C++ Test-driven Development
Unit Testing, Code Assistance and Refactoring for Testability

Prof. Peter Sommerlad
Director IFS Institute for Software
Amazon, September 2014
Software Quality

- **classic approach:**
  - manual testing after creation

Cape of Good Hope

or bury your head in the sand?
But:
Small Cute Things
Grow to become larger Problems!
C++ Unit Testing with CUTE in Eclipse CDT

Test-Driven Development and Refactoring

- CUTE http://cute-test.com - free!!!
- simple to use - test is a function
  - understandable also for C programmers
- designed to be used with IDE support
  - can be used without, but a slightly higher effort
- deliberate minimization of #define macro usage
  - macros make life harder for C/C++ IDEs
TDD Cycle

This is the part I'd like to show automatic support today.

**RED**
make a failing test

**Integrate**
make test and change permanent - check in

**Green**
make a change to pass the test

**Refactor**
make the design simpler
Eclipse CDT TDD Support for CUTE and CUTE-plug-in features

- get the cute plug-in from http://cute-test.com/updatesite or download Cevelop
- create CUTE test project
- create test function
- create function definition from call (in test case)
- create type definition from usage (in test case)
- create variable definition from usage (in test case)
- move type definition into new header file (from test case file)
- toggle function definition between header and implementation (part of CDT)
Simulation of a Switch (class)

alternative possible, if this seems too simple and time permits

THE LIST FOR CLASS SWITCH
1. CREATE, GETSTATE ➔ OFF
2. TURNON, GETSTATE ➔ ON
3. TURNOFF, GETSTATE ➔ OFF
- **New C++ Project**
  - CUTE Test Project

- **Run as CUTE Test**

- **Fix Failing Test**
  - write first "real" test
    - rename test case function
    - write test code
    - make it compile through TDD quick-fixes (2x)
      - create type
      - create function

- **Run Tests**
  - green bar

- **iterate...**
Mockator

C++ Legacy Code Refactoring enabling Unit Testing

Seam Introduction
Object Seam
Compile Seam
Link Seam
Preprocessor Seam

Test Double generation
Mock Object generation
Function Tracer generation

Master Thesis by Michael Rüegg
inspired and supervised by Prof. Peter Sommerlad
available at http://mockator.com
or integrated into Cevelop http://cevelop.com
Four Phase Test Case
Setup, Exercise, Verify, Teardown

- SUT - System under Test
- DOC - Depended on Component,

source: xunitpatterns.com - Gerard Meszaros

How to decouple SUT from DOC?
Seams!

How to decouple SUT from DOC?

- Introduce a Seam:
  - makes DOC exchangeable!
  - C++ provides different mechanisms

- Object Seam (classic OO seam)
  - Introduce Interface - Change SUT to use Interface instead of DOC directly
    - introduces virtual function lookup overhead
  - Pass DOC as a (constructor) Argument
  - Pass Test Double as Argument for Tests

- Compile Seam (use template parameter)
  - Make DOC a default template Argument

Adding a seam is often called “Dependency Injection”
Introduce an Object Seam

**classic inheritance based mocking**
- extract interface for DOC -> IDOC
- make SUT use IDOC
- create MOCK implementing IDOC and use it in UT

In C++ this means overhead for DOC (virtual functions)!

Diagram:
- SUT
- DOC
- UT
- SUT
- IDOC
- MOCK
Introduce an Object Seam

- **classic inheritance based mocking**
  - extract interface for DOC -> IDOC
  - make SUT use IDOC, edit constructor
  - create MOCK implementing IDOC and use it in UT

- in C++ this means overhead for DOC (virtual functions)!
Introduce a Compile Seam

- **C++ template parameter based mocking**
  - make DOC a default template argument of SUT

![Diagram](image-url)
absolutely no change on existing code of SUT needed!

remove Dependency on system/library functions

shadowing through providing an alternative implementation earlier in link sequence

- avoid dependency on system or non-deterministic functions, e.g. rand(), time(), or "slow" calls

wrapping of functions with GNU linker --wrap option, allows calling original also

- good for tracing, additional checks (e.g., valgrind)

wrapping functions within dynamic libraries with dlopen/dlsym&LD_PRELOAD

problem: C++ name mangling if done by hand (solved by Mockator)

replace calls through #define preprocessor macro

- as a means of last resort, many potential problems
What is Refactoring?

- **Ongoing cleaning**
  - “Clean Code”

- **Assure long-term quality**
  - code is always used longer than expected

- **Find better design**
  - understandability - habitable code

- **Remove duplication**
  - maintainability
A very simple game, roll dice, check if you’ve got 4 and you win, otherwise you loose.

We want to test class Die first:

```cpp
#include <cstdlib>

struct Die
{
    int roll() { return rand()%6 + 1; }
};
```
#include "Die.h"
class GameFourWins
{
    Die die;
public: 
    GameFourWins();
    void play();
};

void GameFourWins::play()
{
    if (die.roll() == 4) {
        cout << "You won!\n";
    } else {
        cout << "You lost!\n";
    }
}
#include "Die.h"
#include <iostream>

class GameFourWins
{
    Die die;
public:
    GameFourWins();
    void play(std::ostream &os = std::cout);
};

void GameFourWins::play(std::ostream &os){
    if (die.roll() == 4) {
        os << "You won!\n";
    } else {
        os << "You lost!\n";
    }
}
We now can use a `ostringstream` to collect the output of `play()` and check that against an expected value:

```cpp
void testGame() {
    GameFourWins game;
    std::ostringstream os;
    game.play(os);
    ASSERT_EQUAL("You lost!\n", os.str());
}
```

What is still wrong with that test?
deliver predefined values

- we need that for our Die class

Introduce an Interface

```cpp
struct DieInterface {
    virtual ~DieInterface() {}
    virtual int roll() = 0;
};

struct Die: DieInterface {
    int roll() { return rand() % 6 + 1; }
};
```

now we need to adjust Game as well to use DieInterface& instead of Die

Mockator Pro plug-in will make those code conversions automatic (Summer 2012)
- Changing the interface, need to adapt call sites
- theDie must live longer than Game object

```cpp
class GameFourWins
{
    DieInterface &die;
public:
    GameFourWins(DieInterface &theDie): die(theDie) {} 
    void play(std::ostream &os = std::cout); 
};
```

- now we can write our test using an alternative implementation of DieInterface
- would using pointer instead of reference improve situation? what’s different?
This way we can also thoroughly test the winning case:

```cpp
struct MockWinningDice: DieInterface{
    int roll() { return 4; }
};

testWinningGame() {
    MockWinningDice d;
    GameFourWins game(d);
    std::ostringstream os;
    game.play(os);
    ASSERT_EQUAL("You won!\n", os.str());
}
```
A C++ alternative using templates

- **advantages:** no virtual call overhead, no extra Interface extraction
  - transformation provided by our "Introduce Typename Template Parameter" Refactoring
- **drawback:** inline/export problem potential

```cpp
template <typename Dice=Die>
class GameFourWinsT
{
    Dice die;

public:
    void play(std::ostream &os = std::cout){
        if (die.roll() == 4) {
            os << "You won!\n";
        } else {
            os << "You lost!\n";
        }
    }
};

typedef GameFourWinsT<Die> GameFourWins;
```
The resulting test looks like this:

```cpp
struct MockWinningDice{
    int roll(){return 4;}
};
void testWinningGame() {
    GameFourWins<MockWinningDice> game;
    std::stringstream os;
    game.play(os);
    ASSERT_EQUAL("You won!\n",os.str());
}
```

should we also mock the ostream similarly?
We want also to count how often our dice are rolled. How to test this?

```cpp
struct MockWinningDice: DieInterface{
    int rollcounter;
    MockWinningDice(): rollcounter(0){}
    int roll(){++rollcounter; return 4;}
};
void testWinningGame() {
    MockWinningDice d;
    GameFourWins game(d);
    std::ostringstream os;
    game.play(os);
    ASSERT_EQUAL("You won!\n", os.str());
    ASSERT_EQUAL(1, d.rollcounter);
    game.play(os);
    ASSERT_EQUAL(2, d.rollcounter);
}
```
C++ template parameters can be used for mocking without virtual member function overhead and explicit interface extraction.

- no need to pass object in as additional parameter
- unfortunately no default template parameters for template functions (yet)

You can mock

- Member Variable Types
- Function Parameter Types

Mocking without template inline/export need is possible through explicit instantiations
Mockator Demo - Compile Seam using Templates

- **Extract Template Parameter**
  - part of CUTE plug-in

- **Use Template in Test**
  - introduce Mock object for DIE
    - create test double class...
    - add missing member function
    - implement test double code

- **Add Mock Object Support**
  - check for expected calls (C++11)

- **additional features**
  - dependency injection through
    - templates
    - abstract interface classes
Generating trace functions like this one is easy: Ctrl+Alt+T (on Linux)

Mockator passes mockator_malloc.h to the GNU compiler by using its -include option
Mockator - Link Seam - Shadowing & Wrapping Library Functions

- **Shadowing on Linux only using GCC Linker**
- **Allows wrapping of functions in shared libraries**
  - Both Linux and MacOS X supported
- **Mockator does all the hard work like**
  - creating a shared library project,
  - adding dependencies to the dl library,
  - creating run-time configurations with the following env values, etc.,
    - Linux: LD_PRELOAD=libName.so MacOS X:
    - DYLD_FORCE_FLAT_NAMESPACE=1
    - DYLD_INSERT_LIBRARIES=libName.dylib

```c
int foo(int i) {
    static void *gptr = nullptr;
    if(!gptr) gptr = dlsym(RTLD_NEXT,"_Z3fooi");
    typedef int (*fptr)(int);
    fptr my_fptr = reinterpret_cast<fptr>(gptr);
    // TODO put your code here
    return my_fptr(i);
}
```
Questions?

- http://cute-test.com - Unit Testing
- http://mockator.com - Seams and Mocks
- http://linticator.com - Lint Integration
- http://includator.com - include optimization
- http://sconsolidator.com - SCons integration

Have Fun with TDD
Mockator and Refactoring!
Or get it all at once in a modern C++ IDE

Cevelop

Your C++ code deserves it

Download IDE at:
www.cevelop.com