Unit testing avoiding regression through code coverage in a Continuous Integration infrastructure

Edoardo Ghini

July 21, 2016

Supervisor: **Paolo Merialdo** .....

Assistant Supervisor: **Domenico Lupinetti**

Academic Year: **2015/2016**

Dipartimento di Ingegneria dell'Università degli Studi Roma Tre

Abstract

Bugs are onerous to discover and fix, in particular the fixing cost increment exponentially with the increase of the time gap between the bug introduction and its discovery.

The question: How can we minimize debugging cost ?

The answer: We can minimize this cost through deep and granular testing activity.

Credits

I would like to thank everyone who supported me during my academic path : family, friends, professors and colleagues.

Without their help I would never be able to succeed.

Contents

List of Figures

Listings

Part I

Introduction

1 Thesis on internship

At the very beginning I was wondering where I could begin my apprenticeship. The opportunity showed up in spring during the second half-year.
My courses where about to begin when a professor of my university put me in touch with Translated Company.

Subsequently I scheduled a job interview. It was the first of my life.
I found the company very exciting and soon I accepted the intern ship proposal. The assignment was to write tests about their web application.

1.1 Matecat

It is a web application that offers useful tools for text translations, it is free and available for everyone.

For example, freelance translators could work faster because this application provides services of automate translation, through machine translation engine and a huge translation memory that grows day after day.

2 Company Requirements

Today IT companies are required to be super-competitive. They will have possibilities of success only if they care about some cardinal principles.

2.1 Sustained Pace

The common saying "*If you fall behind you won't come back !*" can't be more appropriate.
In fact competitiveness is huge and keen in this field.
So the only way to hit the big time is to keep up with changes, with new technologies and with rising innovative ideas.
It takes shape of a race against competitors.

2.2 Harmonious Team

If a company has reached solidity and had expanded itself, this means that it had found a good balance between team members.
Consequentially it had built a solid hierarchy in which each component feels self-confidence.
In this scenario each professional figure has a specific role and together they will be able to carry forward the company.

2.3 Necessities for a web application

There are also essential activities in which companies have to focus on if their business concerns web applications.
These shown below are a couple of actions that the IT company carries out every day.

- Periodically releases of new features with patches and updates.
- Support provision to consumer issues dedicating time and human resources for bug-fixing.

Certainly there are other activities that are worth to mention but I had been in contact only with these two.
In fact there are many commercial and managerial aspects that I am neglecting on purpose.

2.4 No room for testing

Sometimes it happens that because of the effort spent chasing features aforementioned, often other things just as much important are neglected.

Make tests on the code is one of the activities that often don't find place in the process or at least isn't privileged among the other activities scheduled.
It could require the same effort or even more than raw coding, because it could be boring and repetitive.

So testing is considered unattractive from developers.
Unfortunately lack of appropriate ***code coverage*** lead to fear of changes and more likely to ***regression***.

2.4.1 Code coverage

Code coverage is an indicator of how much of your code is tested.
This information is obtainable running tests and checking what is the percentage of code that is being exercised.

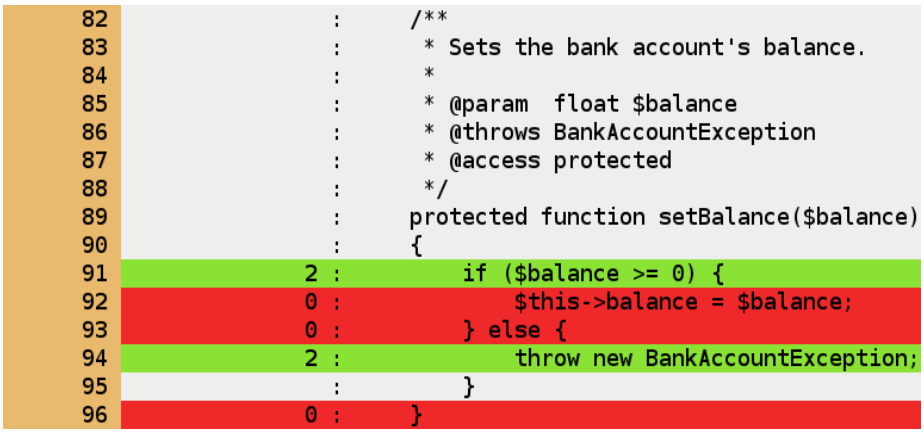

Figure 1: Example of function code coverage.

Green lines showed in figure 1 are successfully covered by tests, instead red ones haven't been exercised at all.

Dijkstra said:
"Testing never proves the absence of faults, it only shows their presence".

2.4.2 Regression

Is defined regression when after a certain event, that imply whatever modification of the code, a feature begins to not work properly or as expected.

When regression happens two main solutions could be undertaken :

- Assuming that bug has been introduced in the lines just modified or added, it would be necessary undoing changes and rewriting from scratch.

- Otherwise it would be necessary starting a bug research, assuming that a bug has been introduced far before the last modification.

Both this patterns are expensive in terms of time and resources, therefore if regression happens frequently there will be uncertainty toward future contributions.

Finally in order to avoid fear of changes and the consequent paralysis, it is recommended to prevent regression through code coverage.

Part II

Development

3 My task in the company

I had to certificate the stability of code in the application.
Substantially this is achieved bringing the code coverage percentage from zero to an acceptable percentage.

This objective can be reached only with a deep and granular testing activity.
In order to do this I was supposed to verify all the firms of functions.
I had to face only a side of the problem because my area of expertise was only back-end.

Otherwise I would have had to consider all the front-end implications:
I would have to follow different logics for testing, for example behavioural tests instead of unity tests.

3.1 Differences between Front-end and Back-end

These are two sides of the same coin. Each is strongly correlated with each other. They are layers constantly exchanging informations.

- Front-end developers focus on user experience because their tasks often are designing the User Interface.

They have to think which are the most important interactions between the user and the web interface.
They pursue a minimalistic style and rendering a smooth and appealing UI. If the software development for applications was an iceberg, front-end development could be linked with the visible part of it like in fig. 2.

Figure 2: Portion of an iceberg above sea level

- On the contrary Back-end developers work in the shadows.

It's not their concern to satisfy the user at least not with brilliant design or cute effects.
They must focus themselves on algorithms. They shouldn't waste CPU time and computational power, both are key factors.

Their state of art can be reached through writing smart algorithms that outperform those written before. If the software development for applications was an iceberg, back-end development could be the submerged part of it, as it is shown in fig. 3.

Figure 3: An entire iceberg

4 First steps

At the beginning I had a rough impact.
It was difficult at the beginning to get into the habit.
There are some automatisms that people acquire working for years in big companies.
For example how to cherish their own personal space during working sessions.
Because they are not alone and they often interact with colleagues.

I was introduced to Matecat with inductive approach, I came up beside a project manager for a couple of days and she explained me some managerial aspects.
After this I followed with reading the support guide and questioning my tutor about some Matecat functional aspects.

At this point I could begin testing.

4.1 Fresh technologies

There where a lot of things that leaved me astonished:
If someone ask me to explain or at least reproduce step by step the sequence of operations made by my colleagues on my machines to set up of my working

environment, I won't figure it out.

The configuration was very complex from my point of view. But it's also true that there is clear documentation provided from the community.

4.1.1 LAMP Configuration

This set up is very common in web application scope. It consist of four pieces:

Linux The development of Linux is one of the most prominent examples of free and open-source software collaboration.
Typically, Linux is packaged in a form known as a Linux distribution that includes the Linux kernel, supporting utilities and libraries.
Linux community can be described as the cradle of open source philosophy.

Apache The Apache HTTP Server is the world's most used web server.
It is developed and maintained by an open community of developers.
It allow users to set up virtual hosts on their personal computers, it is very powerful tool indeed.

MySql It's a relational database used for persistence purpose.
MySql manages **CRUD operations**, C stands for CREATE, R stands for READ, U stands for UPDATE and D stands for DELETE.

PHP This is a recursive acronym that stands for Php Hypertext Preprocessor.
This is a language based on multidimensional associative arrays.
Commonly it is used together with JavaScript that is a front-end programming language.

Anyway I tried on my own to solve problems produced by my scarce knowledge of operative systems.
When all the tricks where made I could watch all these bricks working together with fantastic outcomes.

4.1.2 New main programming language

I had to confront with new syntax and constructs of PHP 5.
Throughout my academic studies I was introduced to the basis of the computer science. Programming languages arise from the same root, they differ from each other by little.
So I worked hard to apply the concepts learned studying other languages, in this scenario.

5 Three months time

I'm going to talk about the centre of my work experience in which my background experience has been enriched mainly.
Below there are some problems that I encountered followed from solutions that I adopted :

5.1 Team Communication

Usually Company **CEO** (chief executive officer) has to convey the objectives that have to be persecuted.
If they are not clear , there will be episodes in which team members are in disharmony and there will be also contentions between colleagues about the direction that should have been taken.

If these events cause the a step back and a rejection of some work already completed, they could generate a sense of unfulfillment and frustration in the members involved.
This is not something that had concerned me directly, however I was already working when the question was debated.

5.1.1 Problem

Translated was founded in 1999 and in these last years it has acquired more and more employers part of which are developers.
When I arrived this number hit roughly sixteen, because of this increment the communication between members was at a historical low.

5.1.2 Solution

After a few weeks since my arrive, a meeting was scheduled with the aim of define the right track.
In this occasion dissatisfaction was expressed from team members and we intended to improve cooperation and team awareness in order to reduce the communication gap and avoid misunderstandings.

To achieve that it was employed one of the most important principles taken from Agile Manifesto : *Scrum*.
It consists in scheduling weekly meetings in order to discuss of every decision made, of problems encountered and of solutions proposed.

Week after week I observed an enhancement of the interactions between team members. There was being more participation.

There was been much focus on the knowledge of who is doing what and how long it takes. In fact this knowledge is the key for a productive task assignment.

5.2 Code Testability

Starting from the assumption that sooner or later the code written will be tested, developers should write code easy to be tested.

5.2.1 Problem

Visibilities of variables, constants and some methods in classes are usually restricted only to these classes through visibility modifiers
(*protected* , *private*).

```
1  class Universe {
2          private $blackHole= "black hole";
3          /**
4        * @var Star
5        */
6          protected $neutronStar;
7          public $comet= "Halley comet";
8          public function explore(){...}
9          private function getSpectre(){...}
10 }
```

Code 1: Example of a Class with various visibilities

In this example only the variable *comet* and the method *explore()* in code 1 are visible from external objects.
A good practice is to maintain visibilities to objects through dependency injection, it consists in feeding objects to methods as parameters instead of letting methods to instantiate these objects.

For didactic purpose the object *telescope*, in code 2, is visible, for parameter, to the object *neutronStar*.

```
1  class Universe {
2          public function explore(){
3                  $telescope= new Telescope();
4                  return $this->neutronStar->getActualLuminosity($
                       telescope);
5          }
6  }
7  class Star {
8          private $ID;
9          public function getActualLuminosity( $telescopeParam ){
10                 $return $telescopeParam->getLuminosity($this->ID);
11         }
12 }
```

Code 2: Two strongly correlated methods

The injection of the object *telescope*, in the argument of *getActualLuminosity* method, in line 4 in code 2, is a sign of good testability.

But in code 3 the code testability would be worse than before:

```
1  class Universe {
2          public function explore(){
3                  return $this->neutron_star->getActualLuminosity();
4          }
5  }
6  class Star {
7          private $ID;
8          public function getActualLuminosity(){
9                  $telescope= new Telescope();
10                 $return $telescope->getLuminosity($this->ID);
11         }
12 }
```

Code 3: Another possible implementation of methods in code 2

In this case there are less lines of code and it could seem cleaner and more elegant.
But if i would like to test the method *getActualLuminosity* making it work with a different object, for example DummyTelescope, in the first case I will be able to do this passing a different parameter in line 4 but in the second case I can't.

It means that in the second case there is a strong dependence between classes *Star* and *Telescope*, this is inadvisable.

The question is how how classes with strong dependences and private and protected visibilities can be tested efficiently ?

5.2.2 Solution

The key for the problem could be found bypassing dependency and visibility constraints. While you are testing a class, you can broke them.

There are whole frameworks devoted to testing activity. Object Oriented Programming Languages offer some very useful tools:

Reflection *Reflection* is the ability of computer program of introspect itself at runtime.
It is commonly used for observe and modify variable values during the execution.
Sometimes it is employed to change even the code of executing computer programs.
Thanks to its potentialities, I heavily took advantage of it writing tests.

I managed to bypass visibility constraints because when I had reflected a class I could have access to all the protected and private variables and methods.

Accordingly with previous examples Code 4 is one of possible usages of *reflection*:

```
1  class SampleReflection {
2  protected $reflectedClass;
3
4  public function reflectUniverse(){
5   /**
6    * reflecting universe class
7    */
8    $this->reflectedClass = new ReflectionClass(new Universe());
9   /**
10   * obtaining properties and letting them be accessible
11   */
12   $private_property = $this->reflector->getProperty('blackHole');
13   $private_property->setAccessible(true);
14
15   $protected_property = $this->reflector->getProperty('neutronStar');
16   $protected_property->setAccessible(true);
17
18   $private_function = $this->reflector->getMethod('getSpectre');
19   $private_function->setAccessible(true);
20  /**
21   * getting or modifying values of reflected properties and invoking
         a reflected method
22   */
23   $private_property->setValue($this->reflectedClass, "whiteHole" );
24
25   $protected_property->getValue($this->reflectedClass );
26
27   $private_function->invoke($this->reflectedClass );
28   }
29 }
```

Code 4: This class reflect some properties of another class

Problems due to visibility constraints can be easily solved with *reflection*.

Mock Objects *Mock Objects* are very powerful to help handling outside dependencies.
They are a specific application of *reflection* commonly used while testing.

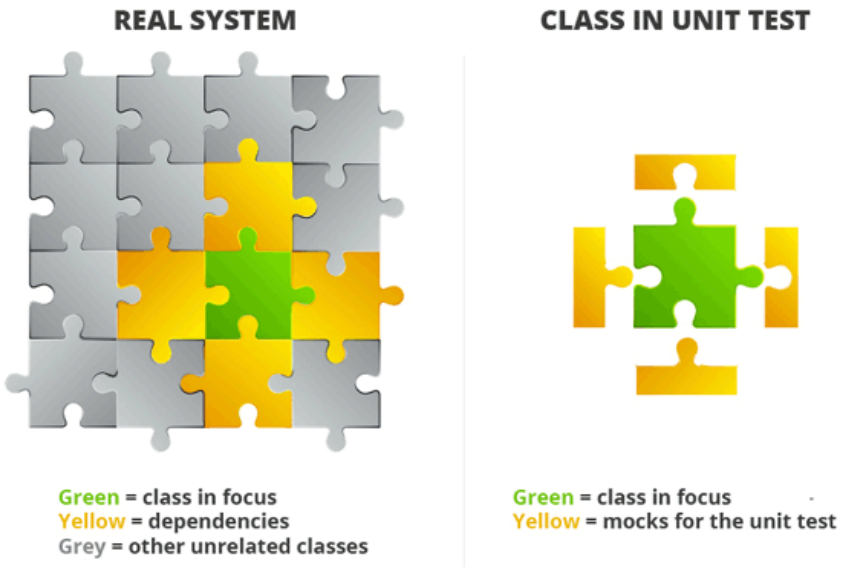

Figure 4: Example of testing through mock objects

This technique basically involves replacing the actual object with a fake, or 'mock', object that we fully control.
After that, all dependencies on outside systems or on code, that haven't to be tested, prove to be removed as it is shown in fig.4 .
The creation of a *Mock Object* can be performed with different results, accordingly to developer's needs.
Mocking a class, with methods inside, could result in a *mock object* with both **stub methods** and **mock methods**.
A *stub method* is a method contained within a mock object that returns null by default, but allows its return value to be easily overridden.
Instead a *mock method* does the exact same thing its original method would. In other words any code that is in the method you are mocking will actually run as the original one.

Basically *mock methods* became useful to me for when I needed to do some assertions on the behavior of the method.
These assertions could be that specific parameters are passed to the method or that the method is called exactly a specific number of times or not at all.

```
1  class SampleMock {
2  protected $reflectedClass;
3
4  public function mockUniverse(){
5
6    $mock_of_universe= $this->getMockBuilder('\Universe')->
         setConstructorArgs(array())->setMethods(array('explore'))->
         getMock();
7
8    $mock_of_universe->expects($this->exactly(1))->method('explore')->
         willReturn("I_am_endless");
9    }
10 }
```

Code 5: Example of a *mock object* creation and usage

The mock object just created has one *stub method* which is explore() and it will return the string specified in line 8 of code 5.
Any other method within that class other than that will run their original code.

Mocking objects is the solution for dependencies constraints but if dependency injection rule has been broken, both *reflection* and *mocks* wouldn't be enough to overcame the problem of scarce testability.

5.3 Code Versioning Management

When a group works to the same project, each member should feel comfortable working on his own environment.

When a member has completed a task he should be able to ”contribute” in the project so the others could benefit from his effort and gave him feedbacks.

5.3.1 Problem

When a group works to the same project often their contributions could produce conflicts, for example because two or more team members have modified the same part of the project and the modifications can't be *merged* automatically.

In this circumstances there would be an interruption of development that could cost precious time spent to resolve conflicts.

5.3.2 Solution

One very powerful element that allow a team to work together simultaneously on the same front is **Git**, it is widely used from developers.
Git is a version control system that records changes to a file or set of files over time so that specific versions can be recalled later.
It allows to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem or who introduced an issue and when.

Git is organized with branches like in fig.5, every branch is a work environment and it has an head. The heads are navigable from team developers.

All the branches before or after have to be merged in the main branch: ”*master*”.

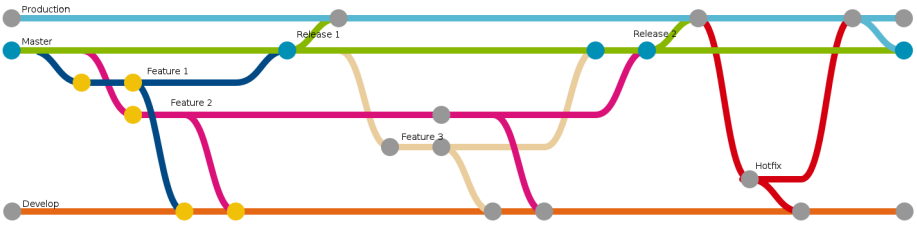

Figure 5: Git branches commonly used from developers

Using *Git* means that if you screw things up or lose files, you can easily recover.
Git has three main states in which files can reside in: **committed**, **modified**,

and **staged**.

Committed means that the data is safely stored in your local database.

Modified means that you have changed the file but have not committed it to your database yet.

Staged means that you have marked a modified file in its current version to go into your next commit snapshot.
I had to learn basic prompt line commands, which are shown in code 6, to change states of my contributions to *Matecat* project.

```
1  $ git clone "www.gitHub.com/matecat"; // I download the project
       repository.
2  $ git branch matecat_unitTest ; // I created a branch where I will
       test some classes.
3  $ git checkout matecat_unitTest ; // I arranged the HEAD on the just
       created branch.
4  $ git status ; // I always check if there is any modified file.
5  $ git add example_test.php // I changed the state a file from
       modified to staged.
6  $ git commit -m "Completed_a_sample_test" ; // I transformed staged
       files in committed files followed by a brief comment.
7  $ git checkout develop ; //I swapped branch to develop arranging my
       HEAD from matecat_unitTest to develop.
8  $ git pull origin master; //I updated the branches stored on my
       machine with all the modifications committed on the remote
       branch.
9  $ git checkout matecat_unitTest ; //I brought back my HEAD on the
       branch containing my committed files.
10 $ git merge develop ; //I merged my changes with changes made from
       the other developers and eventually I resolve conflicts.
11 $ git push my_matecat matecat_unitTest ; //I transferred all changes
       successfully merged to my remote branch.
```

Code 6: An example of my typical git work-flow

After all this steps I always complete my contribution compiling a *pull request* through **GitHub.com** and I wait that it will be accepted from the chief developer.
Git and GitHub allow an easy management of conflicts or even their avoidance.

5.4 Unfathomable Project

I had to learn how the pillars of the application work, to be able to write down tests about it.
I acquire comprehension of basic behaviours essentially watching the application working.
As external observer I witnessed step by step the logic of methods, inputs and outputs of functions and also the class hierarchy.

5.4.1 Xdebug

A debugger is a computer program suited for testing and bug-fixing other programs.
It analyse the code running it on an instruction set simulator too, this technique allows to halt the computation in chosen breakpoints and observe a snapshot of stack, registers and variables.

It helps a lot in code comprehension and bugs detection.
Xdebug in particular is a debugger available for PHP and offers the following informations:

- stack and function traces
- parameters and returns of functions
- memory allocation
- protection from infinite recursion
- last but not least code coverage analysis

Xdebug allows to set breakpoints in lines of code in which we want to stop the CPU execution.
After the execution was interrupted, I was able to deeply examine all the values of variables and also all the input or output of functions.

5.4.2 Personal Matecat Examination

I will follow the same order of classes that I come after when I was testing.

Cat-Utils I started from one of the longest classes of the whole project, it provides core utilities like several format conversion algorithms.

In particular there are some formats like *xliff*, that are very common in translation files.
In code 7 there are some special characters (< , >) in a *xliff* string that have to be replaced in order to wrap this string in a HTTP request.

```
1          /**
2   * @group regression
3   * @covers CatUtils::rawXliff2view
4   * this battery of tests sends one string in input as $source_segment
        to CatUtils::rawXliff2view method and
5   * verifies that the output is equal to the $expected_segment.
6   * User: dinies
7   * Date: 30/03/16
8   * Time: 18.05
9   */
10  class RawXliff2ViewTest extends AbstractTest
11  {
12      protected $source_segment;
13      protected $expected_segment;
14      /**
15       * @group regression
16       * @covers CatUtils::rawXliff2view
17       */
18      public function test_raw_X_liff2view1()
19      {
20          $this->source_segment = <<<LAB
21  <g id="1">[AH1]</g><g id="2">Is fold &amp; crease the same??</g>
22  LAB;
23          $this->expected_segment = <<<LAB
24  &lt;g id="1"&gt;[AH1]&lt;/g&gt;&lt;g id="2"&gt;Is fold & crease the
        same??&lt;/g&gt;
25  LAB;
26
27          $this->assertEquals($this->expected_segment, CatUtils::
                rawxliff2view($this->source_segment));
28      }
29
```

Code 7: Test about the class CatUtils

Database I followed with another very important class that manage **CRUD** operations with database.

Database.php is a singleton class. It means that it is allocated when the application runs and it has only one instance at time.
Singleton classes are rare because they come handy only in some circumstances. For example when singleton methods are invoked from a great number of classes, which is our case indeed.
Therefore *Database.php* has static methods that have global visibility.

Code 8 concerns with an UPDATE operation.

```
1          protected $reflector;
2      protected $property;
3      /**
4       * @var Database
5       */
6      protected $database;
7      protected $affected_rows;
8                                                          /**
9       * Equivalent query executed from update :
10      *    $query = "UPDATE Persons
11      *    SET PersonID=(678432)
12      *    WHERE PersonID=(475144);";
13      * @group regression
14      * @covers Database::update
15      * User: dinies
16      */
17     public function test_update_simple_table()
18     {
19         $this->database = Database::obtain(INIT::$DB_SERVER, INIT::$
               DB_USER, INIT::$DB_PASS, INIT::$DB_DATABASE );
20
21
22         $this->reflector = new ReflectionClass($this->reflectedClass)
               ;
23         $this->property = $this->reflector->getProperty('
               affected_rows');
24
25         $table = "Persons";
26         $data = array('PersonID' => 678432 );
27         $where = "PersonID␣=␣475144";
28         $this->database->update($table,$data,$where);
29         $this->affected_rows= $this->property->getValue($this->
               database);
30
31         $expected = array(0 => array("PersonID" => 678432), 1 =>
               array("PersonID" => 890788));
32         $actual = $this->database->query($this->sql_read)->fetchAll(
               PDO::FETCH_ASSOC);
33
34         $this->assertEquals($expected, $actual);
35         $this->assertEquals(1, $this->affected_rows);
36     }
```

Code 8: Test about the class Database

Engines Translation engines provide translations of strings if there are matches belonging to previous translations of very similar strings.

This is the base concept of *Matecat*. In particular the company focuses on a main engine, *MyMemory*.
It gains, day after day, huge amounts of users translations and it improves its accuracy giving translation matches.

MyMemory This test verifies the behaviour of a get request for the translation of a string.
The one in code 9 is one of the most performed actions in the whole application.

```
1           class GetMyMemoryTest extends AbstractTest{
2           /**
3        * @var EnginesModel_EngineStruct
4        */
5       protected $engine_struct_param;
6       protected $others_param;
7       /**
8        * @var Engines_MyMemory
9        */
10      protected $engine_MyMemory;
11      protected $config_param_of_get;
12      protected $reflector;
13      public function setUp()
14      {
15          parent::setUp();
16          $engineDAO        = new EnginesModel_EngineDAO( Database::
                obtain(INIT::$DB_SERVER, INIT::$DB_USER, INIT::$DB_PASS,
                 INIT::$DB_DATABASE ) );
17          $engine_struct= EnginesModel_EngineStruct::getStruct();
18          $engine_struct->id = 1;
19          $eng = $engineDAO->read( $engine_struct );
20          $this->engine_struct_param = @$eng[0];
21          $this->config_param_of_get= array(
22              'translation' => "",
23              'source' => "it-IT",
24              'target' => "en-US",
25              'email' => "demo@matecat.com",
26          );
27      }
28      /**
29       * @group regression
30       * @covers Engines_MyMemory::get
31       */
32      public function test_get_segment_italian_with_id_user_initialized
            (){
33
34          $this->engine_MyMemory = new Engines_MyMemory($this->
                engine_struct_param);
35          $this->config_param_of_get['segment']="'LAmministratore␣
                inserisce␣titolo,␣anno␣di␣produzione␣e␣codice␣univoco␣
                del␣nuovo␣film.";
```

```
36          $result = $this->engine_MyMemory->get($this->
                config_param_of_get);
37
38          $this->assertTrue($result instanceof
                Engines_Results_MyMemory_TMS);
39          $this->assertTrue(property_exists($result,'matches'));
40          $this->assertTrue(property_exists($result,'responseStatus'));
41          $this->assertTrue(property_exists($result,'responseData'));
42      }
43  }
```

Code 9: Test about the method get() in MyMemory class

Structs Structures are classes that carry out the management of data obtained from databases.
In *Matecat* there is usually a one to one correspondence between struct classes and database tables.
Therefore if database tables have fields, also structures have them.
In *PHP*, structs can be multidimensional arrays with keys and relative values; there is an example in code 10.

```
1           class ToStringTest extends AbstractTest{
2       /**
3        * @var EnginesModel_EngineStruct
4        */
5       protected $engine_struct_param;
6
7       public function setUp(){
8           parent::setUp();
9           $this->engine_struct_param = new EnginesModel_EngineStruct();
10          $this->engine_struct_param->id = 999 ; //sample value
11          $this->engine_struct_param->name = "Moses_bar_and_foo";
12          $this->engine_struct_param->description = "Machine␣
                translation␣from␣bar␣and␣foo.";
13      }
14
15
16      /**
17       * It checks if the string of summary useful for confrontations
                between instances of engines is being created correctly.
18       * @group regression
19       * @covers EnginesModel_EngineStruct::offsetUnset
20       */
21      public function test_toString_field(){
22          $expected_string= '999Moses_bar_and_fooMachine␣translation␣
                from␣bar␣and␣foo.';
23
24          $this->assertEquals($expected_string,$this->
                engine_struct_param->__toString());
25      }
26  }
```

Code 10: Test about the method toString of an Engine Struct

DAOs *Database Access Objects* are bridges between databases and structures. They communicate with the local database and they make the data available to functions through structs.

The test, showed in code 11, inserts a table entry in database and then it verifies that it was inserted.

```
1  class GetByUidUserTest extends AbstractTest{
2      /**
3       * @var Users_UserDao
4       */
5      protected $user_Dao;
6      protected $user_struct_param;
7      protected $sql_delete_user;
8      protected $sql_insert_user;
9      protected $database_instance;
10     protected $uid;
11     public function setUp()
12     {
13         parent::setUp();
14         $this->database_instance = Database::obtain(INIT::$DB_SERVER,
               INIT::$DB_USER, INIT::$DB_PASS, INIT::$DB_DATABASE );
15         $this->user_Dao = new Users_UserDao($this->database_instance)
               ;
16         /**
17          * user insertion
18          */
19         $this->sql_insert_user = "INSERT␣INTO␣".INIT::$DB_DATABASE."
               .`users`(`uid`,␣`email`,`create_date`,`first_name`,`
               last_name`,`api_key`)VALUES(NULL,'bar@foo.net','2016-04-
               11␣13:41:54','Bar','Foo','');";
20         $this->database_instance->query($this->sql_insert_user);
21         $this->uid=$this->database_instance->getConnection()->
               lastInsertId();
22         $this->sql_delete_user = "DELETE␣FROM".INIT::$DB_DATABASE.".`
               users`WHERE␣uid='" . $this->uid . "';";
23     }
24     public function  test_getByUid(){
25
26         $user= $this->user_Dao->getByUid($this->uid);
27         $this->assertTrue( $user instanceof Users_UserStruct);
28         $this->assertEquals("{$this->uid}", $user->uid);
29         $this->assertEquals("bar@foo.net", $user->email);
30         $this->assertEquals("12345", $user->salt);
31         $this->assertEquals("987654321qwerty", $user->pass);
32         $this->assertEquals("2016-04-11␣13:41:54", $user->create_date
               );
33         $this->assertEquals("Bar", $user->first_name);
34         $this->assertEquals("Foo", $user->last_name);
35     }
36
37 }
```

Code 11: Test about the class UserDAO

Abstract Classes An abstract class can't be instantiated but methods inside it can be inherited by classes that extend it.
Utilization of abstract classes is encouraged because it mitigate duplicated code.

In fact there is a common pattern used in software design that advises to use abstract classes where it is possible.
Testing a class that can't be instantiated could be a problem. I tested this classes through classes that extend them.

WordCounter *WordCounter.php* has the responsibility of counting words of strings in a document that is being translated through the application.

It communicate with database updating counters of already *translated words* and *untouched* words.

Code 12 is about obtaining the correct data from database.

```
1  class GetUpdatedValueTest extends AbstractTest{
2      protected $word_counter;
3      protected $job_id;
4      protected $job_password;
5      protected $segment_id;
6      /**
7       * @var WordCount_Struct
8       */
9      protected $word_count_struct;
10
11     public function setup(){
12         parent::setUp();
13         $this->job_id= 9999; //sample
14         $this->job_password= "bar999foo"; //sample
15         $this->segment_id= "789099"; //sample
16
17         $this->word_count_struct= new WordCount_Struct();
18         $this->word_count_struct->setIdJob($this->job_id);
19         $this->word_count_struct->setJobPassword($this->job_password)
               ;
20         $this->word_count_struct->setIdSegment($this->segment_id);
21     }
22         /**
23      * @covers WordCount_Counter::getUpdatedValues
24      * @group regression
25      */
26     public function test_getUpdateValue_with_rejection(){
27         $this->word_count_struct->setNewWords(0);
28         $this->word_count_struct->setDraftWords(0);
29         $this->word_count_struct->setTranslatedWords(30);
```

```
30          $this->word_count_struct->setApprovedWords(0);
31          $this->word_count_struct->setRejectedWords(0);
32          $this->word_count_struct->setOldStatus("TRANSLATED");
33          $this->word_count_struct->setNewStatus("REJECTED");
34
35          $this->word_counter = new WordCount_Counter($this->
                word_count_struct);
36          $this->word_counter->setOldStatus("TRANSLATED");
37          $this->word_counter->setNewStatus("REJECTED");
38
39          $result = $this->word_counter->getUpdatedValues("15.00");
40
41          $this->assertTrue($result instanceof WordCount_Struct);
42          $this->assertEquals($this->job_id, $result->getIdJob());
43          $this->assertEquals($this->job_password, $result->
                getJobPassword());
44          $this->assertEquals($this->segment_id, $result->getIdSegment
                ());
45          $this->assertEquals(0, $result->getNewWords());
46          $this->assertEquals(0, $result->getDraftWords());
47          $this->assertEquals("-15.00", $result->getTranslatedWords());
48          $this->assertEquals(0, $result->getApprovedWords());
49          $this->assertEquals("+15.00", $result->getRejectedWords());
50          $this->assertEquals("TRANSLATED", $result->getOldStatus());
51          $this->assertEquals("REJECTED", $result->getNewStatus());
52      }
```

Code 12: Test about the method getUpdatedValue(float x) of class WordCountCounter

There are some assertions (line 47,49) that prove the correct behaviour of the method getUpdatedValues(..) invoked in line 39 in code 12.

6 Last Week

Suddenly I recognized that I was running out of time.
I focus on finishing to cover the code of partially tested classes and spent last moments refactoring.

6.1 Continuous Integration

Finally I saw the birth of an environment of continuous integration that had the aim of automate contributions workflow, running tests periodically, to certificate that new contributions didn't have generated regression.

When this environment will be fully operative, there is automated deployment in *production branch*, after that all tests have passed.

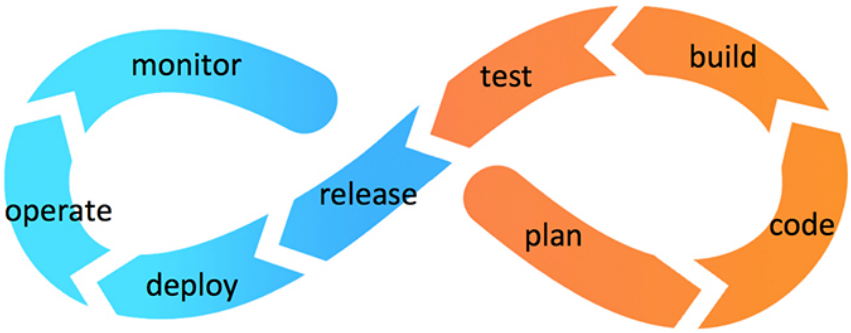

Figure 6: Continuous Integration Cycle

Part III

Conclusion

7 Best practices

None of the following procedures are essential for developers but if they are pursued together they will bring essential advantages.

7.1 Promote Testability

Make code that is well suited for being tested respecting Dependency Injection principle.

7.2 Implementing with Test Driven Development policy

At first glance this policy seems to be the most counter-intuitive action to perform.
People used to try and unavoidably they make mistakes, only then people learning from errors can improve their previous work.

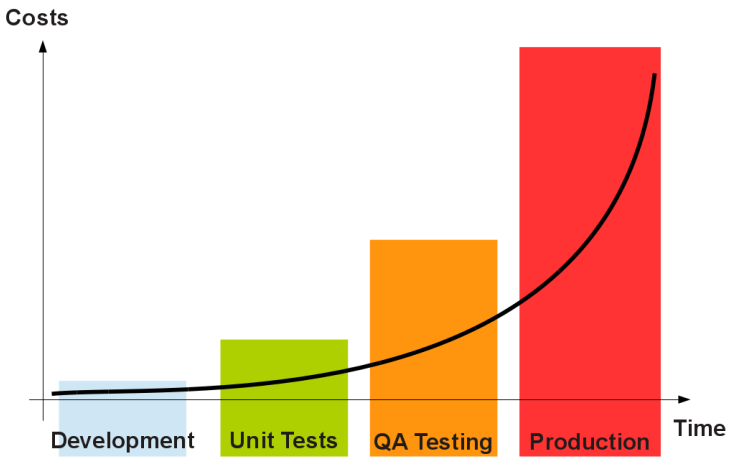

Figure 7: Function that describe the cost of debugging in relation with the time of bug discovery

Instead if developers apply TDD they will land safe paths to travel more and more complex, avoiding mistakes.
Mistakes in information technology are bugs and they are onerous to discover and fix.
In particular the fixing cost increment exponentially with the increase of the time gap between the bug introduction and its discovery as shown in fig.7.

So, why we shouldn't try to avoid bug insertion in code ?

7.2.1 TDD

Test Driven Design development could be the answer.

By definition this technique consists in looping five established steps:

At First Testing We have to make a simple test (code 13) based on one use case .

```
1  class FlowerTest extends AbstractTest {
2          public function test_verifying_flower_color() {
3                  $flower = new Flower('blue');
4                  $this->assertEquals("blue", $flower->getColor()) ;
5          }
6  }
```

Code 13: Basic Test

Failure Expected If we run this test it will fail certainly (code 14) , simply because there isn't any code that obtains color of petals yet.

```
1  Failed asserting that null matches expected 'blue'.
2  Expected :blue
3  Actual   :null
4
5  FAILURES!
6  Tests: 1, Assertions: 1, Failures: 1.
7  ** Resetting environment to development
8
9
10 Process finished with exit code 1
```

Code 14: Failure output

Coding We need to write for example code 15 that cover our use case .

```
1  class Flower {
2          /**
3          * @var String
4          */
5          protected $color;
6
7          public function __construct( $color ) {
8                  $this->color= $color;
9                  }
10         public function getColor(){
11                 return $this->color;
12                 }
13 }
```

Code 15: Implementation of algorithm

Expecting Success We just have to restart the test suite and finally the test will pass (code 16).

```
1  FlowerTest test_verifying_flower_color - Did in 1467395200.2418
       seconds.
2
3  Time: 16.3 seconds, Memory: 6.75Mb
4
5  OK (1 test, 1 assertions)
6  ** Resetting environment to development
7
8  Process finished with exit code 0
```

Code 16: Success output

Refactor code It consists in improving constantly the project structure.
If we spend time refactoring, we are following common design patterns to pursue clarity.

8 Goals

Finally I wrote about four hundred tests covering a quarter of the Matecat code.

Initially matecat project was covered less than 10%.
Now it is covered for almost the 25%.
I am proud of which I have accomplished and I am happy that my efforts could have positive repercussions on the future development of the application.

This experience was very instructive because I understood the importance of teamwork and some advanced testing techniques that will come handy during my career.