https://forms.gle/3tKHdwzyBnKHUAga9

A word: How do you feel today?

Don't say "tired" or "exhausted" as that always applies

A Tweet: Why test software?

Testing

CS130, Lecture 2

Notes on working with systems

- You will encounter a significant number of large software systems in the class
    - Docker, GCloud, cmake, git, Gerrit, C++, etc.
- The class is generally not about these specific systems -- lectures generally aim at principles, not specifics
- There is a wide variety of existing tutorial material about all of them
- You will be well served by spending time gaining expertise in these systems -- they are well built and worth knowing.
    - *But you will never learn all of even one of them.*
- Use their own error messages, search, StackOverflow, etc. to guide how you use them.
- As with writing code for the class: ideal is that you solve problems directly, autonomously.
    - But don't suffer alone -- ask for help if needed.

Goals of this lecture

- Understand **why** to unit test
- Understand **what** to unit test
- Understand **how** to unit test
- Be able to unit test your code
- Introduce other kinds of testing

The most important lessons of this course

- Use revision control.
- If you don't want it to break, it needs a test.

The Beyonce Rule

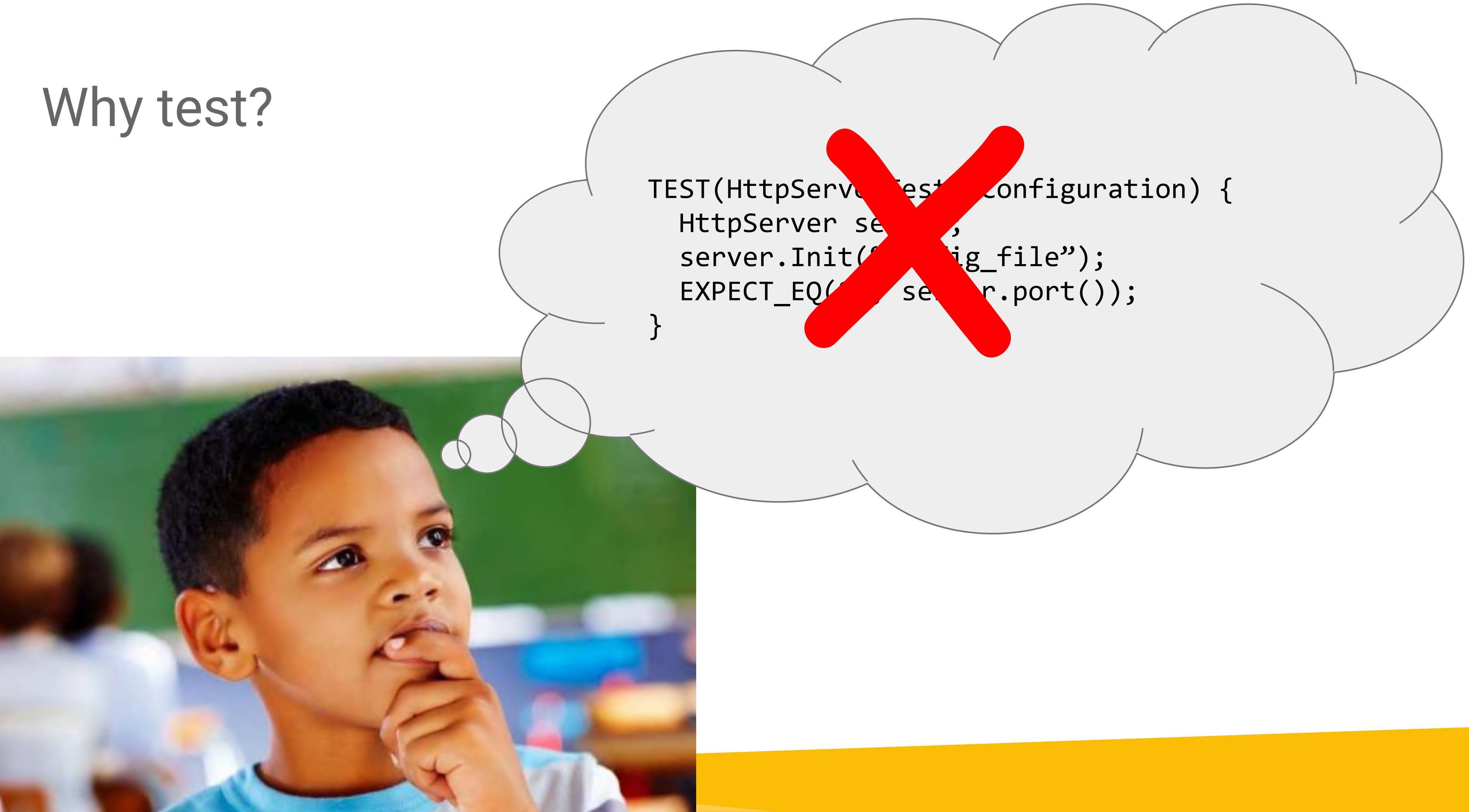
'Cause if you liked it, then you should have put a test on it

Why test?

Why test?

Why test?

Why test?



```
TEST(HttpServe  est  onfiguration) {
  HttpServer se    ;
  server.Init(    ig_file”);
  EXPECT_EQ(    se   r.port());
}
```

Why test? Mars Pathfinder

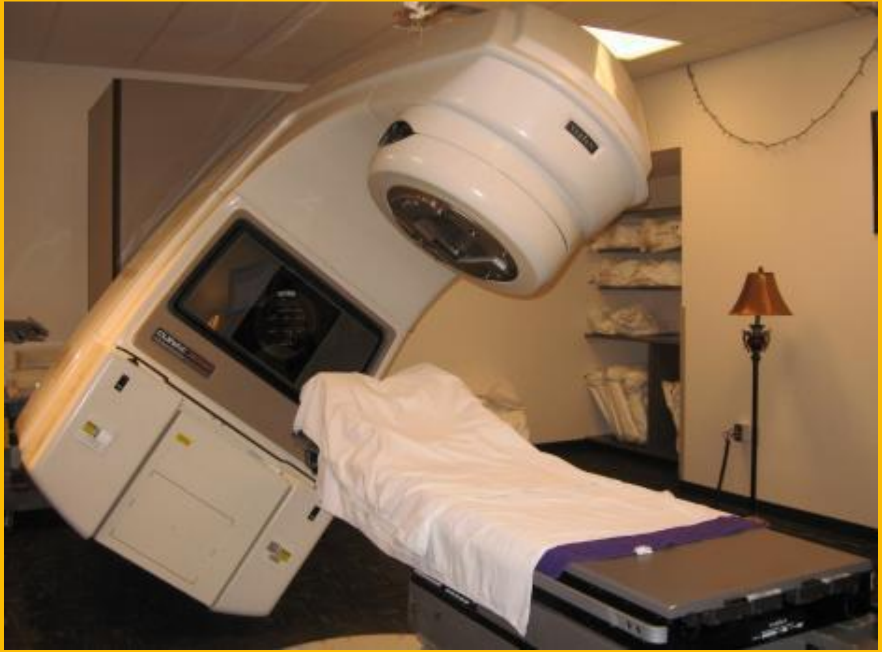
- $280 million (cheap for NASA)
- A few days after landing, the lander started freezing and restarting.

How do you diagnose and fix a problem when the hardware is on another planet?

Why test? Therac-25

- Would sometimes deliver 100x the intended dose of radiation
- 3 deaths

“The software should should be subjected to extensive testing and formal analysis at the module and software level; system testing alone is not adequate.”



Good tests are:

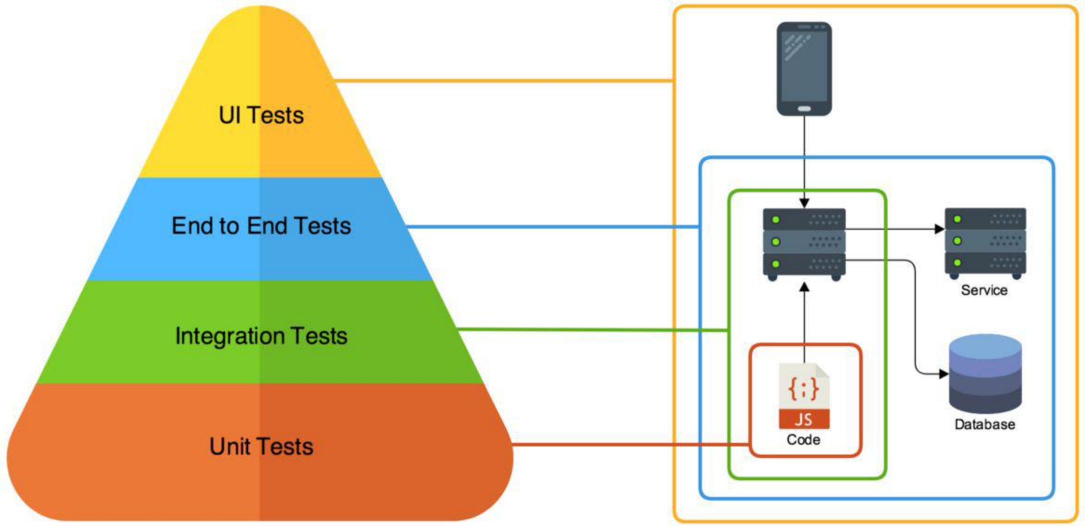
- Automated
- Repeatable
- Meaningful
- Reliable
- Easy to run
- Fast
- Checked into revision control

Types of tests

- Unit tests
- Integration tests
- Regression tests
- etc.

A well-tested system has:

- Many tests
- Different kinds of tests, providing overlapping coverage



Writing tests first?

- "Test-driven development"
- API prototyping
  - Write unit tests that are examples of using your API.
- When you know the desired result before you know how to implement it.
- Before fixing a bug, write a unit test that fails because of the bug.

When can I stop?!

- Signs you need more tests:
  - “Low-level”
  - “Mission-critical”
  - Lots of people depending on it (customers, coworkers, etc.)
  - Failure is expensive (Pathfinder)
  - Failure will result in death (Therac-25)

A real-world example

```
% wc -l pipeline.cc pipeline_test.cc
  1810 pipeline.cc
  9201 pipeline_test.cc
 11011 total
```

“Write more tests” isn’t always the answer

- Prototypes and rapidly changing code.
- One-off scripts, non-production code.
- Tests can be flaky.
- Sometimes there is no substitute for manual testing
  - User interfaces
  - Hardware (“dogfooding”)

“Write more tests” isn’t always the answer

Tests are expensive to write and maintain

- Time
- Money
- Opportunity

✘ “Should I write more tests?”

✔ “What are the right kinds of tests for this situation?”

General guidelines for this course

- Code should have unit tests, at least for basic use cases and common errors.
- Pretty much all classes and source files should have a unit test.
- Binaries should have an integration test.

2 UNIT TESTS, 0 INTEGRATION TESTS
via reddit.com/r/programmerhumor

Testing case study: your project

Config Parsing

- You need a way to configure the webserver you will write.
  - `% vi config`
  - `% ./webserver config`

Config Parsing: Why is this separate from my server?

- Separating binary and config is an important robustness surface
  - Let’s you roll back misconfigs easily
  - Allows for testing other parts of the system easily using testing configs that exercise them
- But this introduces a layer in the code that itself needs testing.

Why test? Config Parsing

- You need a way to configure the webserver you will write.
  - `% vi config`
  - `% ./webserver config`
- We need to **parse** the config file.
- In order to parse it, we need a **format** for the config file.

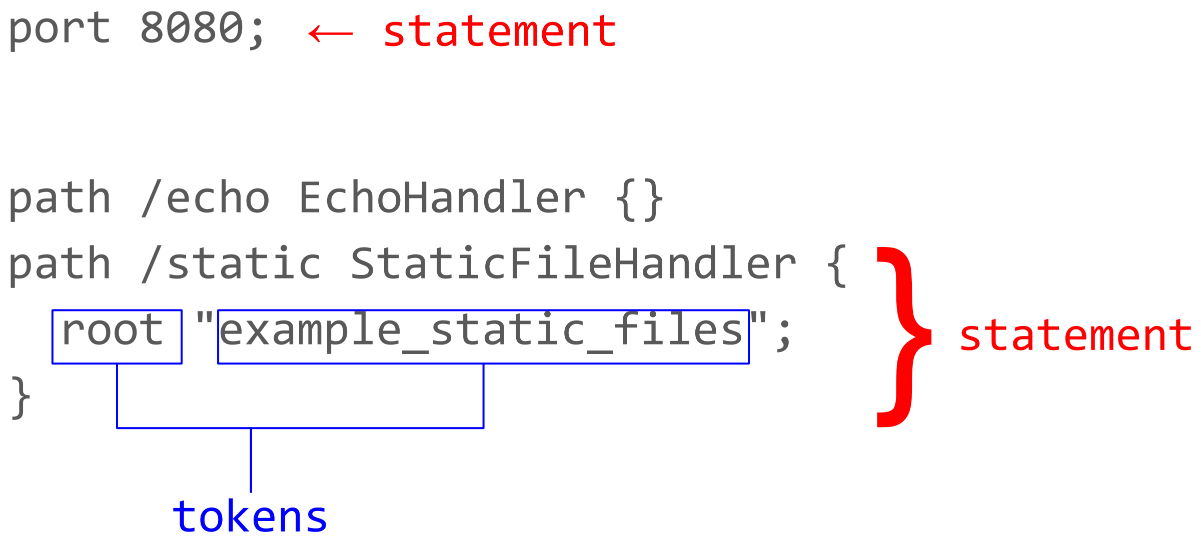
Config Parsing: File Format

- NginX config file format
- Example:

```
port 8080;
path /echo EchoHandler {}
path /static StaticFileHandler {
  root "example_static_files";
}
```

https://www.nginx.com/resources/wiki/start/topics/examples/full/

Config Parsing: File Format



```
port 8080;

path /echo EchoHandler {}
path /static StaticFileHandler {
  root "example_static_files";
}
```

Config Parsing: File Format

```
port 8080;
path /echo EchoHandler {}
path /static StaticFileHandler {
  root "example_static_files";
}
```

Grammar:

```
<config> ::= <statement>*
<statement> ::= <token>+ ";"
  | <token>+ "{" <config> "}"
```

Config Parsing: API

```
class NginxConfig {
  vector<std::shared_ptr<
    NginxConfigStatement>>
      statements_;
};
class NginxConfigStatement {
  vector<string> tokens_;
  std::unique_ptr<NginxConfig>
    child_block_;
};
```

Grammar:

```
<config> ::= <statement>*
<statement> ::= <token>+ ";"
  | <token>+ "{" <config> "}"
```

Config Parsing: API

```
class NginxConfigParser {
 public:
  // Takes an opened config file or file name (respectively) and stores
  // the parsed config in the provided NginxConfig out-param. Returns true
  // iff the input config file is valid.
  bool Parse(std::istream* config_file, NginxConfig* config) const;
  bool Parse(const char* file_name, NginxConfig* config) const;
};
```

What tests should we write for this parser?

Config Parsing: How It Works

- **Lex** the characters into tokens.
- **Parse** the stream of tokens.
- Each is a state machine.

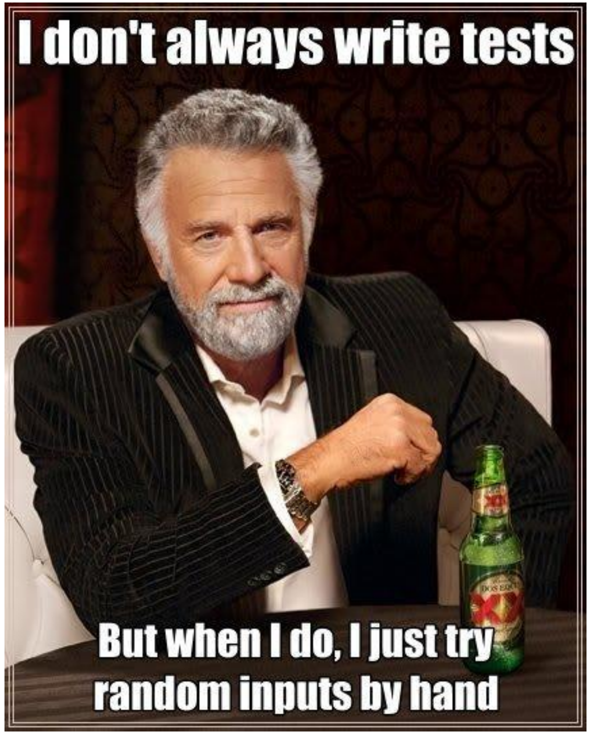
Ethics Survey

https://forms.gle/Rpmj1kr7zKdqHBVA7

What to test?

What to test?

“But I test my code!”


I don't always write tests
But when I do, I just try random inputs by hand

What to test? Apple SSL bug

```
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
```

What to test? Apple SSL bug

```
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
```

What to test? Lessons

- Testing could have found these problems
- But, the systems **did** have tests
  - Knowing **what to test** is important.

- Designing and building for testability is important.
  - (more on this in upcoming lectures)

What to test? How to come up with test cases

- The public API.
  - Typical use cases first.
- Error handling
- Anything particularly complex

What to test? Why you should test error handling

From "Simple Testing Can Prevent Most Critical Failures":

- Almost all (92%) of the catastrophic system failures are the result of incorrect handling of non-fatal errors explicitly signaled in software.
- In 58% of the catastrophic failures, the underlying faults could easily have been detected through simple testing of error handling code.

How to test?

- Unit tests
- Integration tests
- End-to-end tests

How to test?

- Unit tests ← Today
- Integration tests ← Later
- End-to-end tests ← Later

What’s missing from the config parser?

- A parseable config:

  ```
  asdf 1429801253 { sfdjklfsda;fsda asdf fdasjklfsda; }
  ```
- After parsing, you need logic to assign meaning to the config.
- How would you configure the port to listen on?
  - port { 82; }
  - 82 port;
  - __PORT__ “eighty-two”;

Unit tests

- Test units of code (classes, files, modules) in isolation.
- Written in the same language as the code they test.
- High bang-for-the-buck.
    - Fast to write + fast to execute

Some unexpected benefits:

- Serve as documentation, and demonstrates the API.
- You can make changes (refactor) quickly and confidently.
- Faster in the long run.

Finding good test cases

- Boundary conditions
  - int Add(int a, int b) → What if a, b are 0, 1, negative, really big, etc.
  - double Average(vector<double> a) → What if a is empty, 1 element, really big, etc.
- Pre and post-conditions
  - What should be true before and after a function, loop, or conditional?
- Be defensive
  - What happens if something that "can't happen" does?
- Error conditions
  - What happens in case of an error? (e.g. a file isn't found)
  - Do we crash, return early, throw an exception, etc.

Finding good test cases

- When in doubt…
  - Write your API docs on public methods
  - For each public method, write a test case for each statement of the docs

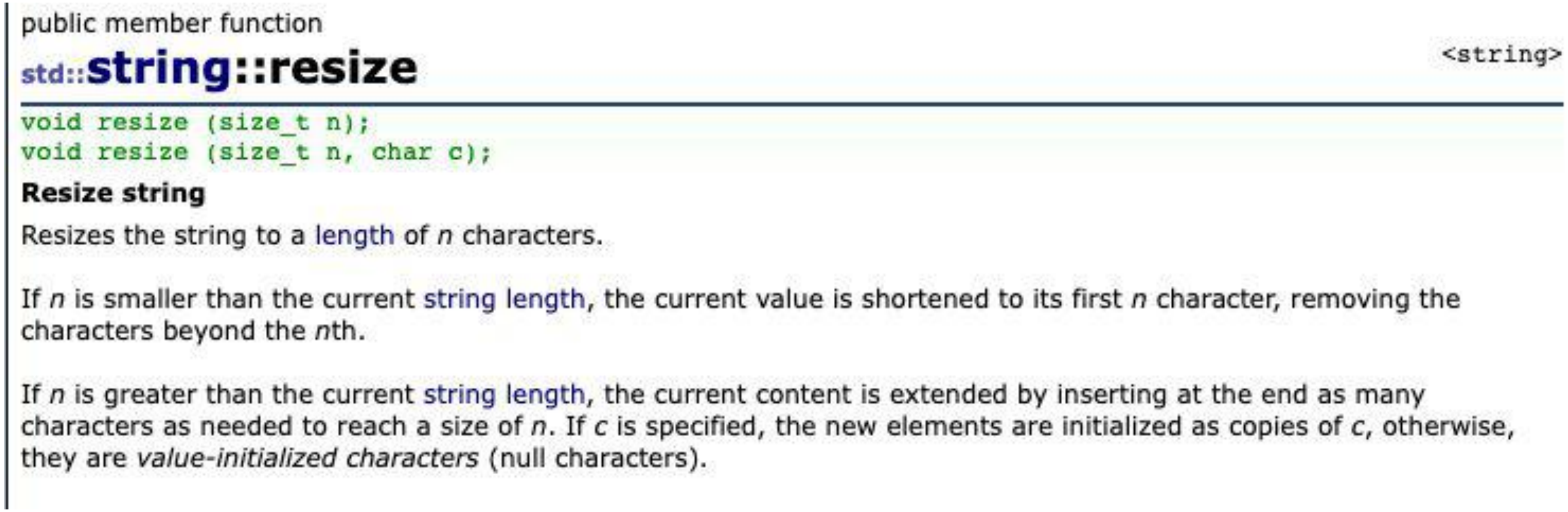
Finding good test cases, examples

```
class List

    void add(int index, E element)
```

Inserts the specified element at the specified position in this list (optional operation).

Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).

Finding good test cases, examples



public member function

std::string::resize

<string>

```
void resize (size_t n);
void resize (size_t n, char c);
```

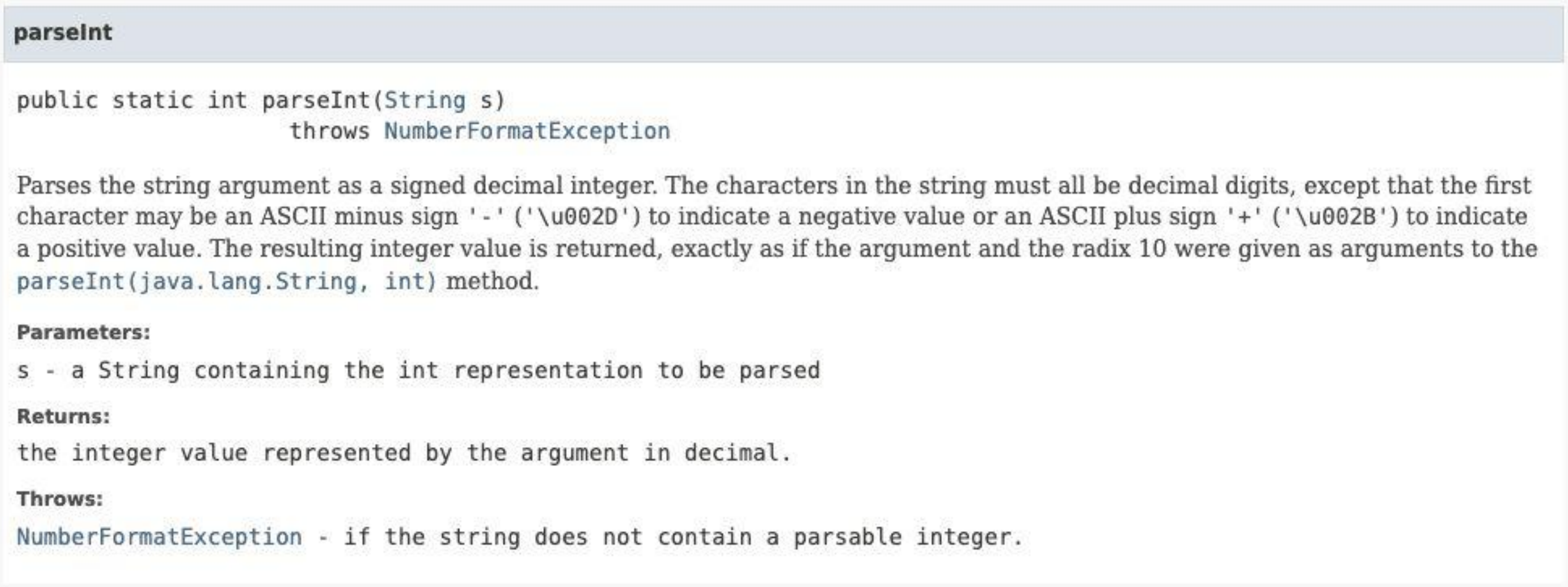
Resize string

Resizes the string to a length of *n* characters.

If *n* is smaller than the current string length, the current value is shortened to its first *n* character, removing the characters beyond the *n*th.

If *n* is greater than the current string length, the current content is extended by inserting at the end as many characters as needed to reach a size of *n*. If *c* is specified, the new elements are initialized as copies of *c*, otherwise, they are *value-initialized characters* (null characters).

Finding good test cases, examples



parseInt

```
public static int parseInt(String s)
                    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign `'-'` (`'\u002D'`) to indicate a negative value or an ASCII plus sign `'+'` (`'\u002B'`) to indicate a positive value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` method.

Parameters:

`s - a String containing the int representation to be parsed`

Returns:

`the integer value represented by the argument in decimal.`

Throws:

`NumberFormatException - if the string does not contain a parsable integer.`

Things that are hard to unit test

- Global variables
- Long, complex methods
- Objects with state
- Tight coupling (interdependent classes)
- Bad abstractions

How to unit test?

How to unit test? GUnit

How to unit test? GUnit: Code Layout

foo.h

```
class Foo {
 public:
  int Bar(...);
};
```

foo.cc

```
int Foo::Bar(...) { ... }
```

foo_test.cc

Unit tests for the Foo class.

How to unit test? GUnit: Code Layout

foo_test.cc

```
TEST(TestCaseName, TestName) { ... }               // A test.
class FooTest : public ::testing::Test { … };  // A fixture.
TEST_F(FooTest, Bar) {                             // A test using the fixture.
  EXPECT_TRUE(...);
  ASSERT_TRUE(...)
}
```

foo_test.cc layout:

- Fixtures and helper code
- Basic tests first
- Then, other uses cases, error handling, weird edge cases, bugs, etc.

https://github.com/google/googletest/tree/master/googletest/docs

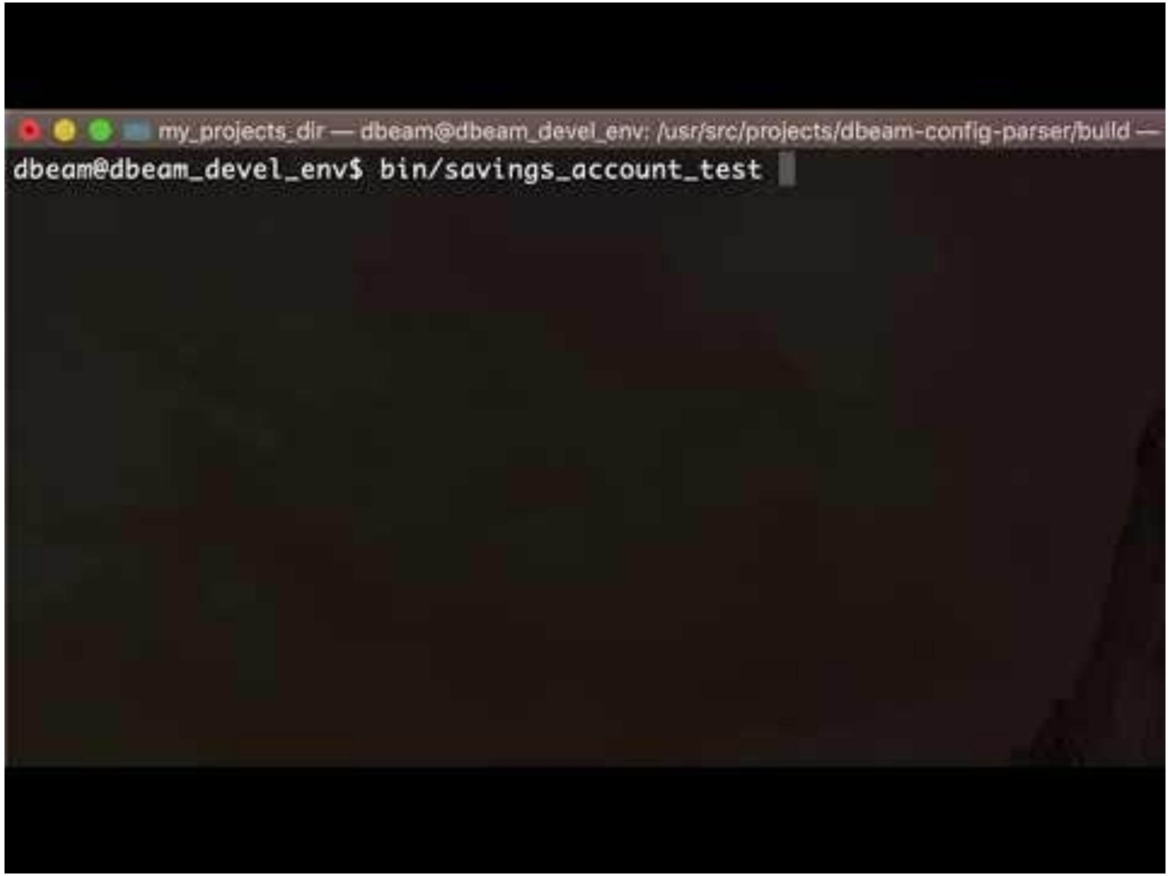
How to unit test? GUnit: Demo

```
class SavingsAccount {
 public:
  SavingsAccount() {}
  SavingsAccount(int balance) : balance_(balance) {}

  int GetBalance() const { return balance_; }
  void Deposit(int amount) { balance_ += amount; }
  void Withdraw(int amount) { balance_ -= amount; }

 private:
  int balance_;
};
```

How to unit test? GUnit: Demo

```
TEST(SavingsAccountTest, StartAndGetBalance) {
  const int kStartBalance = 50;
  SavingsAccount account(kStartBalance);
  EXPECT_EQ(account.GetBalance(), kStartBalance);
}

TEST(SavingsAccountTest, Deposit) {
  SavingsAccount account(0);
  const int kDepositAmount = 100;
  account.Deposit(kDepositAmount);
  EXPECT_EQ(account.GetBalance(), kDepositAmount);
}
```

```
TEST(SavingsAccountTest, Withdraw) {
  SavingsAccount account(500);
  account.Withdraw(50);
  EXPECT_EQ(account.GetBalance(), 450);
}
```

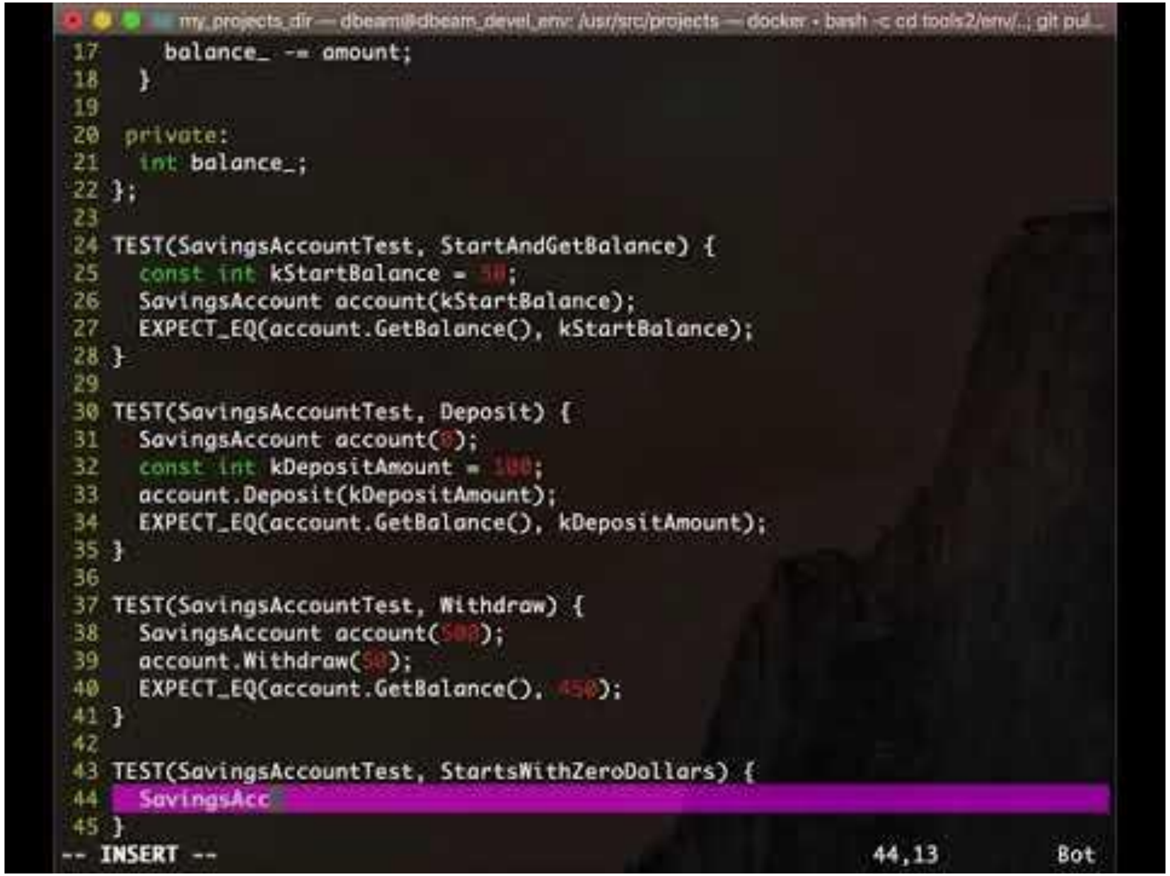


my_projects_dir — dbeam@dbeam_devel_env: /usr/src/projects/dbeam-config-parser/build —

```
dbeam@dbeam_devel_env$ bin/savings_account_test
```

UCLA CS 130 Software Engineering

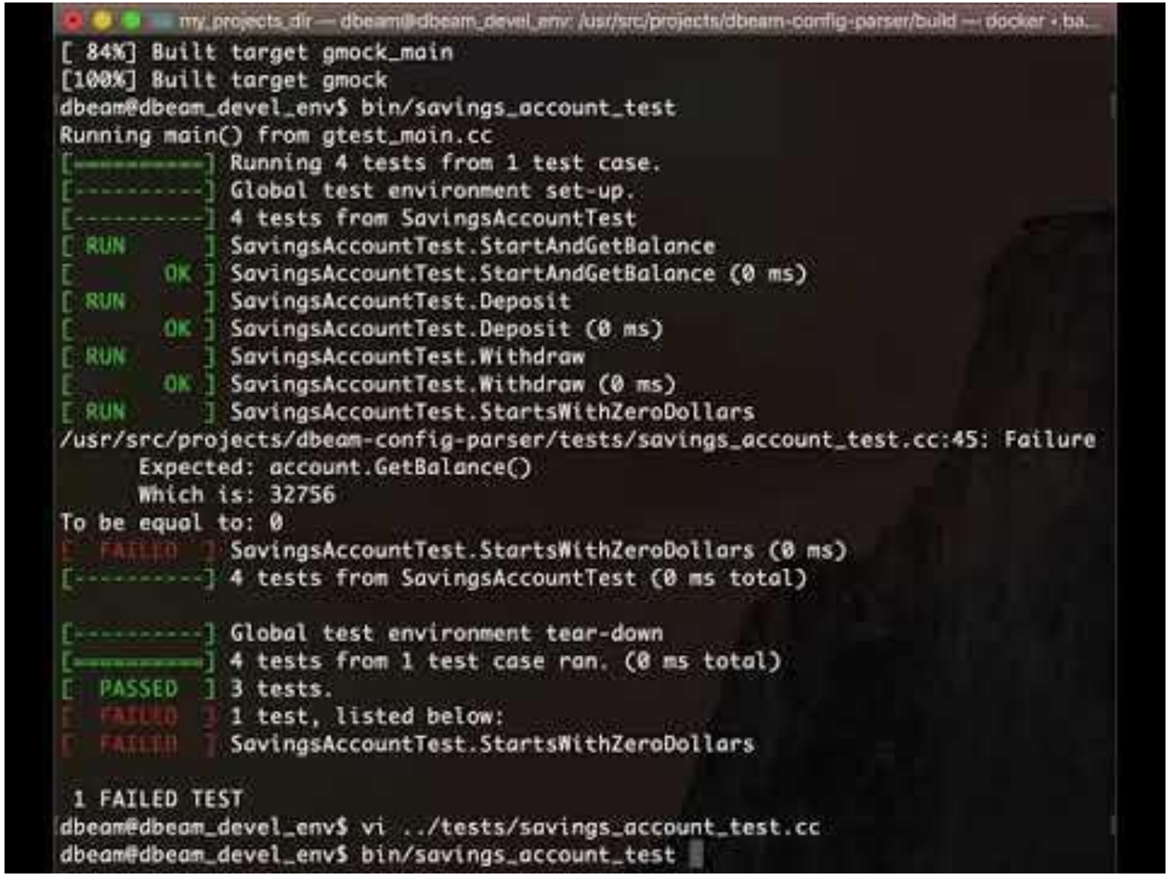
How to unit test? GUnit: Demo



my_projects_dir — dbeam@dbeam_devel_env: /usr/src/projects — docker • bash -c cd tools2/env/...; git pul...

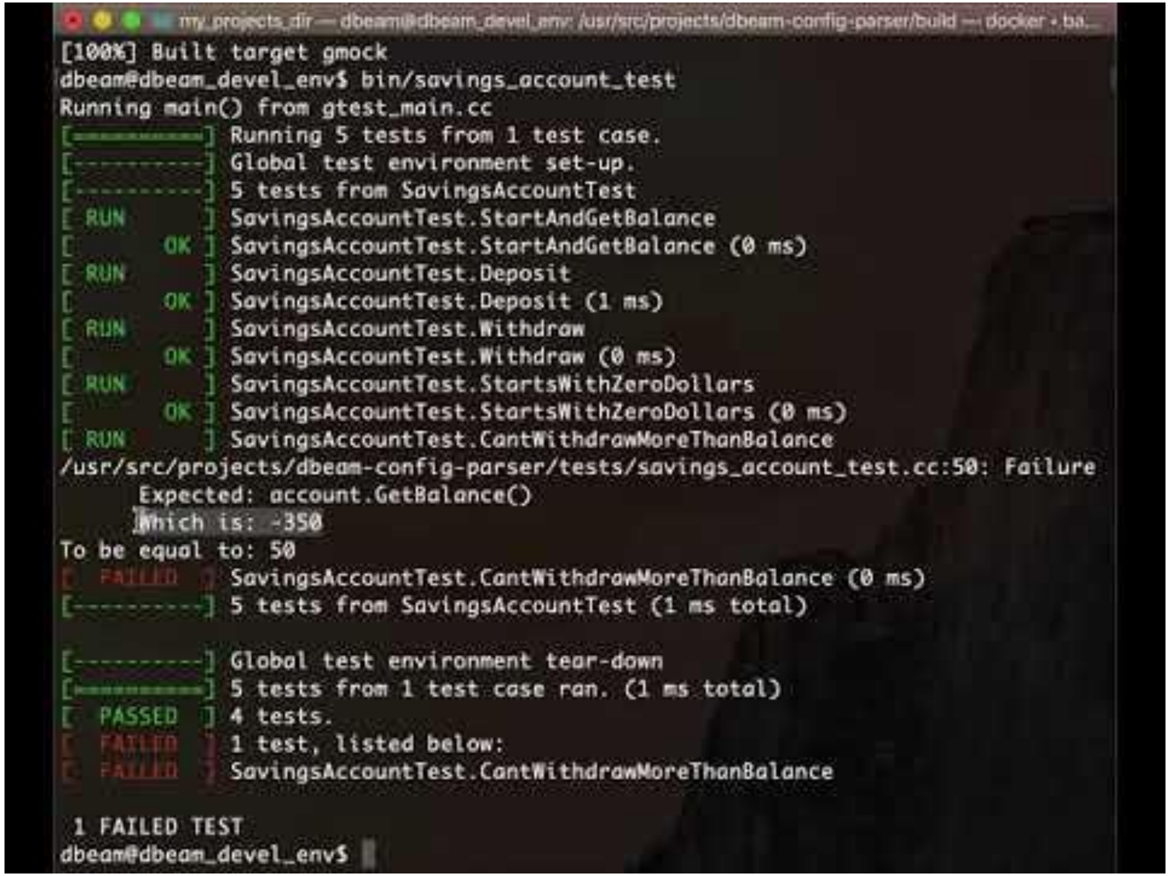
```
    balance_ -= amount;
  }

 private:
  int balance_;
};

TEST(SavingsAccountTest, StartAndGetBalance) {
  const int kStartBalance = 50;
  SavingsAccount account(kStartBalance);
  EXPECT_EQ(account.GetBalance(), kStartBalance);
}

TEST(SavingsAccountTest, Deposit) {
  SavingsAccount account(0);
  const int kDepositAmount = 100;
  account.Deposit(kDepositAmount);
  EXPECT_EQ(account.GetBalance(), kDepositAmount);
}

TEST(SavingsAccountTest, Withdraw) {
  SavingsAccount account(500);
  account.Withdraw(50);
  EXPECT_EQ(account.GetBalance(), 450);
}

TEST(SavingsAccountTest, StartsWithZeroDollars) {
  SavingsAcc
}
-- INSERT --                                                44,13         Bot
```

How to unit test? GUnit: Demo



```
my_projects_dir — dbeam@dbeam_devel_env: /usr/src/projects/dbeam-config-parser/build — docker • ba...
[ 84%] Built target gmock_main
[100%] Built target gmock
dbeam@dbeam_devel_env$ bin/savings_account_test
Running main() from gtest_main.cc
[==========] Running 4 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 4 tests from SavingsAccountTest
[ RUN      ] SavingsAccountTest.StartAndGetBalance
[       OK ] SavingsAccountTest.StartAndGetBalance (0 ms)
[ RUN      ] SavingsAccountTest.Deposit
[       OK ] SavingsAccountTest.Deposit (0 ms)
[ RUN      ] SavingsAccountTest.Withdraw
[       OK ] SavingsAccountTest.Withdraw (0 ms)
[ RUN      ] SavingsAccountTest.StartsWithZeroDollars
/usr/src/projects/dbeam-config-parser/tests/savings_account_test.cc:45: Failure
      Expected: account.GetBalance()
      Which is: 32756
To be equal to: 0
[  FAILED  ] SavingsAccountTest.StartsWithZeroDollars (0 ms)
[----------] 4 tests from SavingsAccountTest (0 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 3 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] SavingsAccountTest.StartsWithZeroDollars

 1 FAILED TEST
dbeam@dbeam_devel_env$ vi ../tests/savings_account_test.cc
dbeam@dbeam_devel_env$ bin/savings_account_test
```

How to unit test? GUnit: Demo



```
my_projects_dir — dbeam@dbeam_devel_env: /usr/src/projects/dbeam-config-parser/build — docker • ba...
[100%] Built target gmock
dbeam@dbeam_devel_env$ bin/savings_account_test
Running main() from gtest_main.cc
[==========] Running 5 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 5 tests from SavingsAccountTest
[ RUN      ] SavingsAccountTest.StartAndGetBalance
[       OK ] SavingsAccountTest.StartAndGetBalance (0 ms)
[ RUN      ] SavingsAccountTest.Deposit
[       OK ] SavingsAccountTest.Deposit (1 ms)
[ RUN      ] SavingsAccountTest.Withdraw
[       OK ] SavingsAccountTest.Withdraw (0 ms)
[ RUN      ] SavingsAccountTest.StartsWithZeroDollars
[       OK ] SavingsAccountTest.StartsWithZeroDollars (0 ms)
[ RUN      ] SavingsAccountTest.CantWithdrawMoreThanBalance
/usr/src/projects/dbeam-config-parser/tests/savings_account_test.cc:50: Failure
      Expected: account.GetBalance()
      Which is: -350
To be equal to: 50
[  FAILED  ] SavingsAccountTest.CantWithdrawMoreThanBalance (0 ms)
[----------] 5 tests from SavingsAccountTest (1 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test case ran. (1 ms total)
[  PASSED  ] 4 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] SavingsAccountTest.CantWithdrawMoreThanBalance

 1 FAILED TEST
dbeam@dbeam_devel_env$
```

How to unit test? GUnit: Demo

Apply the knowledge: How can we focus a test to only our class?

```
class HttpServer {
 public:
  // Read the config and initialize the server. Returns true if successful.
  bool Init(const string& config_file) {
    NginxConfig config;
    if (parser_.Parse(config_file, &config) {
      return false;  // Failed to parse the config. An unrecoverable error.
    }
    // Now, do stuff with the config.
  }
 private:
  NginxConfigParser parser_;
};
```

How to unit test? What we want to be able to test

- I expect `parser_.Parse()` to be called once.
- I expect it to be called with the following arguments…
- When it’s called, I expect the return value to be…
- When it’s called, I expect it to set the config pointer to…
    - (return via pointer)

How to unit test? Mocks: Defining

config_parser.h:

```
class ConfigParser {
  virtual bool Parse(const string& config_file, NginxConfig* config);
};
```

http_server_test.cc:

```
#include "gmock/gmock.h"
Class MockConfigParser : public ConfigParser {
   MOCK_METHOD2(Parse, bool(const string&, NginxConfig*));
}
```

https://github.com/google/googlemock/tree/master/googlemock/docs

How to unit test? Mocks: Injecting

Before:

```
class HttpServer {
 public:
  HttpServer() {}
 private:
  ConfigParser parser_;
};
```

After:

```
class HttpServer {
 public:
  HttpServer(NginxConfigParser* parser)
     : parser_(parser) {}
 private:
  ConfigParser* parser_;  // A pointer
};
```

How to unit test? Mocks: Writing expectations

```
TEST(HttpParserTest, Configuration) {
  MockConfigParser mock_parser;
  HttpServer server(&mock_parser);
  NginxConfig injected_config;
  // Set up the config we want to use.
  EXPECT_CALL(mock_parser, Parse("fake filename", _))
    .WillOnce(DoAll(
      SetArgPointee<1>(injected_config),
      Return(true)));
  EXPECT_TRUE(server.Init("fake filename"));
}
```

How to unit test? Mocks: Testing error handling

```
TEST(HttpParserTest, ErrorHandling) {
  MockConfigParser mock_parser;
  HttpServer server(&mock_parser);
  // Simulate a config parsing error.
  EXPECT_CALL(mock_parser, Parse(_, _)).WillOnce(Return(false));
  // Server initialization should fail.
  EXPECT_FALSE(server.Init("fake filename"));
}
```

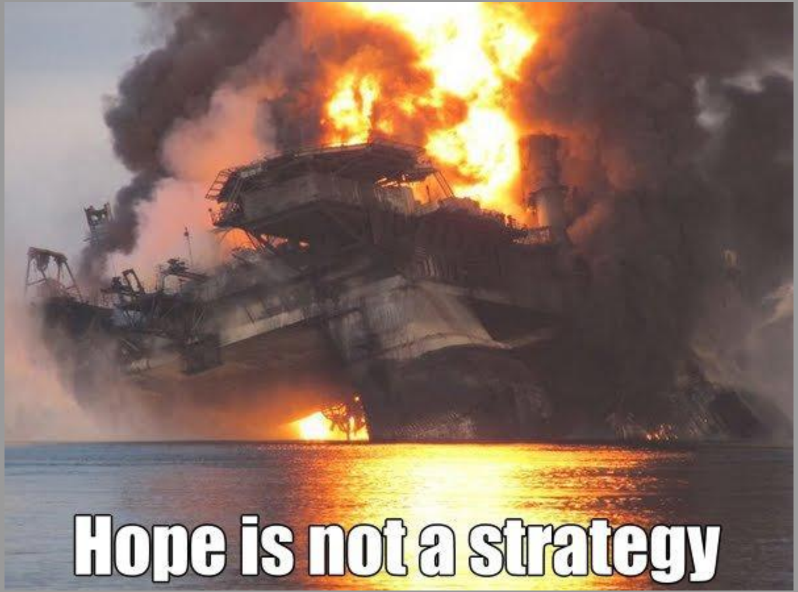
Course Expectations

Course expectations

- Your web server should have unit tests
    - Every class and source file should have unit tests.
    - Unit tests should at least for basic use cases and common error conditions.
    - Limited exceptions, like main(), which should be trivial.
    - If an assignment asks you to implement code, that mean unit tests too.
        - Unless the assignment says otherwise.
- Your web server should have an integration test
    - You'll write one in assignment 3
    - As you add features, update the integration test too.

// This should never happen

- It probably will at some point.
- You'll wish you had a test for it.



Coming up...

- Next Monday
  - Assignment #1 (testing) due Tuesday!
  - Lecture: Code Reviews, Tools for web server development
- Wednesday after that
  - Build systems and Deployment

https://bit.ly/3DoRYcr

- **A word:** How do you feel about software testing?
- **A tweet:** Describe the root cause of a future bug that kills your web server.