# AUnit

## A Simple Language-agnostic Tool for Automated Unit Testing

Michel de Champlain
Department of Computer Science and Software Engineering
Concordia University, Montreal, Canada

**SUMMARY**

**AUnit, abbreviated from "language-Agnostic Unit," is an instructional and educational approach. This simple tool is devoted to accelerating the learning and the adoption of one of the most important agile practices: Test-Driven Development (TDD) for students in learning an agile software process and how to integrate it into their course projects. The first instructional part of this approach is to learn the AUnit tool. The second educational part presents the procedure for using that tool.**

KEYWORDS: Language-agnostic tool, automated unit testing, instructional and educational approach, agile software process, TDD

## Introduction

AUnit is an application that emulates the output of a test framework like xUnit[1] by analyzing the output of test programs to the console. I wrote the first implementation of this simple tool in 2006 and have been using it ever since in my training and coaching sessions for Agile teams in the industry. AUnit was also translated in ANSI C to be portable and used to verify test results on small-footprint embedded systems. Since xUnit generally takes too much space to be used on target systems, therefore it is often confined to test only at a host station. For this main reason, one of the biggest advantages of AUnit is its lightness which can be used for both the host station and the target board. The second advantage is that AUnit is independent of any programming language as it compares program outputs in text format. Its simplicity is certainly its third major benefit, since it requires a minimal learning curve.

The following summarizes the advantages of AUnit vs xUnit:

| **Characteristics** | AUnit | xUnit |
|---|---|---|
| Independent of programming language | Yes | No, one version for each |
| Learning Curve | Very fast | More demanding |

[1]It is frequently called xUnit to represent all versions dedicated to programming languages like Java (JUnit), C# (NUnit), C++ (CppUnit), and so on.

 

## Getting Started

Making the transition to an agile software process is hard, much harder than many organizations anticipate. That transition requires changes in traditional software development practices, from waterfall to agile. It is hard mainly because, changing practices is challenging, but changing minds is way more complicated. It requires a minding shift. Fortunately, it is possible with the right approach. The key to the successful adoption of any agile practices is to obtain quickly a return of investment (ROI) on the effort required to learn it, practice it, and keep using it.

Although xUnit is a wonderful unit testing tool to use, I, unfortunately, witnessed several unsuccessful adoptions of this tool in the industry. Often, one of the main reasons was that developers never got their ROIs. Looking back on all the effort they had invested versus their gain, they quickly let go with bitter disappointment at their attempt.

Hence my motivation to develop an approach with a minimal learning curve allowing a faster ROI to convince them to continue this practice. This simple approach is devoted to accelerating the learning and the adoption of one of the most important agile practices: Test-Driven Development (TDD) for students in learning an agile software process and how to integrate it into their own course projects. The following sections present the two parts used to accelerate the adoption of doing automated unit testing with AUnit:

1. the instructional part to learn it.
2. the educational part to use it.

### 1. Learning the Structure of a Unit Test with AUnit

A unit test with AUnit is a test program that can be developed in any programming language. Generally, the test program is structured to generate three strings on the standard output. Each string is terminated by an end-of-line. Depending on the development platform used, an end-of-line is represented by:

- a carriage-return followed by a linefeed (`\r\n`) on Windows;
- a linefeed (`\n`) on Unix boxes; and
- a carriage-return (`\r`) on Mac.

Here is an example of a test program in the C programming language on Windows printing its results on the standard output:

```
1  // TestOne.c - Test program for function One()
2
3  int One() { return 3-2; }    // Function One() to be tested.
4
5  void main() {
6      printf("Test One\n");    // Print the name of the test program.
7      printf("1\n");           // Expected output.
8      printf("%d\n", One());   // Execution the function One() and print the result.
9  }
```

In order to be compared and verified by AUnit, a test program must print a first-line identifying the test program (line 6). The string format is as follows:

```
"Test " <name-of-the-program><end-of-line>
```

A `"Test "` string must be followed by the `<name-of-the-program>` and an `<end-of-line>`. The second line is the output expected by the test program (line 7). For example, `1` should be the result returned by

the function `One()` (line 3). And finally, the third line will be the one executing the function One() to be called by the test program and printing the result.

AUnit will take care of comparing the second and third lines and printing a dot ('`.`') for success and ('`F`') for failure. This output is similar to most xUnit test frameworks (with the console version). If all tests are successful, AUnit will display '`OK`'.

## 2. Learning the Steps How to Use AUnit

The following are two recurrent steps to (1) compile your test program and (2) execute AUnit to compare the output to verify the results.

### Step 1: Compile and run the test

The first step is to compile, run a test program, and to redirect its output to a text file.

#### Example in C

```
1 void main() {
2     printf("Test One\n");
3     printf("1\n");
4     printf("%d\n", 3-2);
5 }

c:\> cc TestOne              # Compile TestOne.c
c:\> TestOne > TestOne.txt   # Run TestOne.exe and redirect its output in TestOne.txt
```

#### Example in C#

```
1 public class TestOne {
2     public static void Main() {
3         System.Console.Write("Test One\n");
4         System.Console.Write("1\n");
5         System.Console.Write(3-2); System.Console.WriteLine();
6     }
7 }

c:\> csc TestOne             # Compile TestOne.cs
c:\> TestOne > TestOne.txt   # Run TestOne.exe and redirect its output in TestOne.txt
```

#### Example in Java

```
1 public class TestOne {
2     public static void main(String[] args) {
3         System.out.print("Test One\n");
4         System.out.print("1\n");
5         System.out.print(3-2); System.out.println();
6     }
7 }

c:\> javac TestOne               # Compile TestOne.java
c:\> java  TestOne > TestOne.txt  # Run TestOne.class and redirect its output in TestOne.txt
```

### Step 2: Compare the output to verify the results

The second step is to run the AUnit on the text file containing the execution results in order to compare the output stored. Whenever a test passes, a dot (`.`) is displayed. If all tests are successful, then `OK` is displayed. Since the `TestOne.txt` file contains only one test unit, the following example shows only one dot:

```
c:\> launit TestOne.txt
AUnit v1.3.0.20200912
Copyright (c) 2006-2020 Michel de Champlain. All Right Reserved.

.
1 tests OK
```

The `TestOne.txt` file contains a successful output and by running another execution which appends the same output in the following file `TestOne.txt`. We will get two dots and an `OK` to confirm that both tests are passed:

```
c:\> TestOne >  TestOne.txt
c:\> TestOne >> TestOne.txt
c:\> launit TestOne.txt
AUnit v1.3.0.20200912
Copyright (c) 2006-2020 Michel de Champlain. All Right Reserved.

..
2 tests OK
```

In the **Example in C**, if you want to see the behavior of AUnit in the case of a resulting incorrect output, you can change line 4 `printf("1\n");` by `printf("0\n");`. The resulting output will be different for display `0` if you compile again and run the test:

```
c:\> cc TestOne             # Recompile TestOne.bsharp
c:\> TestOne > TestOne.txt  # Execute TestOne et redirige la sortie dans TestOne.txt
c:\> launit TestOne.txt
AUnit v1.3.0.20200912
Copyright (c) 2006-2020 Michel de Champlain. All Right Reserved.

F
Failed Tests:
- Test One
```

## Comparing and Verifying More Results

AUnit is especially useful for automating the analysis of several outputs of test programs. It will display the list of failed tests after analyzing all outputs. AUnit is not limited to a single output line, we can indicate in the first line (identifier) the number of lines (`n`) to compare (`<n`). The next example is an output of six concatenated test programs:

```
Test One
abcdef
abcdef
Test Two
abcdf
abcdef
Test Three<2
abc
def
abc
def
Test Four<2
ab
def
abc
def
Test Five<3
abc
def
ghi
abc
def
ghix
Test Six<3
abc
def
ghi
abc
def
ghi
```

Result:

```
AUnit v1.3.0.20200912
Copyright (c) 2006-2020 Michel de Champlain. All Right Reserved.

.F.FF.

Failed Tests:
- Test Two
- Test Four
- Test Five
```

## How to get AUnit

The AUnit is an open-source tool and written in ANSI C to ease its porting on any platform. This PDF document and the source code is available on your Moodle course website.