Test driven development for embedded C by James Grenning

* Test-Driven Development is the key to making meaningful progress on the code before hardware and throughout the development cycle
* DLP means debug later programming which is anti TDD
* TDD microcycle
* Refactoring : means cleaning up code
* There is Red Green Refactor from the java world
* A unit test harness is a software package that allows a programmer to express how production code should behave

My plan is to do tdd in eclipse so at first make the eclipse like env ready . to code arduino in eclipse follow the better arduino env guide by me

<https://vimeo.com/131194135>

**Faking and Mocking Legacy Embedded C - James Grenning**

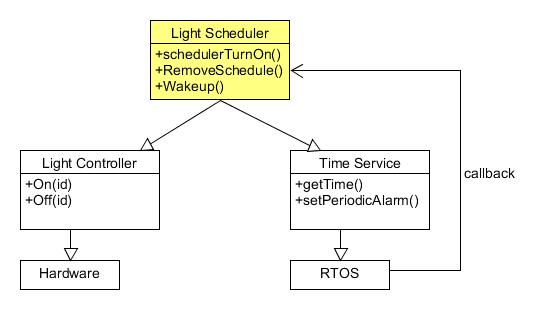
Kinds of test doubles

* Fakes
* Spy
* Null object
* Exploding fakes
* Mock object
* Others

When must you use test doubles ?

1. When the code under test cannot be conveniently tested with real collaborators
2. When manual verification is needed (printed output, user interaction)
3. User Interface ( use model view controller )
4. Database
5. When the results change (Time, random events)
6. When failures need to be simulated ( Network down )
7. Operating system calls, file system calls
8. Special Hardware Interaction

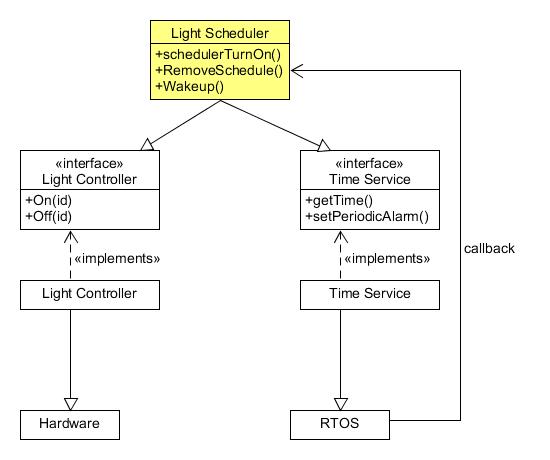
Let’s say our end code is a Light Scheduler. It uses Light Controller & Time Service also



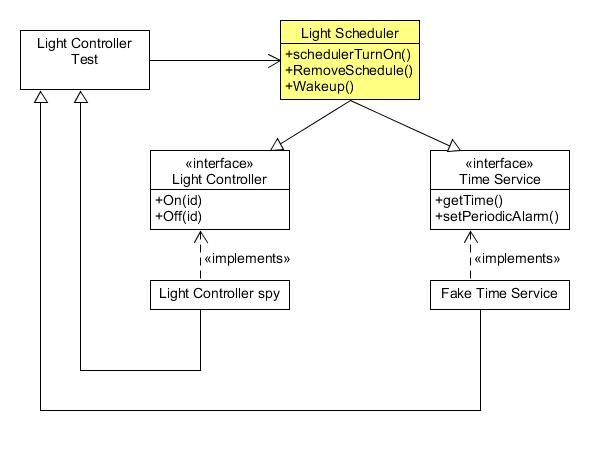
Working way: every minute RTOS callbacks the light scheduler and the light scheduler reads the time and if it is time for Light to be on/off it does it

Now our motive is to test the Light scheduler Code

Now if we make our design a bit more modular then our testing would be great



Now we can create spy/test doubles and test our target code easily and the test controller code will say whether tests are passed or not



Upper picture says how to do the mock in cpp below is a C way of doing things

Assigning the address of a function to a mock function with the same prototype is another example of mock

For example say light scheduler uses the function:

Unsigned long getTime(void)

{ //actual

Some Cpp OOp

Inheritance = Base class: child class = is A relationship = generalization = extends

Child class can override functions of the Base class

If a Function of a class is virtual

If a function of a base class is made pure virtual function then the class becomes Abstract Class then that function must be override in the child class. = implements

Class A {

B b;

}

If a class has other class type variables as members then this relationship is called Aggregation = has a relationship

Well too hard ☹ switching back to jame’s book

The embedded TDD cycle has 4 stages:

1. ***TDD microcycle :*** write code test it on the host development system
2. ***Compiler Compatibility check:*** periodically compile for the target to check compatibility
3. ***Run unit tests in an eval board***
4. ***Run unit tests on the actual hardware***

Some terminology:

**Code under test:**the code under test

**Production Code:** part of the released code

**Test Code:** Code that is used for testing the production code

**Test Case:** A test Code that describes the behavior of code under test

**Test Fixture:** Code that provides proper environment for a series of test cases to exercise the code under test

TEST\_SETUP( ) and TEST\_TEAR\_DOWN( ) keep duplication out of the sprintf( ) tests in Unity

Unity Test framework terminology:

**TEST() macro**

TEST( test group name, name of the test in this group)