MOCKING IN C++

Pittsburgh C++ Meetup 2020-01

Jonathan Povirk

https://github.com/JonTheBurger/mocking-in-cxx

A REVIEW...

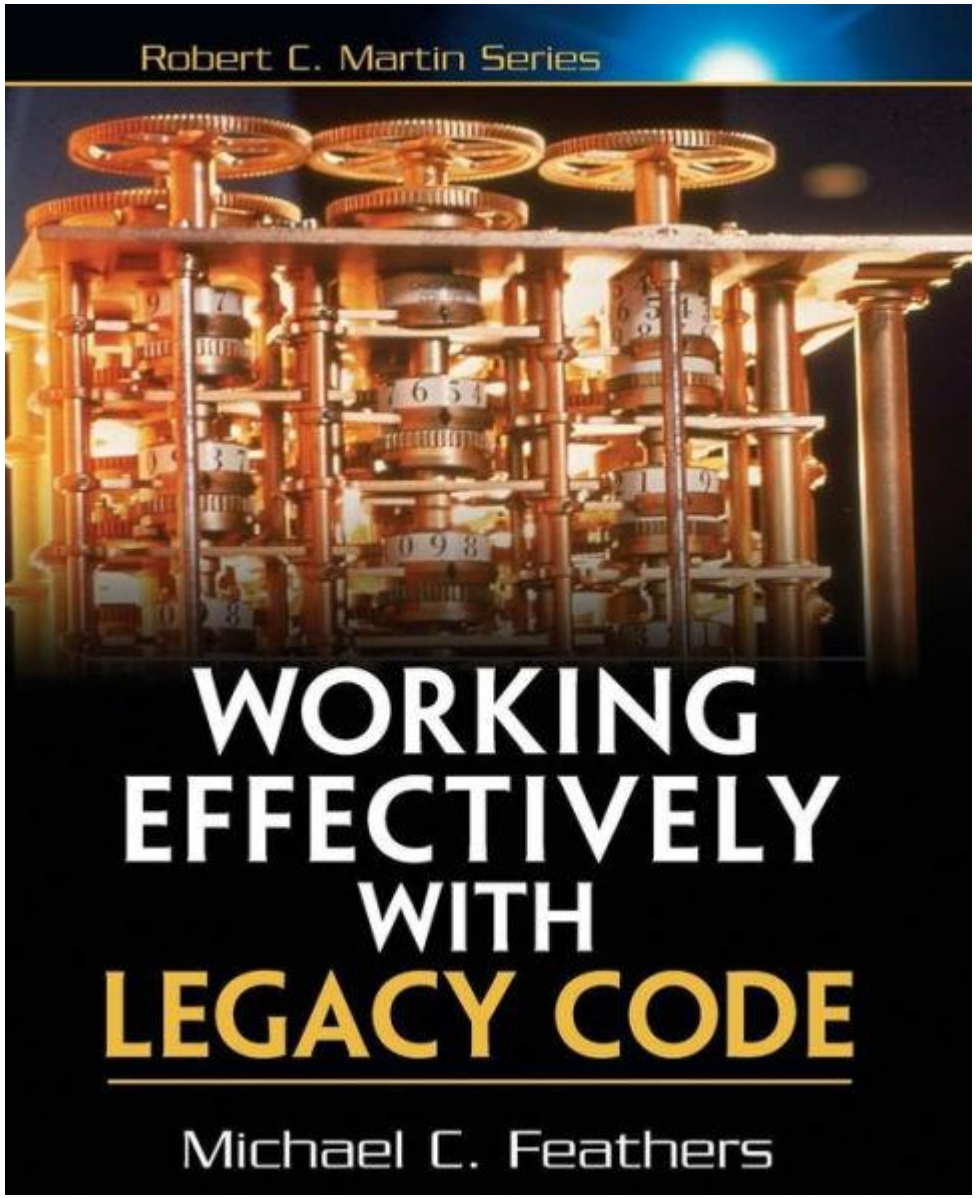
Seams – Point that allows you to modify behavior without modifying code

Types:

- Object
- Preprocessor
- Linker
- Template?

When?

- Runtime
- Preprocessor Time
- Link Time
- Compile Time



AN EXAMPLE...

```
class Uart {
public:
  void read(uint8_t* out, uint32_t size);
};
```

```
int16_t Sensor::getReading()
{
  int16_t reading;
  uart_->read(
    reinterpret_cast<uint8_t*>(&reading),
    sizeof(reading)
  );
  return reading;
}
```

```
class Sensor {
public:
  Sensor(Uart* uart)
    : uart_{ uart }
  {
  }

  int16_t getReading();

private:
  Uart* uart_;
};
```

PRODUCTION CODE

```
int main() {
  Uart uart;
  Sensor sensor{ &uart };

  std::cout << sensor.getReading() << std::endl;

  return 0;
}
```

VIRTUAL DEPENDNECY

```cpp
class Uart {
public:
  virtual ~Uart() = default;
  virtual void read(uint8_t *out, uint32_t size);
};
```

```cpp
#include "Uart.hpp"

class MockUart : public Uart {
public:
  MockUart();
  ~MockUart() override;
  MOCK_METHOD(void, read, (uint8_t * out, uint32_t size), (override));
};
```

How did we do?

- Overhead in production code
- Modify existing code
- No 3rd party code mocking
  - Wrappers?

UNIT TEST

```
TEST(TestSensor, Reading_is_parsed_as_little_endian)
{
  using namespace ::testing;
  NiceMock<MockUart> uart;
  Sensor sensor{ &uart };

  ON_CALL(uart, read(_, 2))
    .WillByDefault(Invoke([](uint8_t* buf, uint32_t size) {
      (void)size;
      buf[0] = 0x34;
      buf[1] = 0x12;
    })
  );

  ASSERT_EQ(0x1234, sensor.getReading());
}
```

TEST VIRTUAL

```
#pragma once

#if TESTING
# define TEST_VIRTUAL virtual
#else
# define TEST_VIRTUAL
#endif
```

```
#include "TestVirtual.hpp"

class Uart {
public:
   TEST_VIRTUAL ~Uart() = default;
   TEST_VIRTUAL void read(uint8_t* out, uint32_t size);
};
```

How did we do?

- Compile definition
  - 2 types of Uart
- Modify existing code
  - A hack only a mother could love
- No 3rd party code mocking
  - Wrappers…

TEMPLATE

```
template <typename TUart = Uart>
class Sensor {
public:
  Sensor(TUart* uart)
    : uart_{ uart }
  {
  }

  int16_t getReading() {
    int16_t reading;
    uart_->read(reinterpret_cast<uint8_t*>(&reading), sizeof(reading));
    return reading;
  }

private:
  TUart* uart_;
};
```

How did we do?

- Compile times?
- Error messages?
- Modify existing code
- Mock 3rd party code!
- 2 types of Sensor

UNIT TEST WITH TEMPLATE

```
TEST(TestSensor, Reading_is_parsed_as_little_endian)
{
  using namespace ::testing;
  NiceMock<MockUart> uart;
  Sensor<MockUart> sensor{ &uart };

  ON_CALL(uart, read(_, 2))
    .WillByDefault(Invoke([](uint8_t* buf, uint32_t size) {
      (void)size;
      buf[0] = 0x34;
      buf[1] = 0x12;
    })
  );

  ASSERT_EQ(0x1234, sensor.getReading());
}
```

EXTERN TEMPLATE

```hpp
// .hpp
template <typename TUart = Uart>
class Sensor {
public:
  Sensor(TUart* uart)
    : uart_{ uart }
  {
  }

  int16_t getReading();

private:
  TUart* uart_;
};
```

```tpp
// .tpp
#include "Sensor.hpp"

template <typename T>
int16_t Sensor<T>::getReading()
{
  int16_t reading;
  uart_->read(
    reinterpret_cast<uint8_t*>(&reading),
    sizeof(reading)
  );
  return reading;
}
```

```cpp
// .cpp
#include "Sensor.hpp"
#include "Sensor.tpp"
template class Sensor<Uart>; // explicit instantiation
```

UNIT TEST WITH EXTERN TEMPLATE

```
#include "Sensor.tpp"
template class Sensor<MockUart>;
```

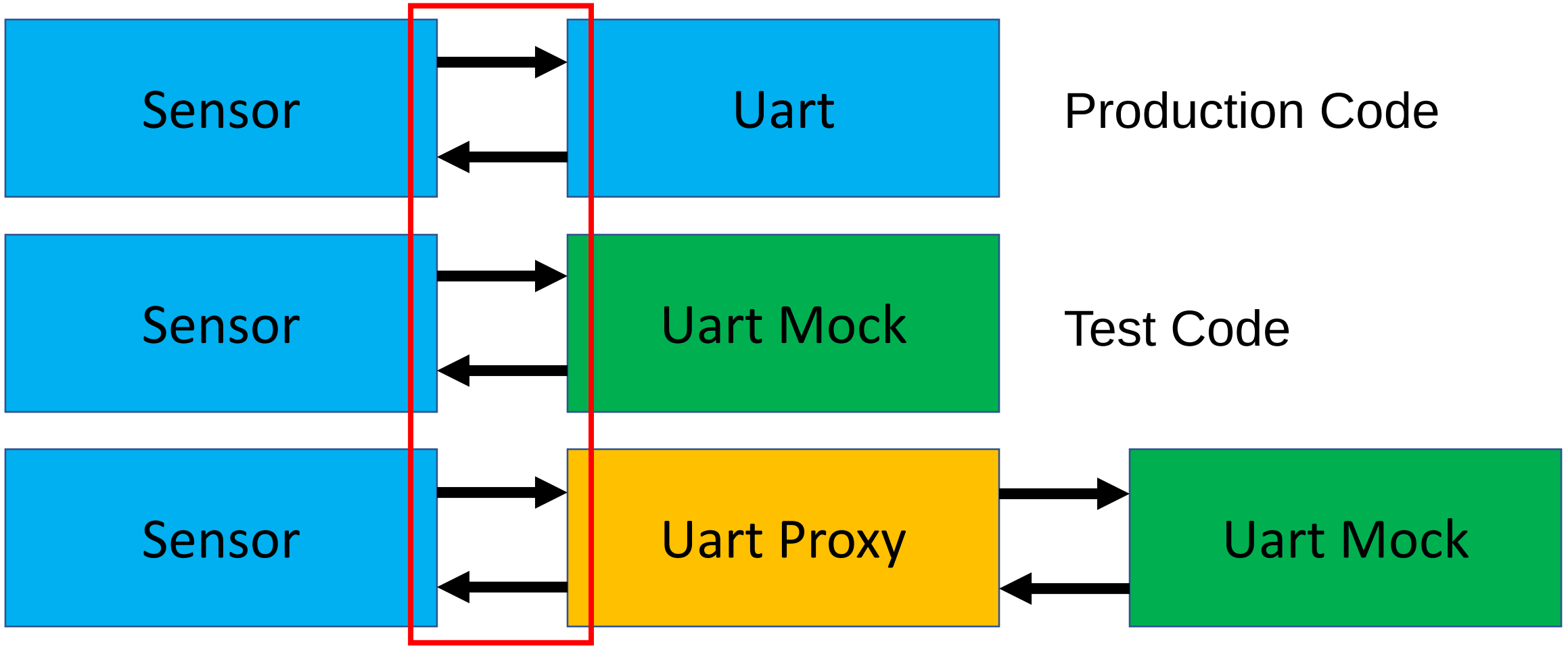
```
TEST(TestSensor, Reading_is_parsed_as_little_endian)
{
  using namespace ::testing;
  NiceMock<MockUart> uart;
  Sensor<MockUart> sensor{ &uart };

  ON_CALL(uart, read(_, 2))
    .WillByDefault(Invoke([](uint8_t* buf, uint32_t size) {
      // ...
    })
  );

  ASSERT_EQ(0x1234, sensor.getReading());
}
```

EXTERN TEMPLATE REVIEW

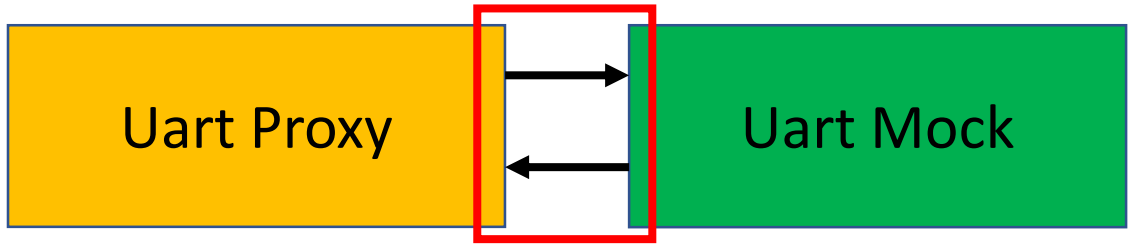
How did we do?

- Compile times solved!
- Error messages?
- .tpp files now?
- Modify existing code
- Mock 3rd party code!

OUR SEAMS


Sensor
Uart
Production Code
Sensor
Uart Mock
Test Code
Sensor
Uart Proxy
Uart Mock

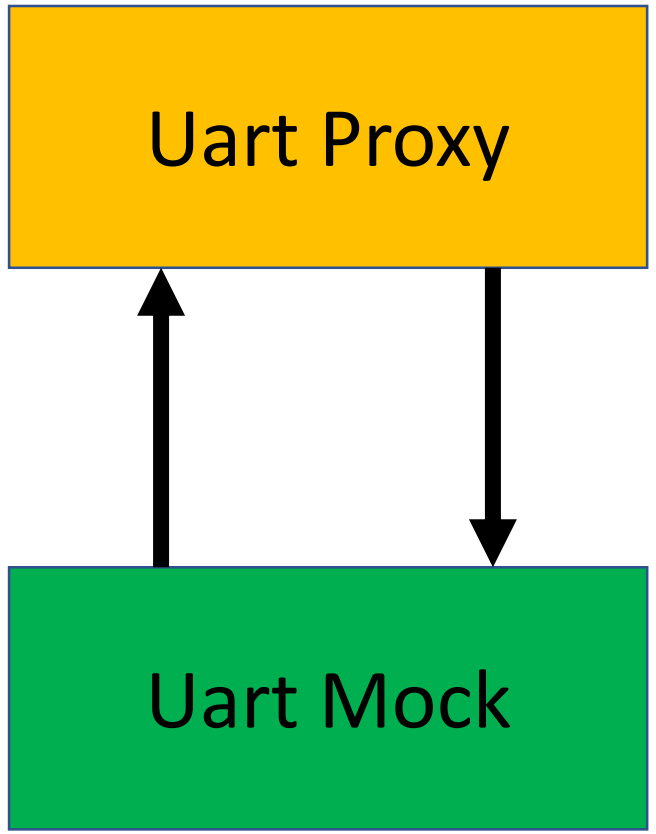
```cpp
template <typename T>
class ProxyMock {
public:
  ProxyMock(const void* instance)
    : instance_{ instance }
  {
    map_[instance] = static_cast<T*>(this);
  }

  static T& getMockFor(const void* instance) { return static_cast<T&>(*map_.at(instance)); }

  ~ProxyMock() {
    auto it = map_.find(instance_);
    if (it != map_.end()) { map_.erase(it); }
  }

private:
  const void* const                instance_;
  static std::map<const void*, T*> map_;
};
template <typename T>
std::map<const void*, T*> ProxyMock<T>::map_;
```



LINKER SEAM IN ACTION

```
void Uart::read(uint8_t* out, uint32_t size)
{
  return ProxyMock<MockUart>::getMockFor(this).read(out, size);
}
```

```
class MockUart : public ProxyMock<MockUart> {
public:
  MockUart(Uart* impl)
    : ProxyMock<MockUart>{ impl }
  {
  }

  virtual ~MockUart();
  MOCK_METHOD(void, read, (uint8_t* out, uint32_t size), ());
};
```



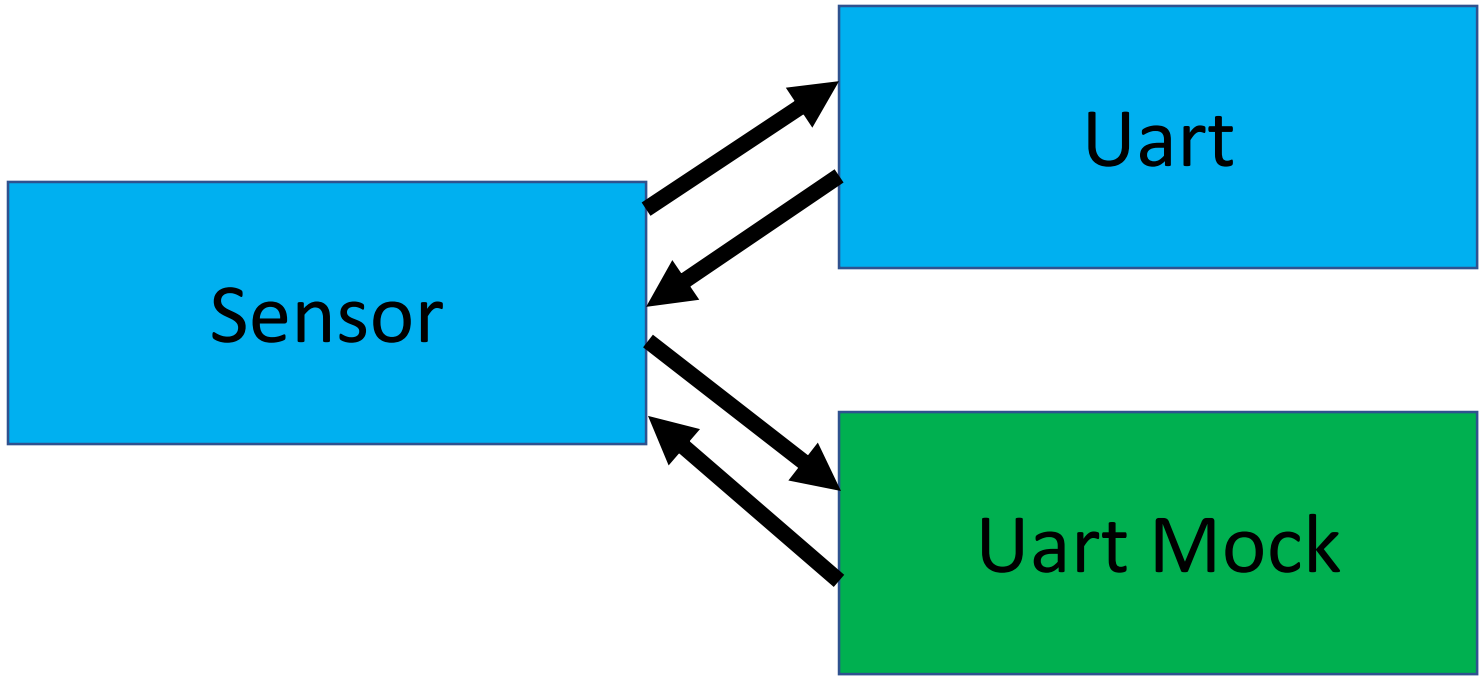
UNIT TEST WITH PROXY MOCK

```
TEST(TestTemperatureSensor, Temperature_is_parsed_as_little_endian)
{
  using namespace ::testing;
  Uart uart;
  NiceMock<MockUart> mock_uart{ &uart };
  Sensor sensor{ &uart };

  ON_CALL(mock_uart, read(_, 2))
    .WillByDefault(Invoke([](uint8_t* buf, uint32_t size) {
      (void)size;
      buf[0] = 0x34;
      buf[1] = 0x12;
    })
  );

  ASSERT_EQ(0x1234, sensor.getReading());
}
```

HOW DID WE DO?

- No touching existing code!
- Mock 3rd party code!
- Can work for any non-inlined function
- Need to understand your build system
- Need a generator… stay tuned!

Monkey Patching



HIPPOMOCKS

```
TEST(TestSensor, Reading_is_parsed_as_little_endian)
{
  MockRepository mocks;
  mocks.OnCallFunc(uart::read)
       .Match([](uint8_t* out, uint32_t size) {
         (void)out;
         return size == 2;
       })
       .Do([](uint8_t* buf, uint32_t size) {
         (void)size;
         buf[0] = 0x34;
         buf[1] = 0x12;
       });

  Sensor sensor;
  ASSERT_EQ(0x1234, sensor.getReading());
}
```

```
namespace uart {
void read(uint8_t* out, uint32_t size);
}
```

How did we do?

- No mock generation!
  - …for free, static, and virtual functions
- No non-virtual member function support
- Non-Inlined functions

Q & A

Thank You!