TDD in C

Bas Vodde - Odd-e
Michael Feathers - Object Mentor

Test-Driven Development

The single rule of Test-Driven Development (or test-first programming) :

- Only ever write code to fix a failing test

- Write a test (which fails -> “red”)
- Write the code (to make test pass -> “green ”)
- Refactor the code and test (you’re still “green ”)

Unit-test

A test is not a unit test if:

- It talks to the database
- It communicates across the network
- It touches the file system
- It can't run at the same time as any of your other unit tests
- You have to do special things to your environment (such as editing config files) to run it.

TDD in C

C or C++?

- Why C++ (e.g. gcc):
  - Able to use C++ ut framework
  - Able to use C++ features in tests
- Why C:
  - Not annoyed by the small differences
  - Able to use a C compiler.
    - E.g. run tests in “real environment”

Compilation

- Fast build:
  - Limit dependencies - Especially no header dependencies!
  - Incremental build - Generate dependency files
  - Compile modules/subsystems
- Execute tests in Makefile!

Refactoring

- All manual -> no tools
  - It sucks
- Function to Function Pointer refactoring!

TDD Cycle

- Same as in other language
- Take about 20 minutes...

C Design

- C can be used as OO language!
  - Good written C is OO
- OO techniques
  - Structs with Function Pointers
  - Class-structs
  - Global function pointers

Structs with FPs

```
struct A
{
   void (*openA)(struct A* a);
   void (*closeA)(struct A* a);

// Private
   int member;
   int anotherMember;
};
```

Takes much memory per object

Class-struct

```
struct classA
{
      void (*open)(struct A* a);
      void (*close)(struct A* a);
};

struct A
{
      struct classA * cls;
// Private
      int member;
      int anotherMember;
};

#define A_open(a) (((struct A*)a)->cls->open(a))
#define A_close(a) (((A*)a)->cls->close(a))
```

Better. Much work though.

Global function ptrs

Header

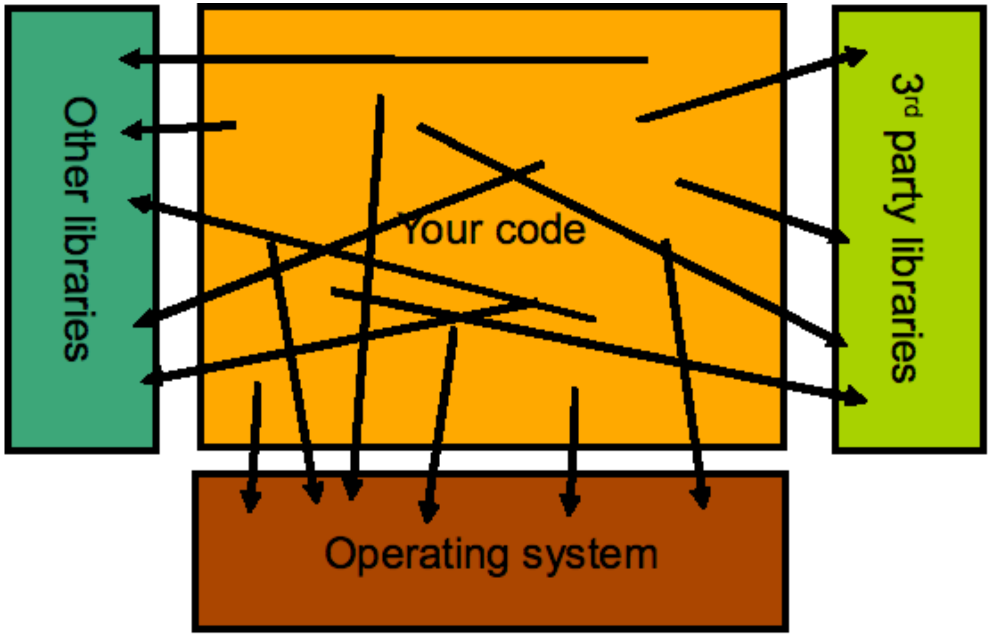
```
struct A
{
	int member;
	int anotherMember;
};

extern void (*a_open)(struct A*);
extern void (*a_close)(struct A*);
```

Source

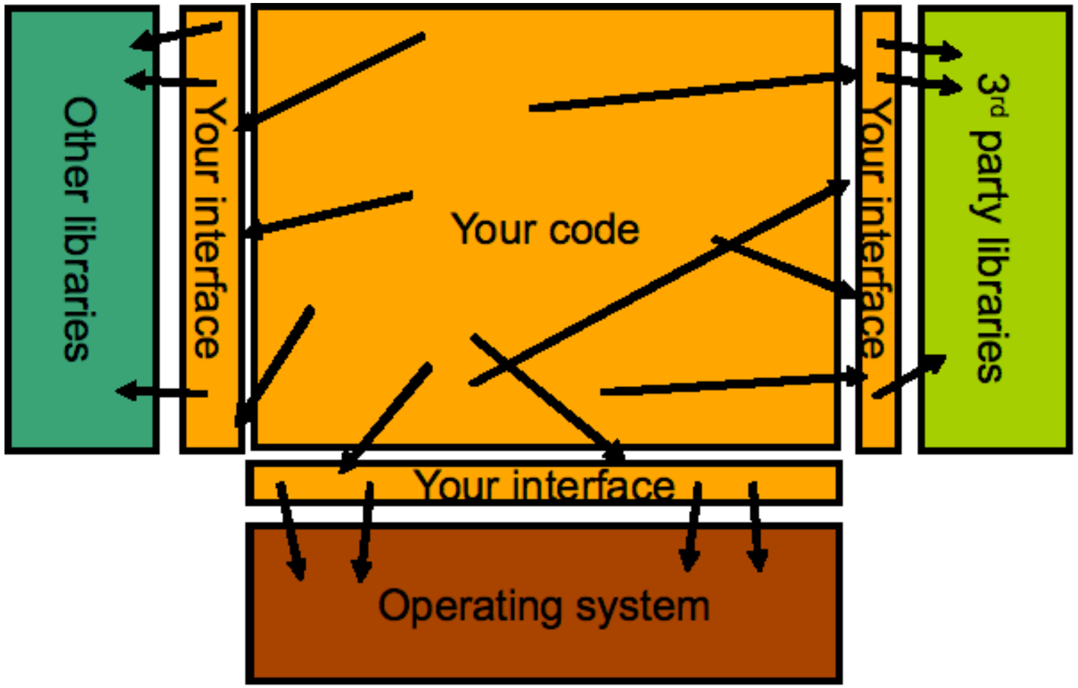
```
void a_open_imp(struct A*)
{
	printf("A Open\n");
}

void (*a_open)(struct A*) = a_open_imp;
```

Simple and allows dynamic stubbing and objects.
Very limited though

Badly structured



Dependencies separated



Stubbing

- Static
  - Preprocessor
  - Link
- Dynamic
  - Function pointers

Different stubs

- Exploding stubs
  - Fail when called
- Generic stubs
  - Configurable
- Using function pointers
  - Settable

Example test

```
// Testing FuncB which calls FuncA

TEST_GROUP(FuncBTest)
{
    static int dummyFuncA()
    {
        return 1;
    }
    void setup()
    {
        UT_FPSET(funcA, &dummyFuncA);
    }
    void teardown()
    {
    }
};


TEST(FuncBTest, Ok)
{
    LONGS_EQUAL(1, funcB())
}
```

Exercises

Exercise #1

- Test-drive “Hello World”

Exercise #2

- Test drive a simple chat client-server
  - Using posix sockets

Exercise #3

- Test-drive a program that counts lines of code of a C program
  - Ignore preprocessor