Google Test

Entorno de pruebas unitarias

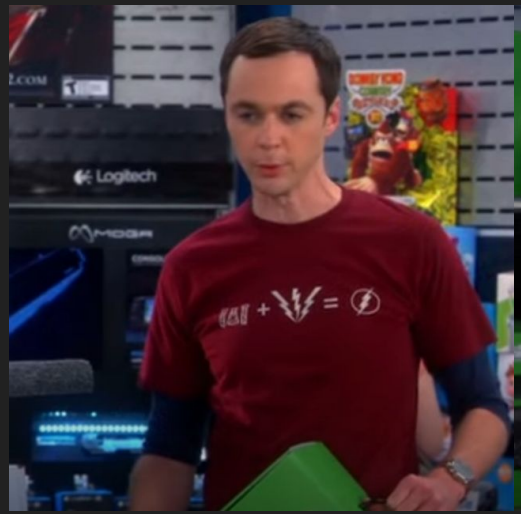
Autor: Antonio Sanchez (jasleon)
Fecha: Julio 29, 2021

Pruebas unitarias

En programación, una prueba unitaria es una forma de comprobar el correcto funcionamiento de una unidad de código.

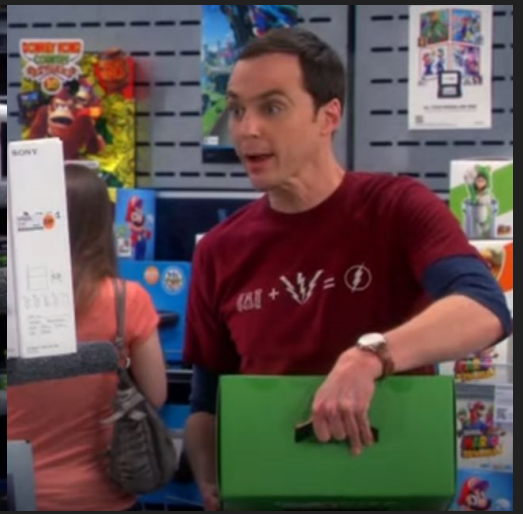
¿Por qué necesitamos probar nuestro código?

- Zune Freeze Result of Leap Year: Microsoft



- Me paré enfrente de una caja de iPods y compré un *Zune*

- ¿Qué es un Zune?



- ¡Sí, exactamente!

Fuente: *The Big Bang Theory* producida por Warner Bros.

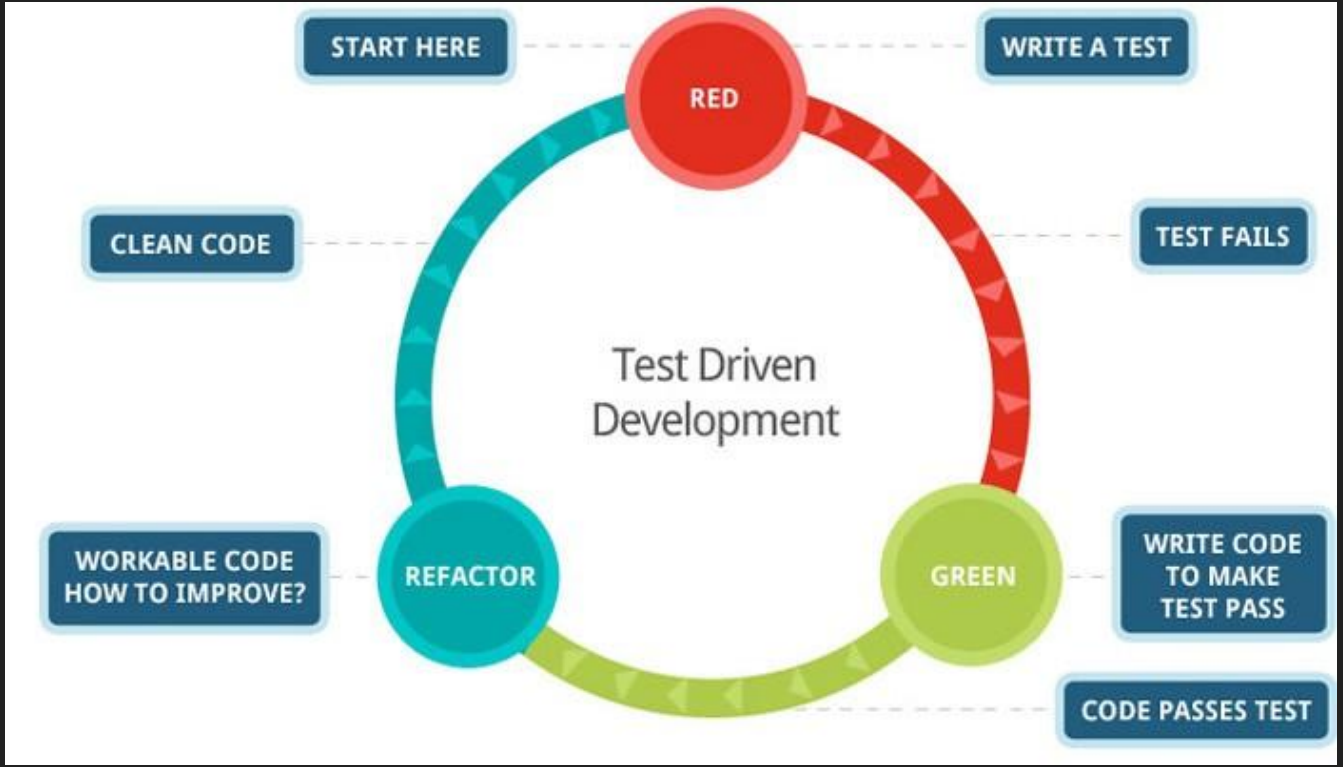
¿Cuándo debemos de agregar pruebas unitarias?

En el desarrollo guiado por pruebas (TDD), no se escribe código de producción sin haber escrito primero una prueba unitaria que falle.

El microciclo del TDD

1. Agregar una prueba.
2. Correr todas las pruebas y confirmar que la nueva prueba falla.
3. Hacer los cambios necesarios para pasar la prueba.
4. Correr todas las pruebas y confirmar que la nueva prueba pasa.
5. Refactorizar para eliminar código duplicado y mejorar la expresividad.

¿Cuándo debemos de agregar pruebas unitarias?



Fuente: *Red, Green, Refactor!* por Tun Khine

Entornos de pruebas unitarias en C++

https://hackingcpp.com/cpp/tools/testing_frameworks.html

- Doctest
- Catch 2
- Google Test
- Boost Test
- CUTE
- QtTest
- Mull
- ...

Google Test

Google Test es una biblioteca de pruebas unitarias para el lenguaje de programación C++, basada en la arquitectura xUnit.

https://github.com/google/googletest

Características de Google Test

- Un entorno de pruebas xUnit
  - *Test runner*, un programa ejecutable que corre las pruebas y reporta resultados
  - *Test case*, una clase elemental de la cual se heredan todas las pruebas unitarias
  - *Test fixture*, un conjunto de precondiciones necesarias para correr una prueba
- Un abundante conjunto de afirmaciones (assertions)
- Fallas fatales y no fatales
- Pruebas unitarias parametrizables por valor
- Varias opciones para correr las pruebas
- Generación de reportes en XML

Instrucciones para instalar Google Test

https://github.com/google/googletest/blob/master/googletest/README.md

Instalar con CMake*

```
git clone https://github.com/google/googletest.git -b release-1.11.0
cd googletest        # Main directory of the cloned repository.
mkdir build          # Create a directory to hold the build output.
cd build
cmake ..             # Generate native build scripts for GoogleTest.
make
sudo make install    # Install in /usr/local/ by default
```

* Si estás en un sistema *nix.

Conceptos básicos

- Las *afirmaciones* son declaraciones que comprueban si una condición es verdadera.
- Las *pruebas* utilizan afirmaciones para verificar el comportamiento del código probado.
- Un *conjunto de pruebas* contiene una o varias pruebas.
- Un *programa de prueba* puede contener varios conjuntos de pruebas.

Elementos básicos de Google Test

```
#include <gtest/gtest.h>
```

Encabezado

```
int add(int a, int b) {
  return a + b;
}
```

Función (unidad) a probar

```
TEST(TestSample, TestAdd) {
  ASSERT_EQ(3, add(1, 2));
}
```

Definición de la prueba unitaria

```
int main(int argc, char **argv) {
  testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

Funcion principal

Elementos básicos de Google Test

```cpp
#include <gtest/gtest.h>

int add(int a, int b) {
  return a + b;
}

TEST(TestSample, TestAdd) {
  ASSERT_EQ(3, add(1, 2));
}

int main(int argc, char **argv) {
  testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from TestSample
[ RUN      ] TestSample.TestAdd
[       OK ] TestSample.TestAdd (0 ms)
[----------] 1 test from TestSample (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.
```

Afirmaciones (*assertions*)

Las afirmaciones son macros que se asemejan a llamadas a funciones y vienen en pares que prueban lo mismo pero tienen diferentes efectos.

- `ASSERT_*` generan fallas fatales y abortan la función actual.
- `EXPECT_*` generan fallas no fatales y no abortan la función actual.

Por lo general, se prefieren `EXPECT_ *`, ya que permiten informar más de una falla en una prueba. Sin embargo, puedes usar `ASSERT_ *` si no tiene sentido continuar cuando falla la afirmación en cuestión.

Fuente: Googletest Primer - Assertions

Pruebas simples

Para crear una prueba:

1. Utiliza la macro `TEST()` para definir y nombrar una función de prueba.
2. Utiliza afirmaciones para verificar los valores.
3. El resultado de la prueba está determinado por las afirmaciones; si alguna afirmación en la prueba falla (fatalmente o no fatalmente), o si la prueba deja de funcionar, toda la prueba falla. De lo contrario, tiene éxito.

```
TEST(TestSuiteName, TestName) {
  ... test body ...
}
```

Fuente: Googletest Primer - Simple Tests

Integración con CMake

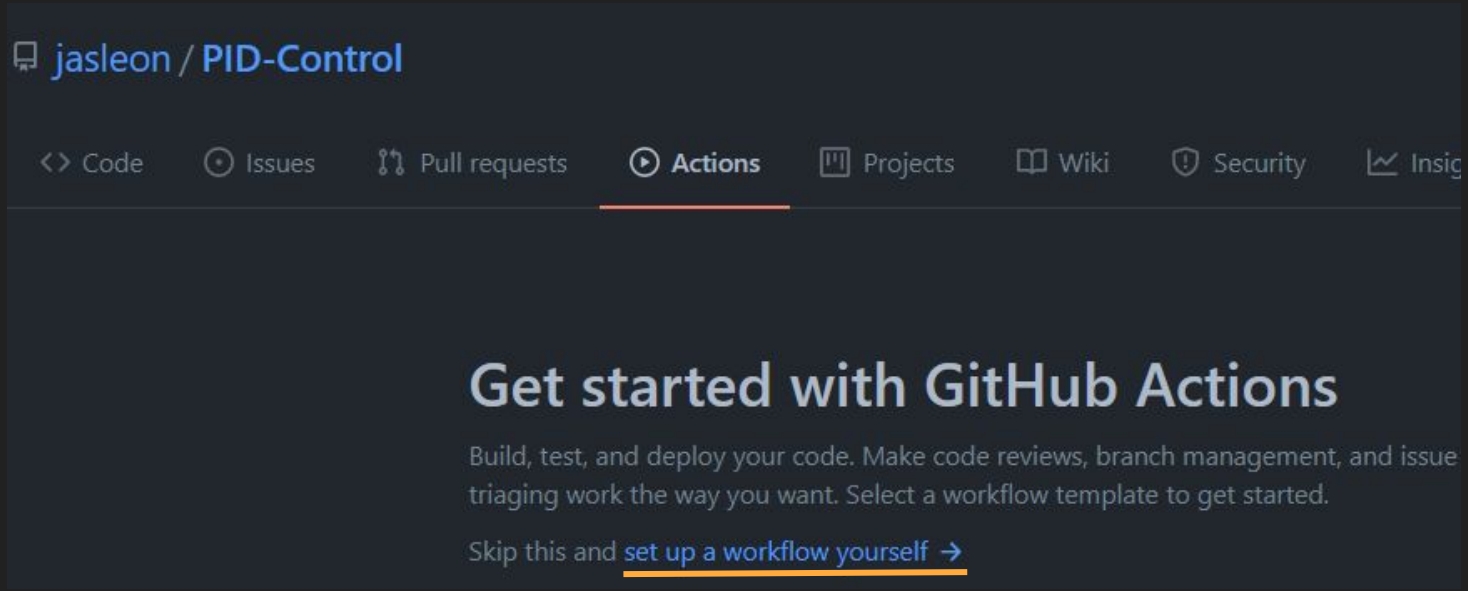
CMake es una familia de herramientas multiplataforma de código abierto diseñada para construir, probar y empaquetar software.

```
# Set project name
project(arithmetic)

# Add executable to run unit tests
find_package(GTest)
if(GTest_FOUND)
  add_executable(test main.cpp)
  target_link_libraries(test ${GTEST_LIBRARIES} pthread)
else()
  message("googletest needs to be installed to generate the test executable")
endif()
```

Integración Continua (CI) con GitHub Actions

GitHub permite automatizar, personalizar y ejecutar flujos de trabajo de desarrollo de software directamente en un repositorio con GitHub Actions.



Integración Continua (CI) con GitHub Actions

```yaml
# Install googletest
- name: Install googletest
  run: |
    git clone https://github.com/google/googletest.git -b release-1.10.0 && cd googletest
    mkdir build && cd build
    cmake .. && make
    sudo make install

# Make a build directory and compile
- name: Build
  run: |
    mkdir build && cd build
    cmake .. && make

# Run the executable
- name: Run
  run: ./build/app

# Run unit tests
- name: Test
  run: ./build/test
```

https://github.com/jasleon/arithmetic/blob/main/.github/workflows/main.yml

Propriedades de buenas pruebas

0. Existencia!

1. Correctas/Apropiadas
2. Completitud
3. Legibilidad
4. Demostrabilidad
5. Resiliencia

- Fáciles de correr
- Rápidas de correr
- TDD
- Deterministas
- Cobertura de código

Recuerden, malas pruebas son casi siempre mejores que ninguna prueba!

Fuente: *The Science of Unit Tests* por Dave Steffen (CppCon 2020)

Bibliografía

- Grenning, J., 2014. *Test-driven development for embedded C*.
- Langr, J., 2014. *Modern C++ programming with test-driven development*.
- Stoenescu, S., (n.d.). *C++ Unit Testing: Google Test and Google Mock* [MOOC]. Udemy. https://www.udemy.com/course/cplusplus-unit-testing-google-test-and-google-mock/
- Schaffranek, J., (n.d.). *C++ Projekte für Fortgeschrittene: CMake, Tests und Tooling* [MOOC]. Udemy. https://www.udemy.com/course/c-projekte-fur-fortgeschrittene-cmake-tests-und-tooling/
- Steffen, D. (2020, September 9-23). *The Science of Unit Tests* [Conference presentation]. CppCon 2020, online. https://youtu.be/FjwayiHNI1w
- Winters, T., & Wright, H. (2015, September 19-25). *All Your Tests are Terrible: Tales from the Trenches* [Conference presentation]. CppCon 2015, Bellevue, WA, United States. https://youtu.be/u5senBJUkPc

Referencias

- Ejemplos de la presentación: https://github.com/jasleon/arithmetic
- Documentación de googletest:
  - Página principal, https://google.github.io/googletest/
  - Google primer, https://google.github.io/googletest/primer.html
  - Referencia de pruebas, https://google.github.io/googletest/reference/testing.html
  - Referencia de afirmaciones, https://google.github.io/googletest/reference/assertions.html
  - Temas avanzados, https://google.github.io/googletest/advanced.html