Accelerated
TDD
day 2
@phil_nash

Recap

Why Accelerated

… and how?

The TDD cycle

good

What is a test, anyway?

^

You're not as smart as you think you are

Single Responsibility

Loose Coupling

Dependency Inversion

The Cat-flap principle

Tests should:

Have a good name

State expectations

Have a single "Logical" Assert

Use Public Interface

Have low Cyclomatic Complexity

Unit Tests should be reproducible and deterministic

"Arrange" → "Act" → "Assert"

Always have A Reason

leftpad
snakes &
ladders

snakes & ladders

revisited

Need to be able to:

Have a game object

Add players

Each player has a colour

Players take turns rolling a die

Track moves and game state

Why do cars have brakes?

Identify our heroes

Michael Feathers

Kevlin Henney

Dan North

Steve Freeman

Nat Pryce

Clare Macrae

James Grenning

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

"Tests that do these things aren't bad. Often they are worth writing, and they can be written in a unit test harness. However, it is important to be able to separate them from true unit tests so that we can keep a set of tests that we can run fast whenever we make our changes."

Michael Feathers - A Set of Unit Testing Rules

Beyond Unit Tests

Integration

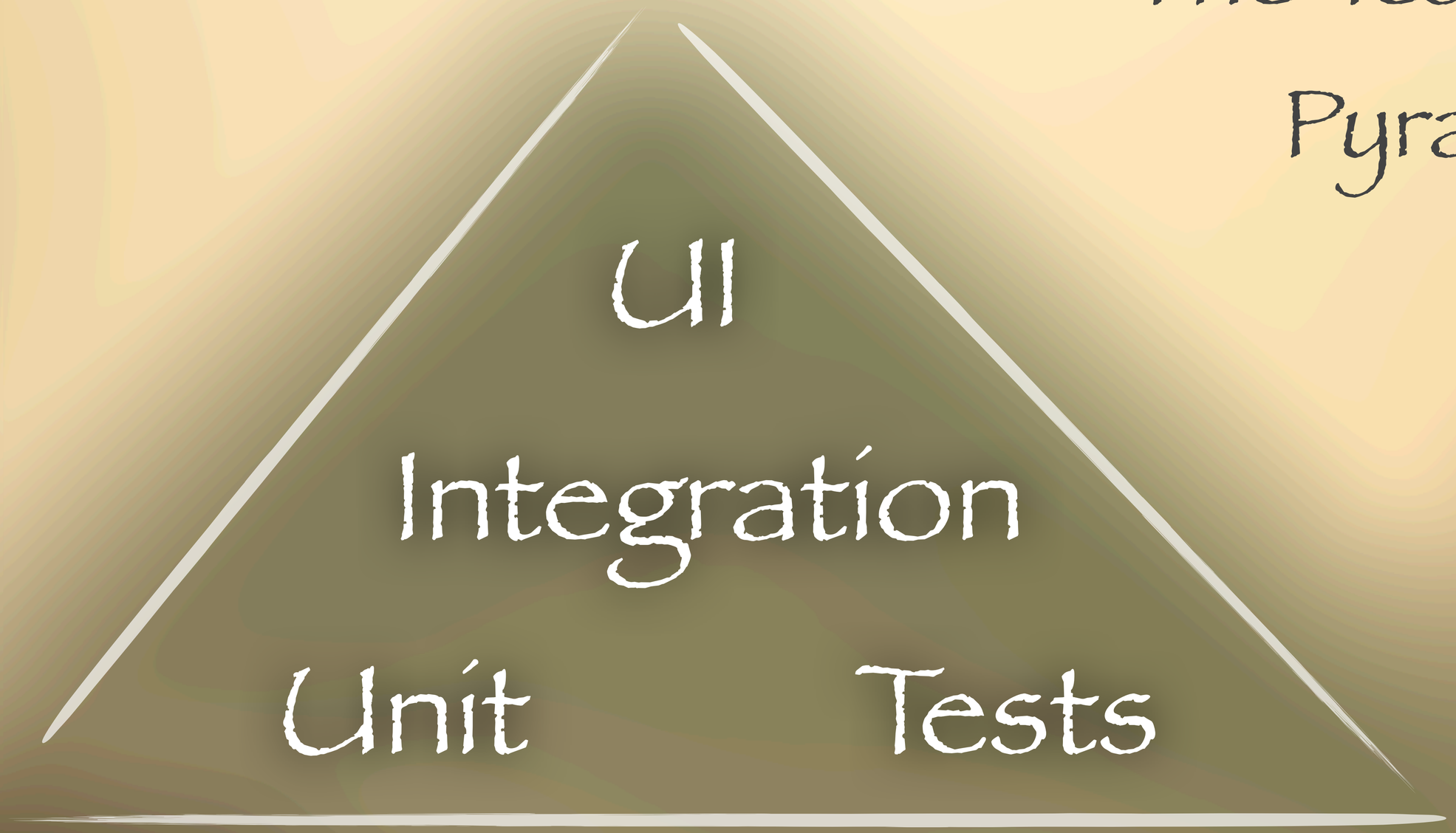
Unit Tests

UI

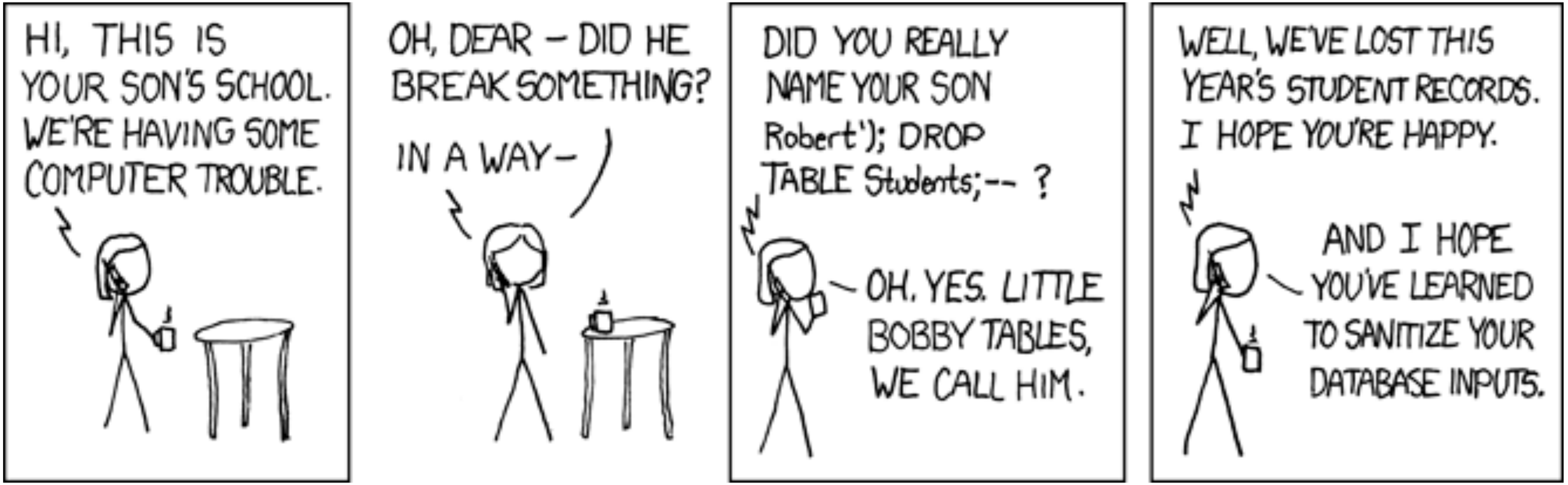
Integration

Unit Tests

The Testing Pyramid



Property-Based Testing


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Fuzz Testing

Programming Wisdom @CodeWisdom

"There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability , in simplicity." - Frederick P. Brooks

1:58 PM - 8 Dec 2017

Programming Wisdom @CodeWisdom

"There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability , in simplicity." - Frederick P. Brooks

1:58 PM - 8 Dec 2017

Chandler Carruth

@chandlerc1024

Following

I honestly think the sanitizers (ASan, TSan, MSan, UBSan), especially combined with coverage directed, continuous fuzz testing come really, really close. @kayseesee

Mutation Testing

Mocking

Test Doubles

Mocks

Fakes

Stubs

Stubs

Stand in for a real objects just enough to compile and run the tests

A stubbed method will return, at most, a “canned answer”

```
class StubbedLogger : public Logger {
    virtual void log(
        LogLevel,
        std::string_view ) {}
};
```

Fakes

Fakes will actually do some work, but in a cut down or short-cut way (with no outside dependencies).

This may include collecting information that can be tested

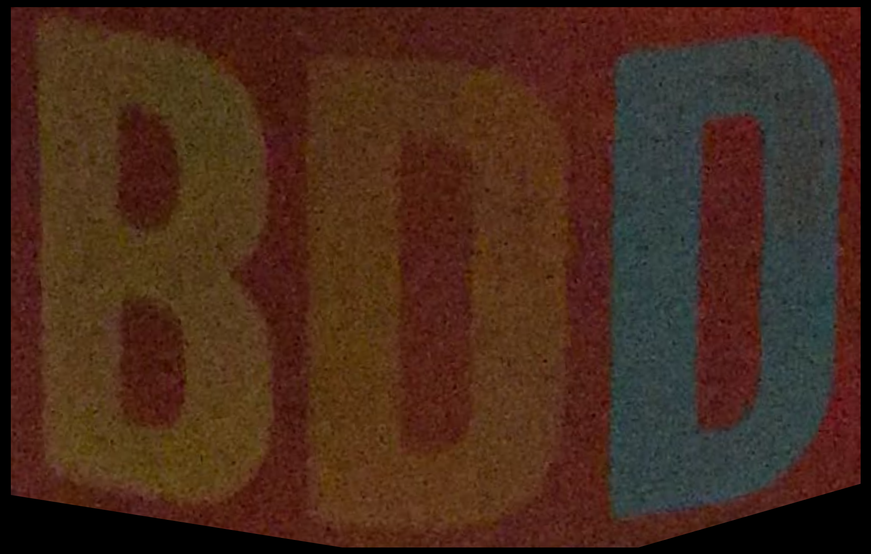
```
class FakeLogger : public Logger {
    std::ostringstream m_oss;
    virtual void log(
      LogLevel level,
      std::string_view line ) {
        m_oss << line << "\n";
    }
public:
    std::string get() const {
        return m_oss.str();
    }
};
```

Mocks

Mocks are fakes which allow you to set exceptions on what calls should be made on the mock object.

Because they provide configurable implementations of mocked methods, they can be useful for creating ad-hoc fakes, too.

```
class MockLogger : public
  trompeloeil::mock_interface
  <Logger>
{
    IMPLEMENT_MOCK2(log);
};

// …

REQUIRE_CALL( mockLogger,
 log( LogLevel::Info, "hello" ) );
```


BDD

{BDD}
BEER DRIVEN DEVELOPMENT
headforwards™

{BDD}
BEER DRIVEN DEVELOPMENT
headforwards™

The deeper I got into TDD, the more I felt that my own journey had been less of a wax-on, wax-off process of gradual mastery than a series of blind alleys. I remember thinking “If only someone had told me that!” far more often than I thought “Wow, a door has opened.” I decided it must be possible to present TDD in a way that gets straight to the good stuff and avoids all the pitfalls.

My response is behaviour-driven development (BDD). It has evolved out of established agile practices and is designed to make them more accessible and effective for teams new to agile software delivery. Over time, BDD has grown to encompass the wider picture of agile analysis and automated acceptance testing

Dan North - Introducing BDD

Communication
Ubiquitous language

User Stories

Title

Narrative

Acceptance Criteria (scenarios)

Arrange

Act

Assert

Arrange	Init your state
Act	Perform an action
Assert	Check the result

Given	Init your state
When	Perform an action
Then	Check the result

Unit Tests should be reproducible and deterministic

```
TEST_CASE( "Most recently used list" ) {

    MRUList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

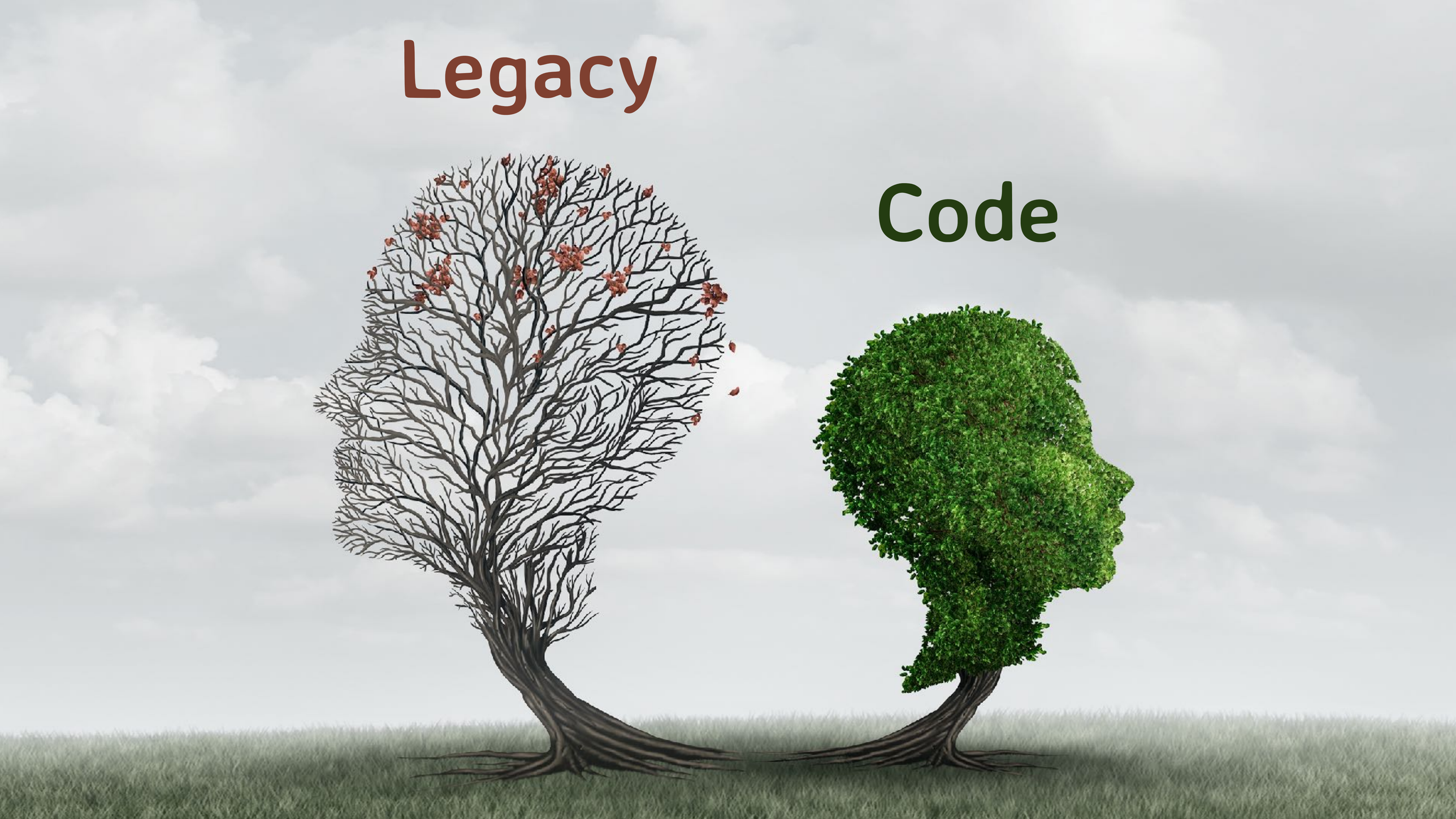
"Arrange"

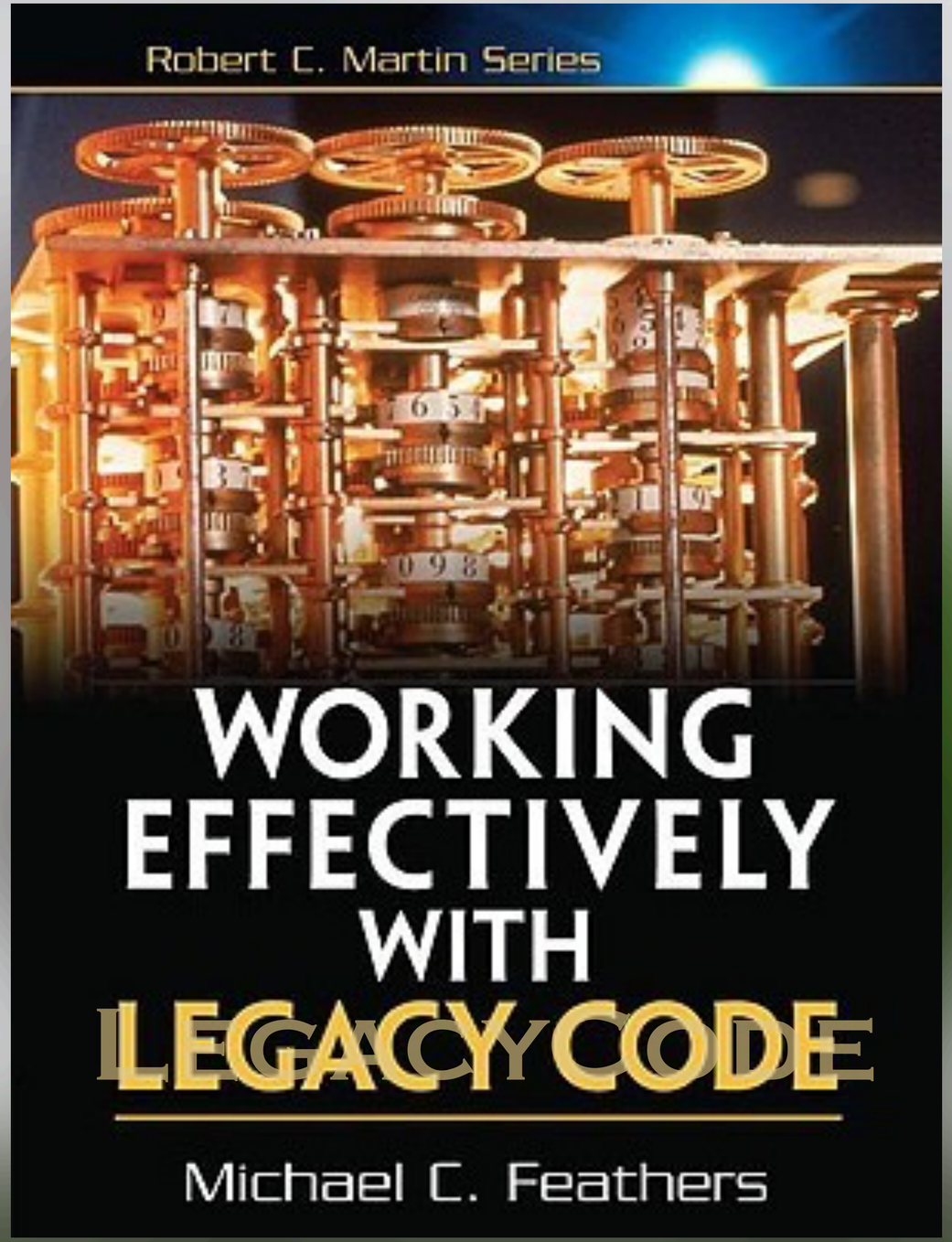
"Act"

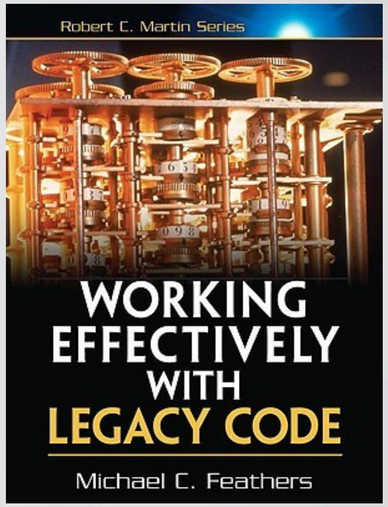
"Assert"

```
SCENARIO( "Most recently used list" ) {
    GIVEN( "an empty MRUList" ) {
        MRUList<std::string> list;

        THEN( "it has no elements” ) {
            REQUIRE( list.empty() );
            REQUIRE( list.size() == 0 );
        }
        WHEN( "an item is added” ) {
            list.add( “item1 ");
            THEN( "it is not longer empty” ) {
                REQUIRE( list.empty() == false );
                REQUIRE( list.size() == 1 );
            }
        }
    }
}
```

Deliberate
~~Practice~~
Learning


Legacy
Code


Robert C. Martin Series
WORKING EFFECTIVELY WITH LEGACY CODE
Michael C. Feathers



Seams

Sprout Method

Dealing with dependencies

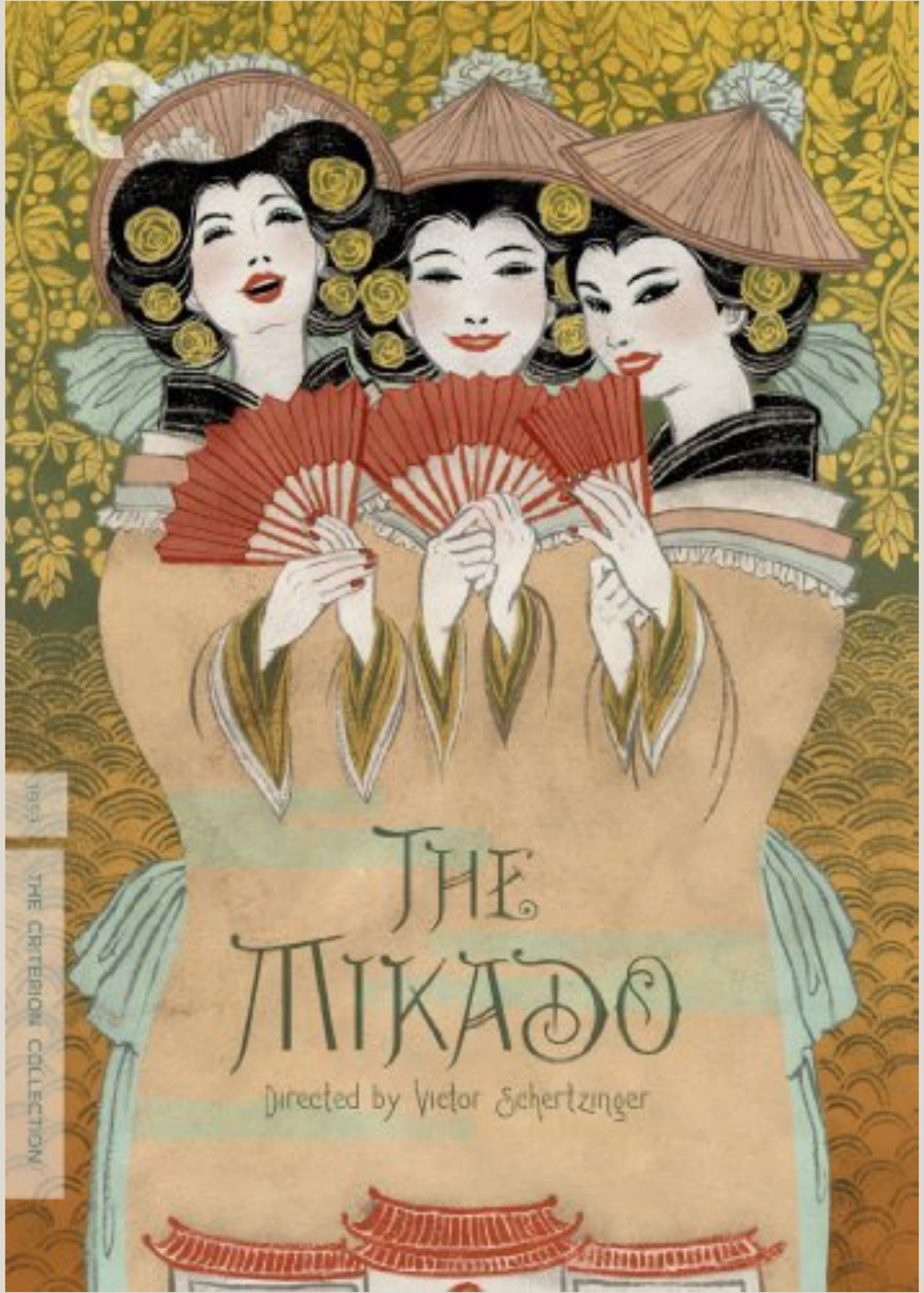
Characterisation Tests*

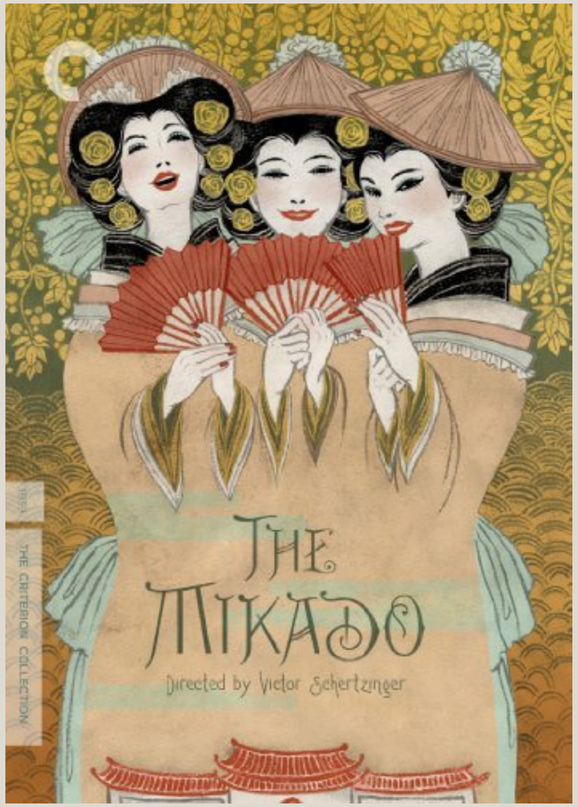
Automated Refactorings

Lean on the compiler

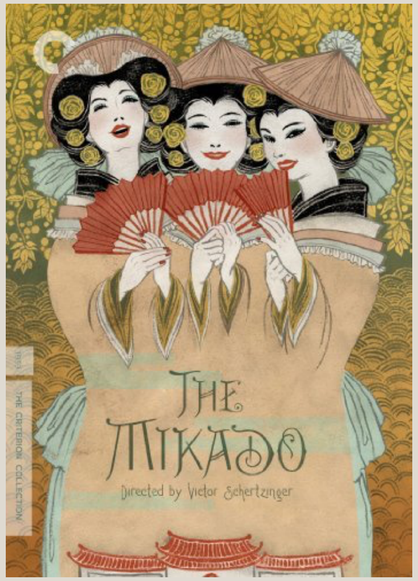
Conquer From Without

Risk
ARMIES
N. AMERICA 5
ASIA 7
S. AMERICA 2
EUROPE 5
AFRICA 3
AUSTRALIA 2
ATLANTIC OCEAN
PACIFIC OCEAN
INDIAN OCEAN
Brazil
Argentina
Venezuela
Egypt
East Africa
Congo
Madagascar
Ukraine
Western Europe
Southern Europe
Middle East
China
India
Siam
Indonesia
New Guinea
Eastern Australia
4
6
8
10
15
20
25
30
35
40
45
50
55
60

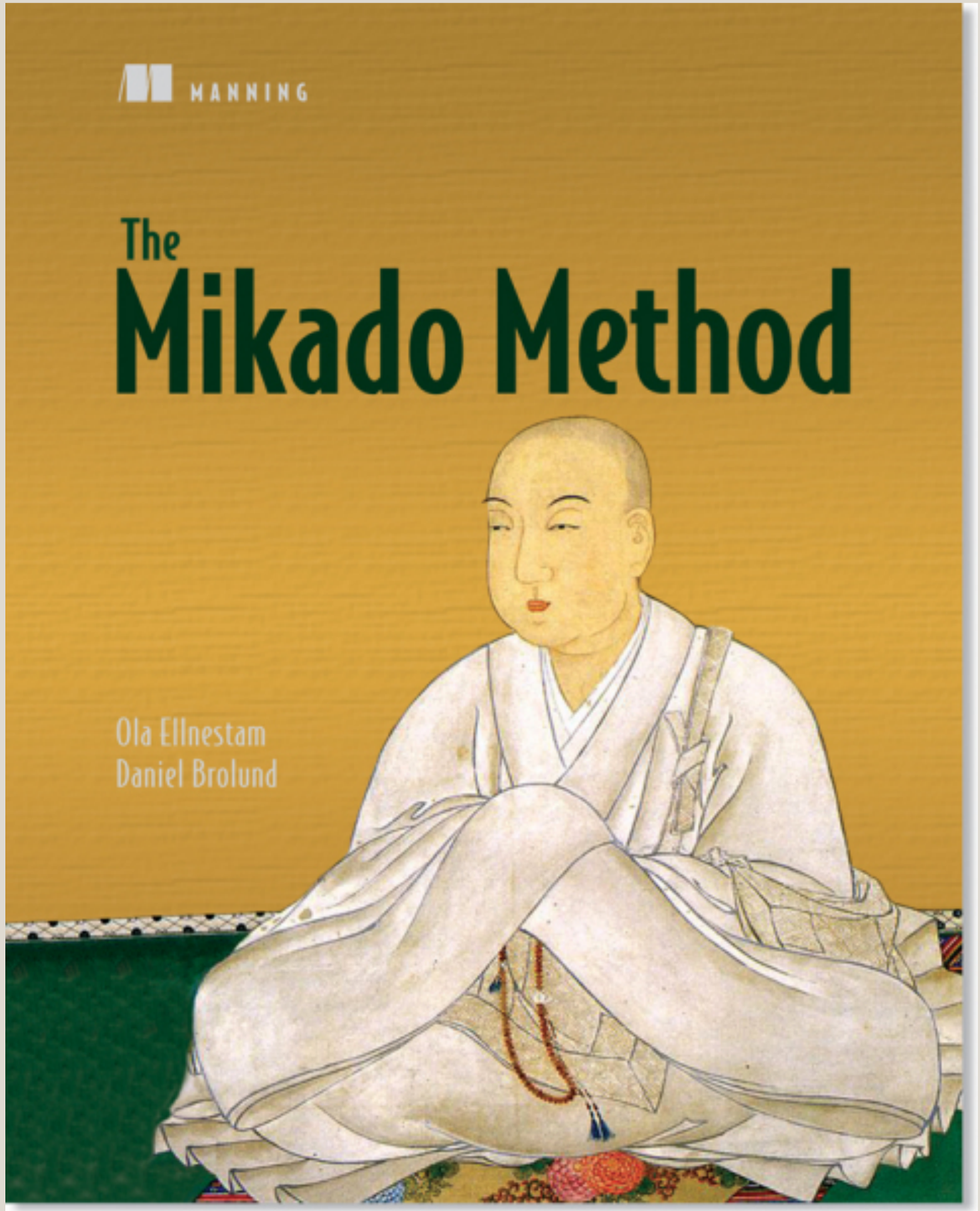

THE MIKADO
Directed by Victor Schertzinger
1939
THE CRITERION COLLECTION

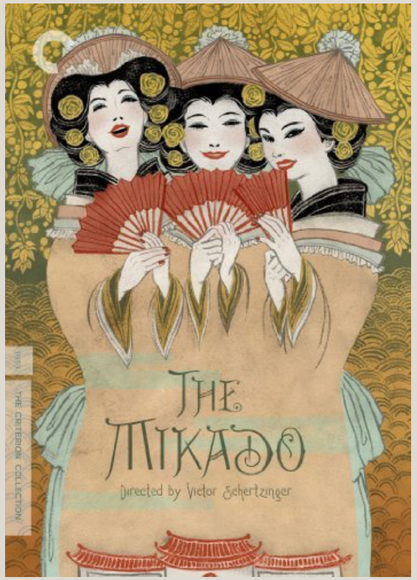

THE MIKADO
Directed by Victor Schertzinger
THE CRITERION COLLECTION

MIKADO
LU
Glico
MIKADO
MILK
CHOCOLATE
Each 2.3g (biscuit) contains
Calories 11 <1%
Sugar 0.8g <1%
Fat 0.4g <1%
Saturates 0.3g 1%
Salt trace <1%
of an adult's guideline daily amount

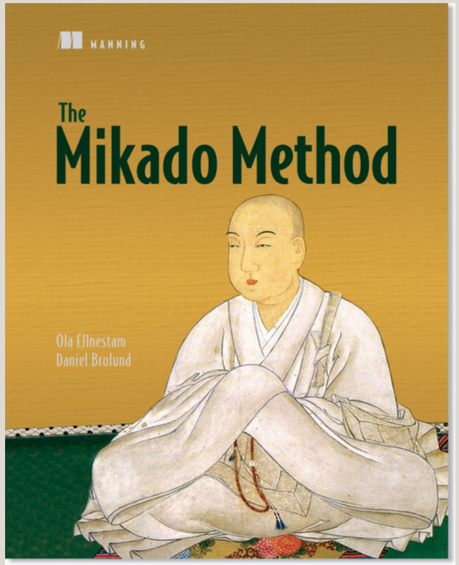

THE
MIKADO
Directed by Victor Schertzinger
THE CRITERION COLLECTION

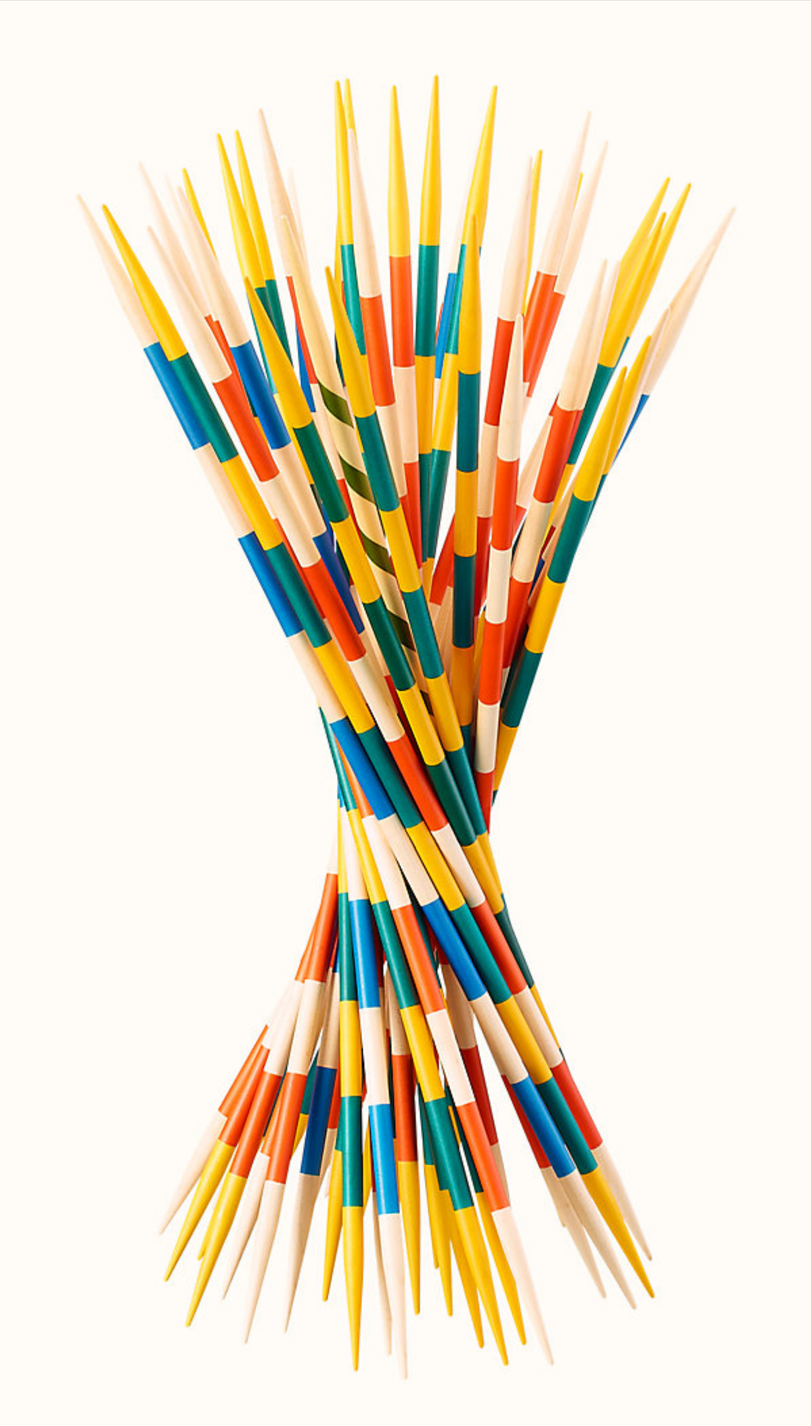
LU
Glico
MIKADO
MILK
CHOCOLATE
Each 2.3g (biscuit) contains
Calories 11 <1%
Sugar 0.8g <1%
Fat 0.4g <1%
Saturates 0.3g 1%
Salt trace <1%
of an adult's guideline daily amount

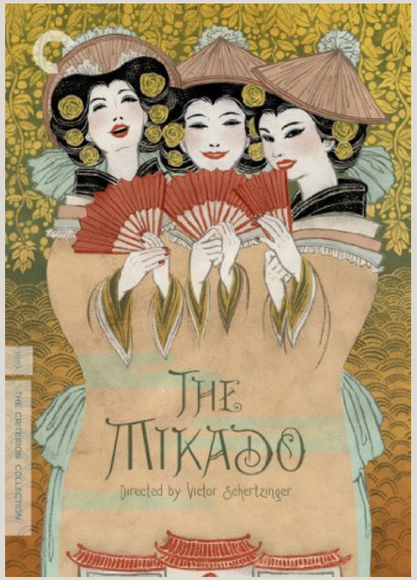

MANNING
The
Mikado Method
Ola Ellnestam
Daniel Brolund


THE MIKADO
Directed by Victor Schertzinger
THE CRITERION COLLECTION

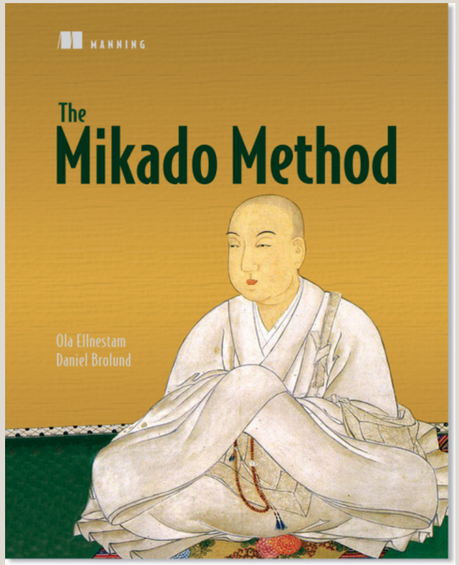
MIKADO
LU
Glico
MIKADO
MILK
CHOCOLATE
Each 2.3g (biscuit) contains
Calories 11 <1%
Sugar 0.8g <1%
Fat 0.4g <1%
Saturates 0.3g 1%
Salt trace <1%
of an adult's guideline daily amount

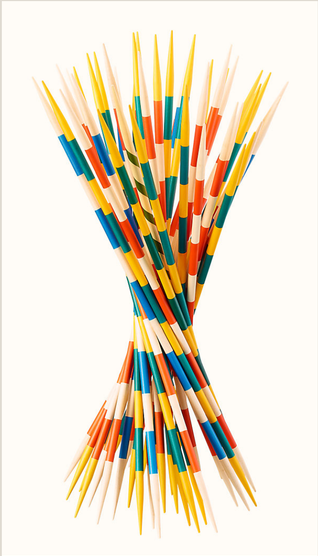

MANNING
The
Mikado Method
Ola Ellnestam
Daniel Brolund

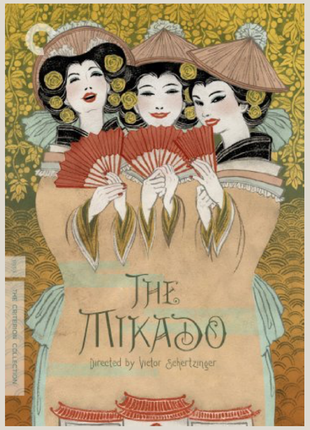



THE MIKADO
Directed by Victor Schertzinger
THE CRITERION COLLECTION

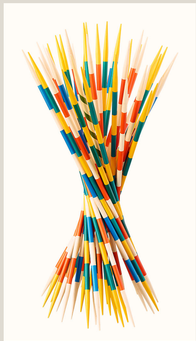
MIKADO
LU
Glico
MIKADO
MILK
CHOCOLATE
Each 2.3g (biscuit) contains
Calories 11 <1%
Sugar 0.8g <1%
Fat 0.4g <1%
Saturates 0.3g 1%
Salt trace <1%
of an adult's guideline daily amount

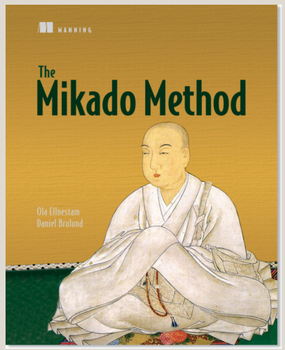

MANNING
The
Mikado Method
Ola Ellnestam
Daniel Brolund

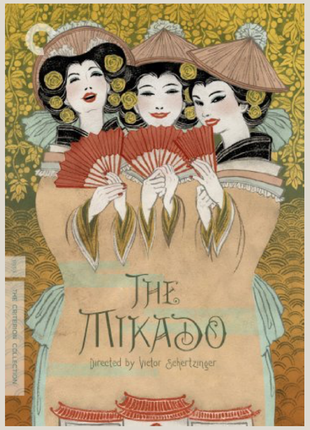



THE
MIKADO
Directed by Victor Schertzinger

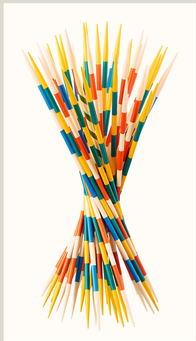
LU
Glico
MIKADO
MILK
CHOCOLATE
Each 2.3g (biscuit) contains
Calories Sugar Fat Saturates Salt
11 0.8g 0.4g 0.3g trace
<1% <1% <1% 1% <1%
of an adult's guideline daily amount

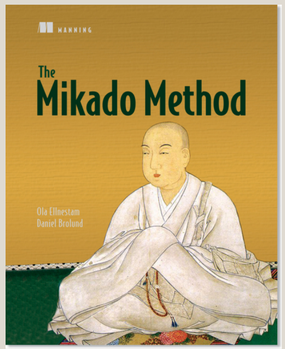


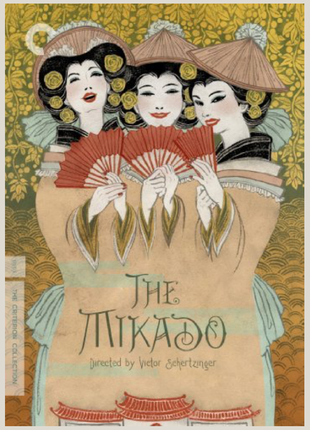

MANNING
The
Mikado Method
Ole Ellnestam
Daniel Brolund


THE
MIKADO
Directed by Victor Schertzinger

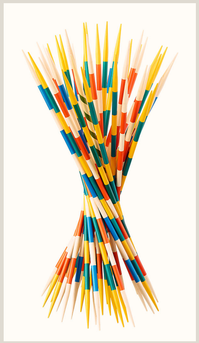
LU
Glico
MIKADO
MILK
CHOCOLATE

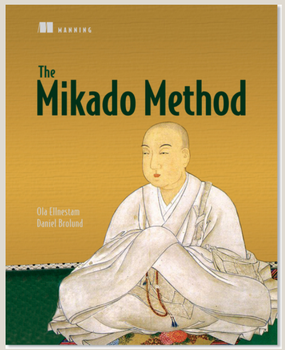


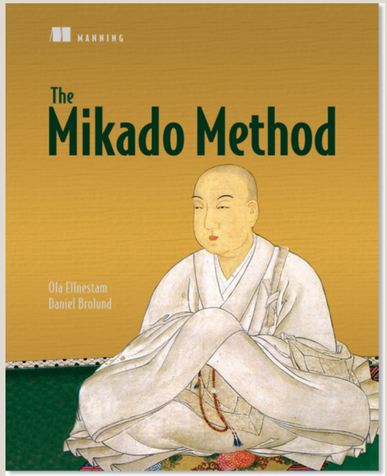

MANNING
The
Mikado Method
Ole Ellnestam
Daniel Brolund

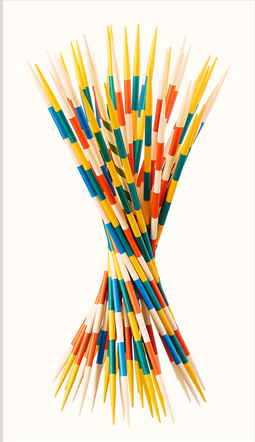

THE
MIKADO
Directed by Victor Schertzinger

LU
MIKADO
MILK
CHOCOLATE




MANNING
The
Mikado Method
Ole Ellnestam
Daniel Brolund


MANNING
The
Mikado Method
Ola Ellnestam
Daniel Brolund



THE
MIKADO

LU
MIKADO
MILK
CHOCOLATE

APPROVED
APPROVED
APPROVED

github.com/approvals/ApprovalTests.cpp

Approval Tests for C++

build passing | build passing | License Apache 2.0 | Contributor Covenant v1.4 adopted

Download the latest version (v.5.0.0) of the **single header file** here.

Contents

- What are Approval Tests?
- Requirements
- Getting Started
- ToString (ostream insertion)
- What's new?
- Feedback

What are Approval Tests?

Also known as **Golden Master Tests** or **Snapshot Testing**, Approval Tests are an alternative to asserts. If you are unfamiliar with them, we have an overview and a tutorial.

They are great for testing objects with lots of fields, or lists of objects.

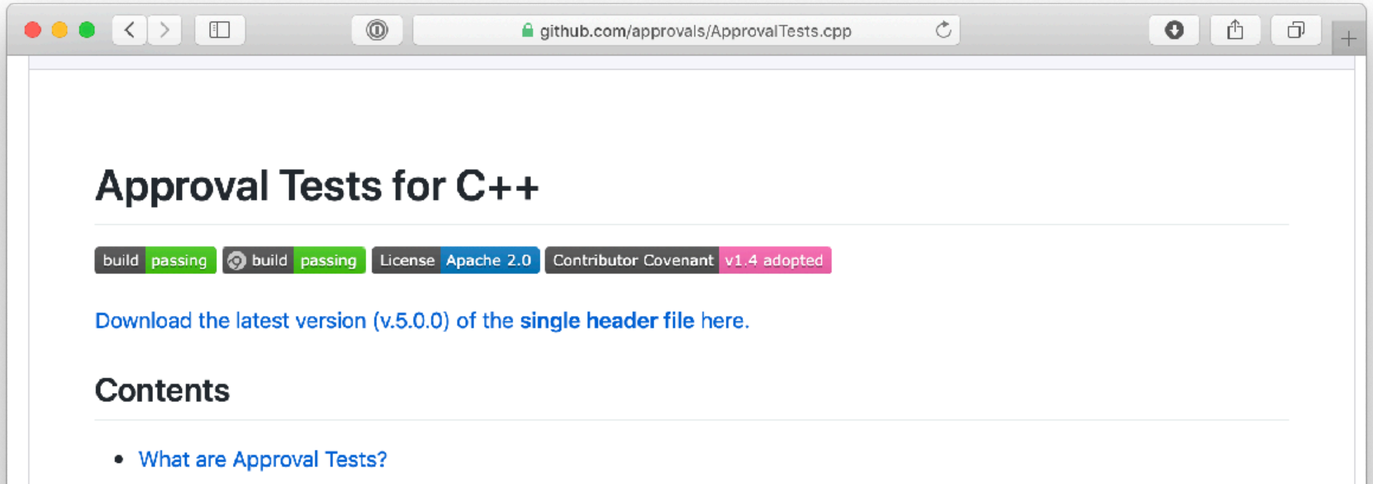
Requirements

- C++11 (or above) compiler
- Mac/Linux/Windows
- One of: GoogleTest, Catch1, Catch2, doctest

Getting Started

APPROVED
APPROVED
APPROVED

Quickly testing legacy code - Clare Macrae [C++ on Sea 2019]
Quickly testing legacy code - Clare Macrae
Approvals Catch2 Test
• A test file (Test02.cpp)
#include "ApprovalTests.hpp"
#include "Catch.hpp"
// Approvals test - test static value, for demo purposes
TEST_CASE("TestFixedInput")
{
Approvals::verify("Some\nMulti-line\noutput");
}
@ClareMacraeUK
46
cpponsea.uk
@cpponsea
18:22 / 1:00:12


github.com/approvals/ApprovalTests.cpp
Approval Tests for C++
build passing
build passing
License Apache 2.0
Contributor Covenant v1.4 adopted
Download the latest version (v.5.0.0) of the single header file here.
Contents
What are Approval Tests?

Taming singletons

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

"Tests that do these things aren't bad. Often they are worth writing, and they can be written in a unit test harness. However, it is important to be able to separate them from true unit tests so that we can keep a set of tests that we can run fast whenever we make our changes."

Michael Feathers - A Set of Unit Testing Rules

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

"Tests that do these things aren't bad. Often they are worth writing, and they can be written in a unit test harness. However, it is important to be able to separate them from true unit tests so that we can keep a set of tests that we can run fast whenever we make our changes."

Michael Feathers - A Set of Unit Testing Rules

The first rule of
Singleton Club:
No singletons!

Fight singletons with singletons

```
class Cthulhu {
    Cthulhu() = default; // private ctor
public:
    Cthulhu( Cthulhu const& ) = delete;

    static Cthulhu const &get() {
        static Cthulhu s_instance;
        return s_instance;
    }
};
```

```
auto const& singleton = Cthulhu::get();
```

```
class Cthulhu {
    static Cthulhu const*& getInternal() {
        static Cthulhu const* instance = new Cthulhu;
        return instance;
    }
protected:
    Cthulhu() = default;
public:
    Cthulhu( Cthulhu const& ) = delete;

    static Cthulhu const& get() {
        return *getInternal();
    }
};
```

```
class Cthulhu {
    static Cthulhu const*& getInternal() {
        static Cthulhu const* instance = new Cthulhu;
        return instance;
    }
protected:
    Cthulhu() = default;
public:
    Cthulhu( Cthulhu const& ) = delete;

    friend Cthulhu const& getCthulhu() {
        return *Cthulhu::getInternal();
    }
};
```

```
class Cthulhu {
    static Cthulhu const*& getInternal() {
        static Cthulhu const* instance = new Cthulhu;
        return instance;
    }
protected:
    Cthulhu() = default;
public:
    friend Cthulhu const& getCthulhu() {
        return *Cthulhu::getInternal();
    }
    friend Cthulhu const* setCthulhu( Cthulhu const* newInstance ) {
        auto prev = Cthulhu::getInternal();
        Cthulhu::getInternal() = newInstance;
        return prev;
    }
};
```

```
class Cthulhu {
    static Cthulhu const*& getInternal() {
        static Cthulhu const* instance = new Cthulhu;
        return instance;
    }
protected:
    Cthulhu() = default;
public:
    friend Cthulhu const& getCthulhu();
    friend Cthulhu const* setCthulhu( Cthulhu const* newInstance );
};
```

```
Cthulhu const& getCthulhu() {
    return *Cthulhu::getInternal();
}
Cthulhu const* setCthulhu( Cthulhu const* newInstance ) {
    auto prev = Cthulhu::getInternal();
    Cthulhu::getInternal() = newInstance;
    return prev;
}
```

```
class CthuluSwapper {
    Cthulhu const* original;
public:
    CthuluSwapper( Cthulhu const* newInstance )
    : original( setCthulhu( newInstance ) )
    {}
    ~CthuluSwapper() {
        delete setCthulhu( original );
    }
};
```

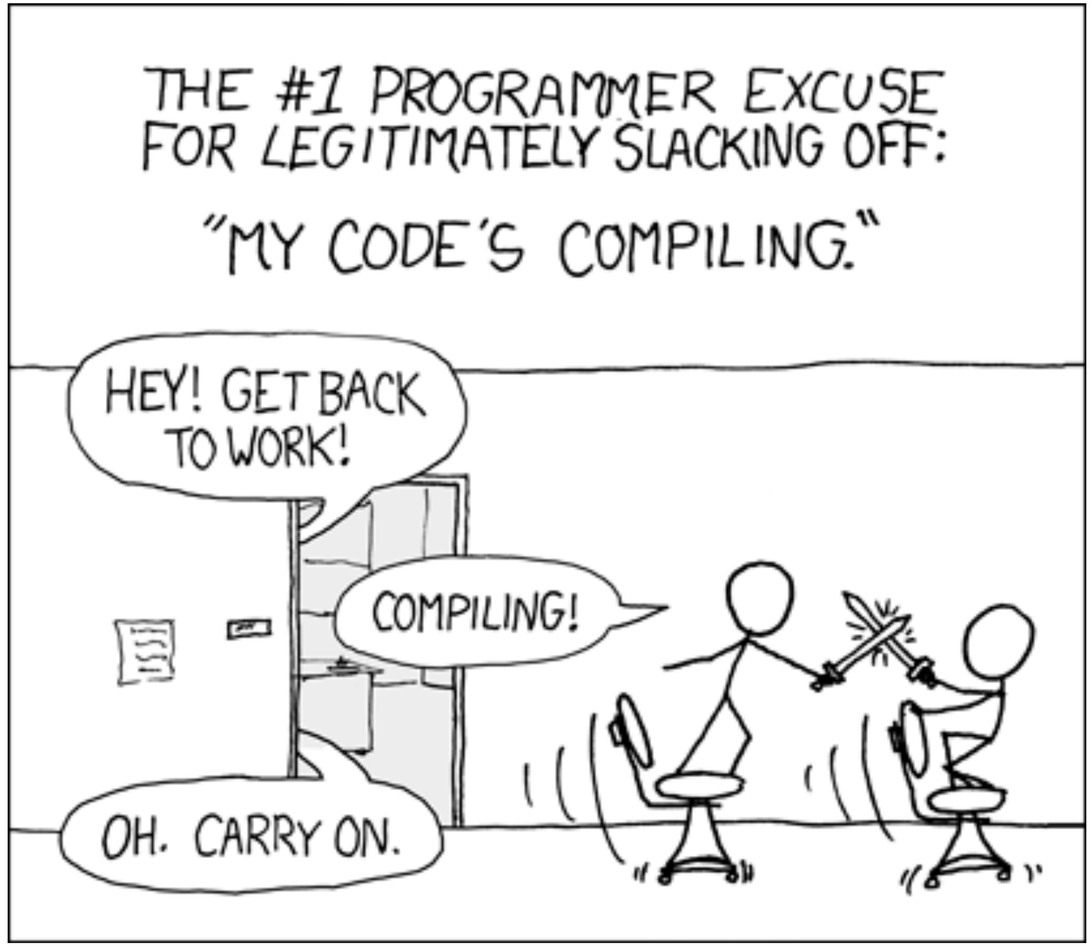
```
auto const& singleton = getCthulhu();
```

```
class MockCthulu : public Cthulhu {
    // ...
};
```

```
{

   CthuluSwapper swapper( new MockCthulu );

   // ...
   auto const& mockSingleton = getCthulhu();

} // original singleton restored here
```

The Mikado approach to removing singletons

Compile times



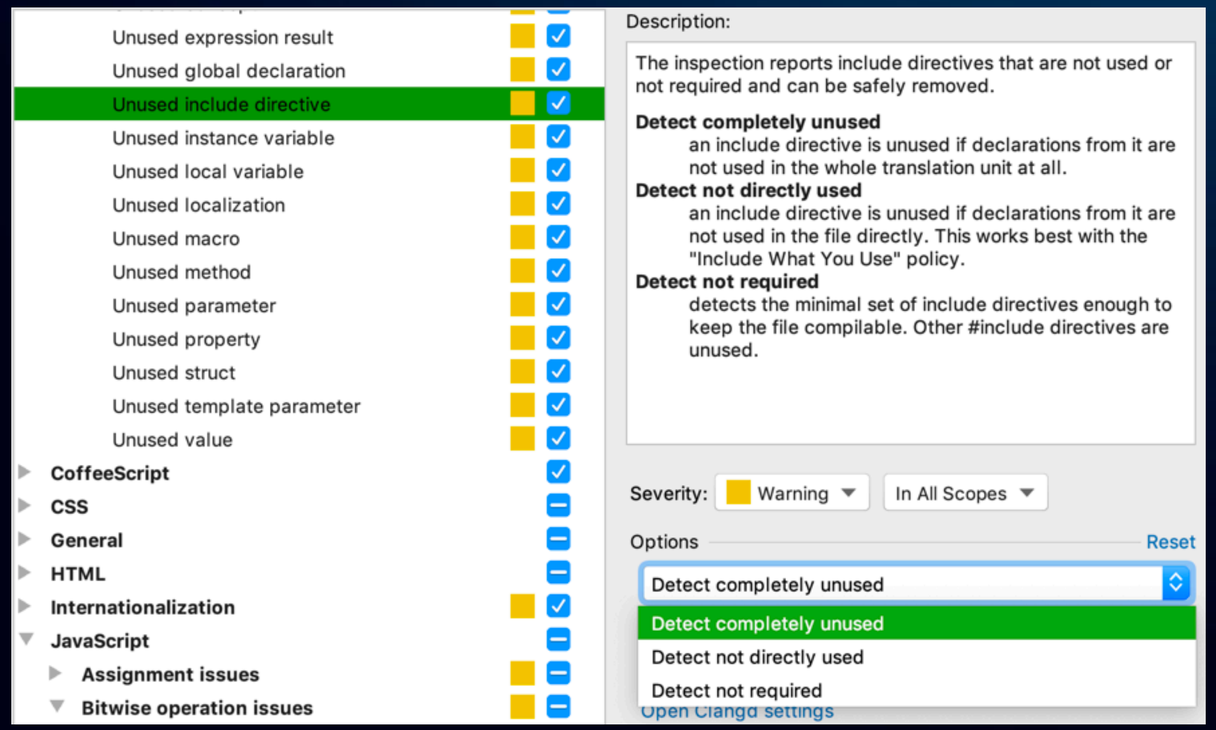
xkcd.com/303

`#define CATCH_CONFIG_MAIN` **in it's own file**

#include hygiene

Include-What-You-Use

CLion's "Unused include directive" inspection



`#define CATCH_CONFIG_MAIN` **in it's own file**

#include hygiene

- Include-What-You-Use
- CLion's "Unused include directive" inspection
- Minimise dependencies
  - forward declarations
  - pointers/ "pImpl"s
- Modules (C++20)

Templates

- .inl files
- extern template
- prefer "Modern C++"
  - constexpr
  - variadic templates
  - concepts (C++20)