterationHeader() gives the available fields

```
tmar.getIterationHeader() == ['operation':false, 'amount':false, 'balance':true]
```

```
tmar.getIterationHeader() == ['operation':false, 'account':false, 'date':false,
'balance':true]
```

Tmar – Iteration's context

- Tables content may also be different between iterations
- Indeed, iterations only see table content defined before it's own declaration in the description file. A table content can be fed several times so that the table content is progressively available.

```
Bank account operations tests

/table valid account
[account number | owner       |
| "9700990991"  | John Smith |

/sequence
[ operation     |    account     |    found ? |
| information   | "9700990991" |     yes    |
| information   | "5700550552" |      no    |

/table valid account
[account number | owner        |
| "5700550552"  | Edith Wilton|

/sequence
[ operation     |    account     |    found ? |
| information   | "5700550552" |     yes    |
```

Tmar – Override value transformation

- By default, Tmar tries to parse Integer, Double and Boolean types.
- To override this behavior you can use the method **setTransformValue**() in the method **setupTest**() for JUnit/TestNG **or setupSpec**() for Spock
- For table, or a table column, you are able to set a closure that overrides the default transformation

```
class TestDemo extends Tmar4JUnit {

  void setupTest() {
    setTransformValue('countries','code'){ value ->
      return "ISO-$value"
    }
  }

  test methods ...
}
```

Will returned
"ISO-FR"
"ISO-US"
"ISO-BR"

Obviously this method also allows to change value type

Tmar – working directories & files

- Tmar working directories are defined in the tmar.config
  - This file has to be in a tmar directory in the test project resources directory

```
myProject/core/src/test/resources/tmar/tmar.config
```

- Default tmar.config content

```
<TMAR>
    <CONFIG>
        <DESCRIPTION_DIRECTORY type="FILE" name="core/src/test/resources/tmar/description"/>
        <RESULT_DIRECTORY name = "target/tmar-test-results"/>
        <REPORT_DIRECTORY name = "target/tmar-test-report"/>
    </CONFIG>
</TMAR>
```

- Tmar's directories
  - DESCRIPTION_DIRECTORY : Tmar test description files
    - **className.testName.tmar**
  - RESULT_DIRECTORY : json execution result files per iteration
    - **className.testName.iterationNum.result**
  - REPORT_DIRECTORY : html test execution reports
    - **className.testName.iterationNum.html (test report, one for each .tmar file)**
    - **TmarIndexReport.html (global index of the test html reports)**

Tmar – Description directory

- Tmar provides great flexibility in organizing description and include files in the description directory and subdirectories
- The description directory may have as many subdirectories as necessary
- Description files can be anywhere in the tree
  - When a description file is call by a test method only the test name is given to Tmar
  - Tmar build the full name with the class name and a .tmar extension
  - Then Tmar scans the tree to find the description file
- With this approach the functional team may reorganize the description directory without any impact on the test code
- By convention
  - Include files have to be in subdirectories named “includes”.
  - Images files have to be in subdirectories named “images”.
  - Documents files have to be in subdirectories named “documents”.

include, image and document files can be spread among as many directories as needed in the description tree.

Tmar description tree example

```
> Description
   > First chapter
      > Calculation test cases
         > Simple test cases
         > advance calculations
         > includes
      > business test cases
         > includes
         > documents
   > Second chapter
      > includes files
      > images
      > documents
```

Tmar – Index.html report

- Tmar build an index.html file report to index each Tmar test report
- The project description files directories tree is used to build this report
- For each directory the report lists the test reports and point out their status
  - No error
  - At list one test iteration is in error
  - Tmar description file found but not executed
- Each test result line is a link to the detail test report

Tmar – Customize index report header

- Default index.html report header is a Jspresso header

- tmar.config file allow to define your own logo and title

```
<TMAR>
    <CONFIG>
        ...
        <TITLE name="My company project"/>
        <LOGO name = "myCompanyLogo.jpeg"/>
        ...
    </CONFIG>
</TMAR>
```

Tmar – <name>.chapter.tmar

- A challenge for test result report is to make the link with project documentation.
  - The tags #url and #documents allow to make this kind of link from the tests report files, but often the perimeter of a documentation is much bigger than a test description file itself.
- To address this issue Tmar has special description files with the name finished by **chapter.tmar**
  - Those file are tmar files but without /table, /each, /sequence or /include statement. Just formatted text, #image, #url and #documents
  - Each time Tmar find a chapter.tmar file in a directory, it include his content in the index.html file.
  - Then the directory name become a link in the index.html file which show/hide the content of the chapter.tmar file
  - This way your able do make descriptions and link to documentations at a chapter level for a group of Tmar test description.

Tmar – Markdown formatting language

- To allow text formatting, TMar support the Markdown formatting language (thanks to txtmark processor)
- Due to syntax overlap with Tmar, link and image Markdown syntax is not supported. You have to use **#url**, **#image** and **#document** Tmar tags instead.

- Markdown syntax example :

```
# Big Title (HTML H1)
##Other title (HTML H2)
```

Big Title (HTML H1)

Other title (HTML H2)

```
*Italic*, **bold**
```

Italic, **bold**

Tmar – Manually running the global test report

- Each time you run a test, the corresponding html report is generated
- Generally, your global html report is generated by the maven build or by the Tmar Eclipse plugin
- However, whenever needed, it's very simple to generate it by running this simple test

```
class TestTmarReport extends GroovyTestCase {

  void testTmarReport() {
    TmarReport tmarReport = new TmarReport()
    tmarReport.generate()
  }
}
```

Tmar – Plugin

Eclipse File Edit Navigate Project Tmar Window Help

Tmar – sample/src/test/resources/tmar-descriptions/Password strength/PasswordTest.test.tmar – Eclipse – /Development/repositories/workspace-juno-camtasia

New Tmar specification... Open Tmar method Run Tmar tests... Quick Access Java Tmar

Tmar Navigator

- sample
  - tmar-descriptions
    - images
    - Issue Tracking
    - Java Map specification
      - Algorithm for map – get
      - Algorithm for map – put
    - Password strength
      - Password strength checke
    - TMAR TEST
  - tmar-test-report

Outline

- Password strength checker
  - /image password.png
  - /each
    - password, A ?, B ?, C ?, D ?, score ?,

PasswordTest.test.tmar

```
Password strength checker

Password strength is a measure of the effectiveness of a password in resisting guessing anc
In its usual form, it estimates how many trials an attacker who does not have direct access t
on average, to guess it correctly.

_The strength of a password is a function of length, complexity, and unpredicta
/image password.png

_What is a strong password?

With the above mentioned in mind, I should stress that a sufficiently strong password does
all of the rules below, but merely a few will do. Consider the following rules to each add an
password is (so that 0 point is the weakest, and 4 is the strongest). UI-patterns.com defines

 * Rule A : Has more than 8 characters
 * Rule B : Contains both small and uppercase letters
 * Rule C : Contains at least one numerical character
 * Rule D : Contains special characters like ~!@#$%^&*_-+='\(){}:;"'<>,.?/() etc.

_This would result in 5 levels of password strength depending on how many c

Here are some examples :

/each
[ password | A? | B? | C? | D? | score? | strength? |
| DEMO | false | false | false | false | 0 | weak |
| password | false | false | false | false | 0 | weak |
| demo1 | false | false | true | false | 1 | weak |
| "123" | false | false | true | false | 1 | weak |
| Password | false | true | false | false | 1 | weak |
| abcdefghi | true | false | false | false | 1 | weak |
| Password | false | true | false | false | 1 | weak |
| $ | false | false | false | true | 1 | weak |

Now some more  FAIR paswords :

| Demo1 | false | true | true | false | 2 | fair |
```

PasswordTest.test.html

OK - Password strength checker
17 iteration(s) with 0 in error - 07/01/14 23:09:29 - PasswordTest.test

Password strength is a measure of the effectiveness of a password in resisting guessing and brute-force attacks.
In its usual form, it estimates how many trials an attacker who does not have direct access to the password would need, on average, to guess it correctly.

The strength of a password is a function of length, complexity, and unpredictability.

Password strength:	Weak
Password strength:	Fair
Password strength:	Good
Password strength:	Strong

What is a strong password?

With the above mentioned in mind, I should stress that a sufficiently strong password does not necessarily need to fulfill all of the rules below, but merely a few will do. Consider the following rules to each add an extra point in how strong the password is (so that 0 point is the weakest, and 4 is the strongest). UI-patterns.com defines a strong password when

Writable | Insert | 8 : 20