Getting Your Legacy Code Under Control

Dror Helper

@dhelper http://helpercode.com

Module Overview

What is “legacy code”?

- Legacy Code definition
- The legacy code dilemma
- Sensing and separation

Patterns of unit testing legacy code

- Injecting fakes into legacy code

What Is Legacy Code?

No longer engineered but continuedly patched

Difficult to add features without breaking functionality

Maintained by someone who didn’t write it

Has users!

“With tests, we can change the behavior of the code quickly and verifiably. Without them, we really don’t know if our code is getting better or worse.”

Michael Feathers – Working effectively with legacy code

Handling Legacy Code

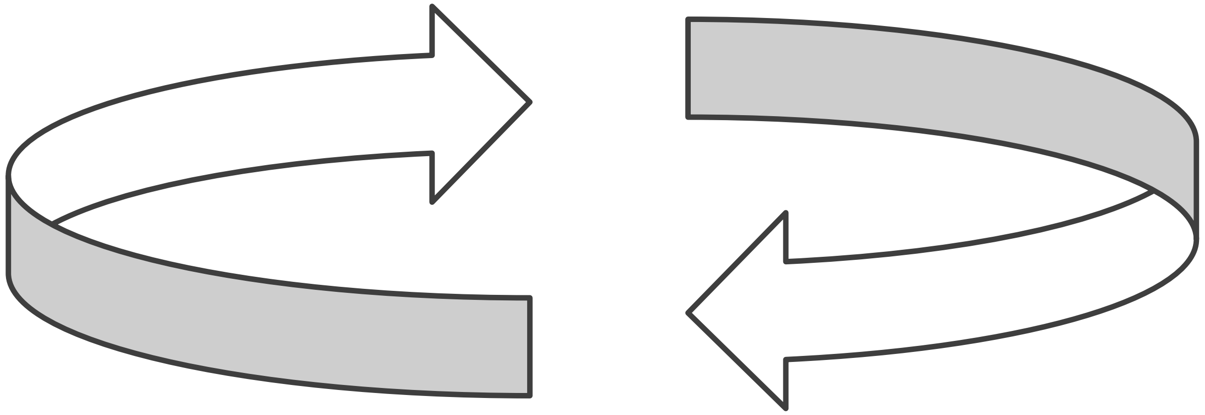
Understand what the code does

Create safety net for change

Refactor

The Legacy Code Dilemma

To change code we need to add tests



To add tests we need to change code

It's All About Dependencies

Dependencies == other classes

- External libraries
- Other resources
- Outside of our control

Dependency-related Issues

- Hard to instantiate class in test
- Difficult to run methods in test
- Impossible to assert results

Sensing and Separation

```
TEST(SendEmailTest)
{
    EmailClient client;
    User user;
    ...

    client.SendEmailToUser(user);

    // Sensing problem
    Assert???
}
```

```
void SendEmailToUser(User user)
{
    auto snmpClient = new ...

    ...

    // Separation problem
    snmpClient.Send(user.Email, msg);

    ...
}
```

Solution: Fake Objects

Easy to initialize

Control behavior

Verify results

Isolate tested code

Difficult to Fake Dependencies

Static methods

Cannot use inheritance

Nonvirtual methods

Cannot be overridden

Singletons

Cannot be replaced by fake objects

Internally instantiated

Cannot replace with fake objects

Heavy classes

The class under test is the dependency

Faking Static and Nonvirtual Methods

Refactor to virtual methods

Fake internal methods called by them

Hi-perf dependency injection

Introduce static/instance delegator

Faking Private and Protected Methods

```
class Foo
{
public:
    virtual bool MyPublicMethod(MyClass* c){...};
protected:
    virtual void MyProtectedMethod(int a, int b){...};
private:
    virtual int MyPrivateMethod(){...}
}
```

Faking Private and Protected Methods

```
class Foo
{
public:
    MOCK_METHOD1(MyPublicMethod, bool(MyClass*));
    MOCK_METHOD2(MyProtectedMethod, void(int, int));
    MOCK_METHOD0(MyPrivateMethod, int());
}
```

```cpp
template <class PacketStream>
class PacketReader {
public:
  void ReadPackets(PacketStream* stream, size_t packetNum);
};
```

Using Hi-perf Dependency Injection

1. **Create a fake class with method definitions similar to dependency**
2. **Use template on dependent class to inject the dependency type**

Introduce Instance Delegator

1. **Identify problematic static method**
2. **Create an instance method**
   - Calls the static method
3. **Use DI to pass the class instance**

Faking Singletons – Introduce Static Setter

```
class MySingleton {
public:
    static MySingleton* GetInstance(){
        ...
        return instance;
    }
private:
    MySingleton(){...}
};
```

Step 1 – Add Forward Declaration & Friends

```
class MyFakeSingleton;
class MySingletonAccessor;


class MySingleton {
    ...
    friend class MyFakeSingleton;
    friend class MySingletonAccessor;
};
```

Step 2 - Create Accessor

```
class MySingletonAccessor {
public:
    static void Set(MySingleton* other) {
        MySingleton::GetInstance();
        delete MySingleton::instance;
        MySingleton::instance = other;
     }
};
```

Step 3 – Write Test

```
TEST(ICanFakeSingletons)
{
    auto myFake = new MyFakeSingleton();

    MySingletonAccessor::Set(myFake);

   ...
}
```

Extract and Override

When to use:

- When hard coded initialization in constructor
- When object created during method
- When method calls external dependency

1. **Identify dependency creation point**
2. **Extract into a protected factory method**
3. **In the test create a derived class and override method**

Summary

Legacy code and its challenges

- Using fake objects to test legacy code

Legacy code patterns:

- Faking private/protected methods
- Using hi-perf Dependency Injection
- Introduce instance delegator
- Faking Singletons
- Extract and override