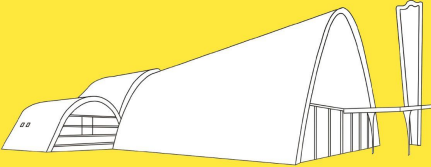


# SOFTWARE ENGINEERING

A Modern Approach



MARCO TULLIO VALENTE

## Chapter 8 - Testing

Prof. Marco Tulio Valente

<https://softengbook.org>

CC-BY: This license enables anyone to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the author.

```
public class Math {  
  
    public static long factorial(int n) {  
        if (n == 0 || n == 1) {  
            return 1;  
        } else {  
            long result = 1;  
            for (int i = 2; i <= n; i++) {  
                result *= i;  
            }  
            return result;  
        }  
    }  
}
```

```
public class Math {  
  
    public static long factorial(int n) {  
        if (n == 0 || n == 1) {  
            return 1;  
        } else {  
            long result = 1;  
            for (int i = 2; i <= n; i++) {  
                result *= i;  
            }  
            return result;  
        }  
    }  
}
```

What code is  
missing here?

```
public class Math {  
  
    public static long factorial(int n) {  
        if (n == 0 || n == 1) {  
            return 1;  
        } else {  
            long result = 1;  
            for (int i = 2; i <= n; i++) {  
                result *= i;  
            }  
            return result;  
        }  
    }  
}
```

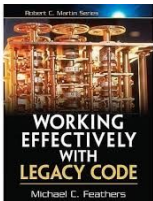
```
public class MathTest {  
  
    @Test  
    public void testFactorial() {  
        assertEquals(1, Math.factorial(0));  
        assertEquals(1, Math.factorial(1));  
        assertEquals(120, Math.factorial(5));  
    }  
}
```



Unit Testing is strongly encouraged and widely practiced at Google. All code used in production is expected to have unit tests.



At Facebook, engineers conduct any unit tests for their newly developed code.



Code without tests is bad code.

-- Michael Feathers

# Stack Overflow Developer Survey 2019

## Does Your Company Employ Unit Tests?

Yes, it's part of our process

**41.8%**



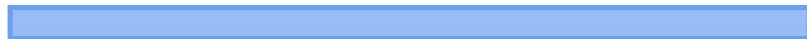
Yes, it's not part of our process but the developers do it on their own

**20.5%**



No, but I think we should

**33.2%**



No, and I'm glad we don't

**4.4%**



62,668 responses

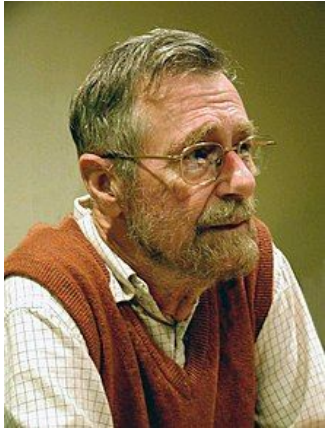
# Recalling Chapter 1 (Introduction)

# Software Testing

- Verify if a program produces an expected result when executed with some test cases
- Tests can be:
  - Manual
  - Automated



Software testing can reveal the presence of bugs, but not their absence.



Edsger W. Dijkstra

limitation

goal

# Defects, Bugs, and Failures

- Example of defect or bug:

```
if (condition)
    area = pi * radius * radius * radius;
```

- The correct formula is “area = pi \* radius \* radius”
- When it is executed, it will cause a failure, meaning an incorrect result.

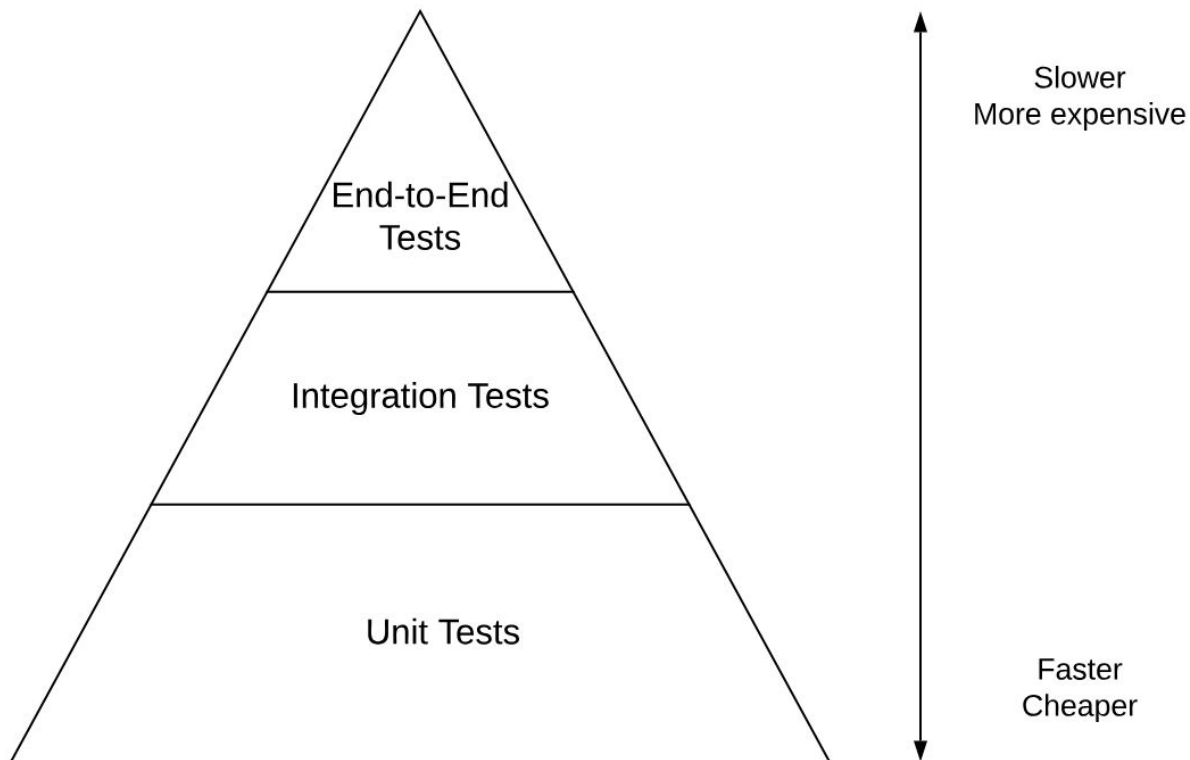
# Verification vs Validation

- **Verification:** Are we building the product right?
  - According to the defined specification
- **Validation:** Are we building the right product?
  - The one that meets the customer needs

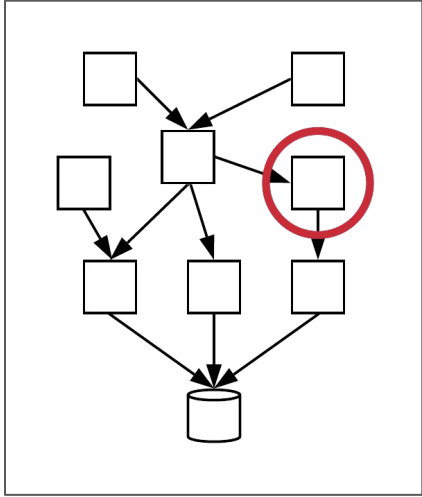
# Testing & Agile Methods

- Tests are automated
- Written by the developer of the code under testing

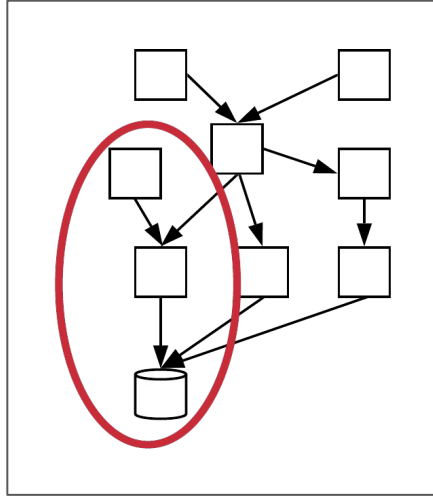
# Test Pyramid



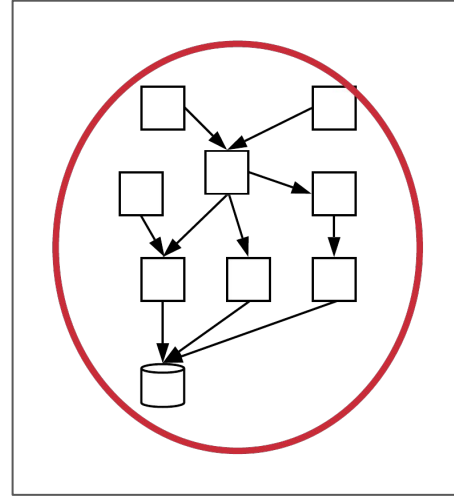
# Types of Automated Tests



Unit



Integration



End-to-End

# Unit Tests

(our main subject of study)

# Unit Tests

- Automated tests of small units of code (typically, classes)



# First Example: unit test for a Stack class

## Class Under Test

```
import java.util.ArrayList;
import java.util.EmptyStackException;

public class Stack<T> {

    private ArrayList<T> elements = new ArrayList<T>();
    private int size = 0;

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return (size == 0);
    }

    public void push(T elem) {
        elements.add(elem);
        size++;
    }

    public T pop() throws EmptyStackException {
        if (isEmpty())
            throw new EmptyStackException();
        T elem = elements.remove(size-1);
        size--;
        return elem;
    }
}
```

## Class Under Test

```
import java.util.ArrayList;
import java.util.EmptyStackException;

public class Stack<T> {

    private ArrayList<T> elements = new ArrayList<T>();
    private int size = 0;

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return (size == 0);
    }

    public void push(T elem) {
        elements.add(elem);
        size++;
    }

    public T pop() throws EmptyStackException {
        if (isEmpty())
            throw new EmptyStackException();
        T elem = elements.remove(size-1);
        size--;
        return elem;
    }
}
```

## Test (which is also a class)

```
import org.junit.Test;
import static org.junit.Assert.assertTrue;

public class StackTest {

    @Test
    public void testEmptyStack() {
        Stack<Integer> stack = new Stack<Integer>();
        boolean empty = stack.isEmpty();
        assertTrue(empty);
    }

}
```

# Anatomy of a Unit Test

```
import org.junit.Test;
import static org.junit.Assert.assertTrue;

public class StackTest {

    @Test
    public void testEmptyStack() {
        Stack<Integer> stack = new Stack<Integer>();
        boolean empty = stack.isEmpty();
        assertTrue(empty);
    }

}
```

Test methods (without parameters, usually start with test)

Fixture (context)

Calls the method under test

Assert command: checks if the result is as expected; if not, throws an exception

# AAA Pattern

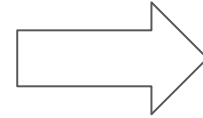
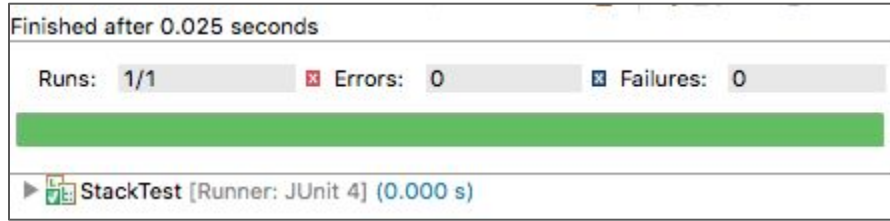
```
@Test
public void testEmptyStack() {
    Stack<Integer> stack = new
Stack<Integer>();
    boolean empty = stack.isEmpty();
    assertTrue(empty);
}
```

Arrange

Act

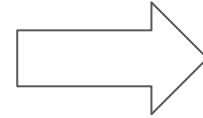
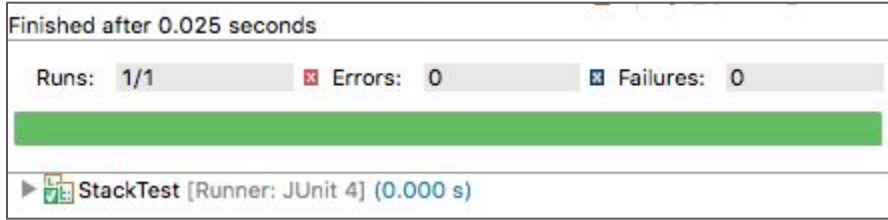
Assert

# Testing framework: xUnit

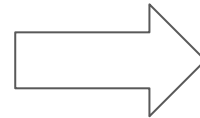
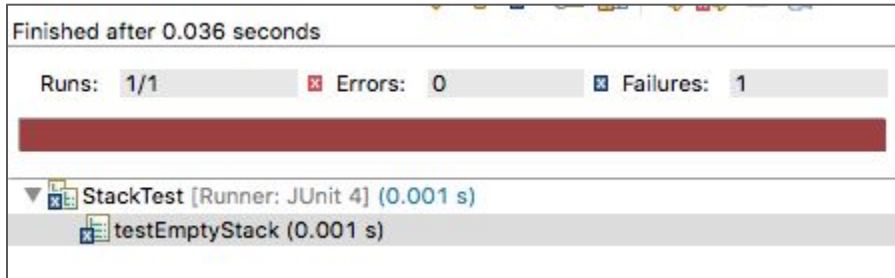


Tests passed!

# Testing framework: xUnit



Tests passed!




A test failed!

# More test methods



```
public class StackTest {  
  
    Stack<Integer> stack;  
  
    @Before  
    public void init() {  
        stack = new Stack<Integer>();  
    }  
  
    @Test  
    public void testEmptyStack() {  
        assertTrue(stack.isEmpty());  
    }  
  
    @Test  
    public void testNotEmptyStack() {  
        stack.push(10);  
        assertFalse(stack.isEmpty());  
    }  
}
```

Setup method executed before any @Test method



```
@Test
public void testSizeStack() {
    stack.push(10);
    stack.push(20);
    stack.push(30);
    int size = stack.size();
    assertEquals(3,size);
}
```


```
@Test
public void testPushPopStack() {
    stack.push(10);
    stack.push(20);
    stack.push(30);
    int result = stack.pop();
    result = stack.pop();
    assertEquals(20,result);
}
```

Expected value (3) and found value (size), in this order

Message when the assert fails:

Expected 3 but found [value]

```
@Test(expected = java.util.EmptyStackException.class)
public void testEmptyStackException() {
    stack.push(10);
    int result = stack.pop();
    result = stack.pop();
}
}
```



**assert** is not useful here; since it  
wouldn't be reached

# More concepts about testing

# Benefits

- Detecting bugs
  - In the class under test C
  - In other classes (regressions)
- Documentation

# FIRST Principles (good characteristics of unit tests)

- **F**ast
- **I**ndependent (execution order does not matter)
- **R**epeatable (deterministic, non-flaky, and non-erratic)
- **S**elf-checking (green vs red)
- **T**imely (written as soon as possible)

# Flaky Tests

- Non-deterministic tests: Sometimes they pass, sometimes they fail
- Example:

Results of successive executions of the same test T in a program that has not undergone any modifications:



# Why are some tests flaky?

Source: An Empirical Analysis of Flaky Tests, FSE 2014.

```
1 @Test
2 public void testRsReportsWrongServerName() throws Exception {
3     MiniHBaseCluster cluster = TEST_UTIL.getHBaseCluster();
4     MiniHBaseClusterRegionServer firstServer =
5         (MiniHBaseClusterRegionServer)cluster.getRegionServer(0);
6     HServerInfo hsi = firstServer.getServerInfo();
7     firstServer.setHServerInfo(...);
8
9     // Sleep while the region server pings back
10    Thread.sleep(2000);
11    assertTrue(firstServer.isOnline());
12    assertEquals(2, cluster.getLiveRegionServerThreads().size());
13    ... // similarly for secondServer
14 }
```

Concurrency  
(65% of cases)

And if the server takes more than 2 seconds to respond?



# Exercises

1. If end-to-end tests check "what's most important" (i.e., the entire system), why is it not recommended to implement only such tests?

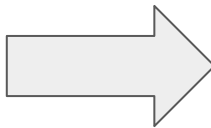

2. What will be printed by the following test methods?

```
class ExampleTest {  
    int i = 10;  
  
    @Test  
    public void test1() {  
        i++;  
        println(i);  
    }  
  
    @Test  
    public void test2() {  
        i++;  
        println(i);  
    }  
}
```

# Number of asserts per test


- Most of the time, use a single assert / test

```
@Test
public void testEmptyStack() {
    assertTrue(stack.isEmpty());
    stack.push(10);
    assertFalse(stack.isEmpty());
}
```



```
@Test
public void testEmptyStack() {
    assertTrue(stack.isEmpty());
}

@Test
public void testNotEmptyStack() {
    stack.push(10);
    assertFalse(stack.isEmpty());
}
```



## But there are exceptions ...

```
@Test
public void testRepeat() {
    String input = "20";
    assertEquals("", Strings.repeat(input,0));
    assertEquals("20", Strings.repeat(input,1));
    assertEquals("2020", Strings.repeat(input,2));
    assertEquals("202020", Strings.repeat(input,3));
    ...
}
```



How many tests do I have to write?

# Test Coverage

- Test coverage = (number of statements executed by the tests) / (total number of statements)

```
public class Stack<T> {  
    private ArrayList<T> elements = new ArrayList<T>();  
    private int size = 0;  
  
    public int size() {  
        return size;  
    }  
  
    public boolean isEmpty(){  
        return (size == 0);  
    }  
  
    public void push(T elem) {  
        elements.add(elem);  
        size++;  
    }  
  
    public T pop() throws EmptyStackException {  
        if (isEmpty())  
            throw new EmptyStackException();  
        T elem = elements.get(size-1);  
        size--;  
        return elem;  
    }  
}
```

100% coverage



```
public class Stack<T> {  
    private ArrayList<T> elements = new ArrayList<T>();  
    private int size = 0;  
  
    public int size() {  
        return size;  
    }  
  
    public boolean isEmpty(){  
        return (size == 0);  
    }  
  
    public void push(T elem) {  
        elements.add(elem);  
        size++;  
    }  
  
    public T pop() throws EmptyStackException {  
        if (isEmpty())  
            throw new EmptyStackException();  
        T elem = elements.get(size-1);  
        size--;  
        return elem;  
    }  
}
```

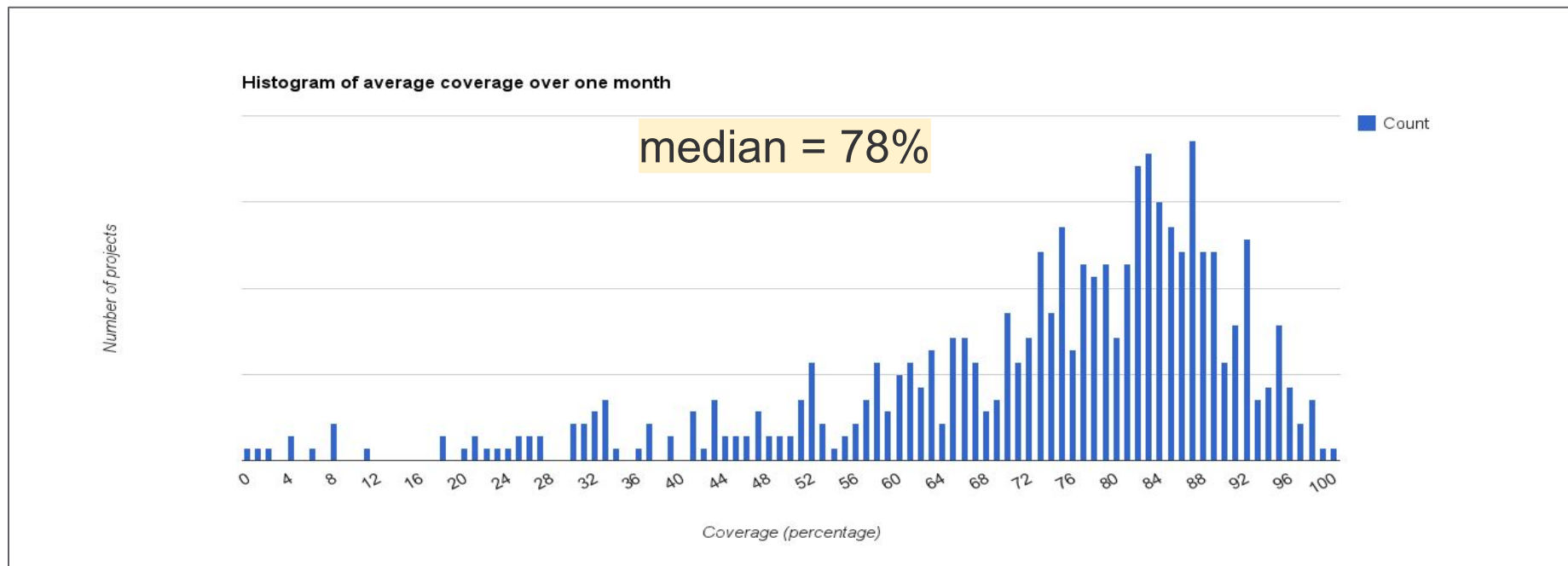
yellow: only one branch is tested;  
any "if" has two branches: T and F

red: command not covered by tests

# What is the ideal test coverage?

- Varies from project to project, but doesn't need to be 100%
- At least 60%, according to industry authors

# Example: Google



# Exercises

1. For the following function, fill in the table with the statements and branch coverage results.

```
void f(int x, int y) {  
    if (x > 0) {  
        x = 2 * x;  
        if (y > 0) {  
            y = 2 * y;  
        }  
    }  
}
```

Test	Stm	Brch
f(0,0)		
f(1,1)		
f(0,0) e f(1,1)		

2. In a university, students receive score A if they have a grade greater than or equal to 90. This function implements this requirement:

```
boolean isScoreA(int grade) {  
    if (grade > 90)  
        return true;  
    else return false;  
}
```

- (a) Does this implementation have a bug? If so, when does it result in a failure?
- (b) Suppose this function is tested with grades 85 and 95. What is the statement coverage of this test? And the branch coverage?

3. Consider the following statement:

if a program has 100% statement coverage, it is bug-free.

Is this statement true or false? Justify.

4. Why is it usually not necessary to achieve 100% statement coverage?

# Testability



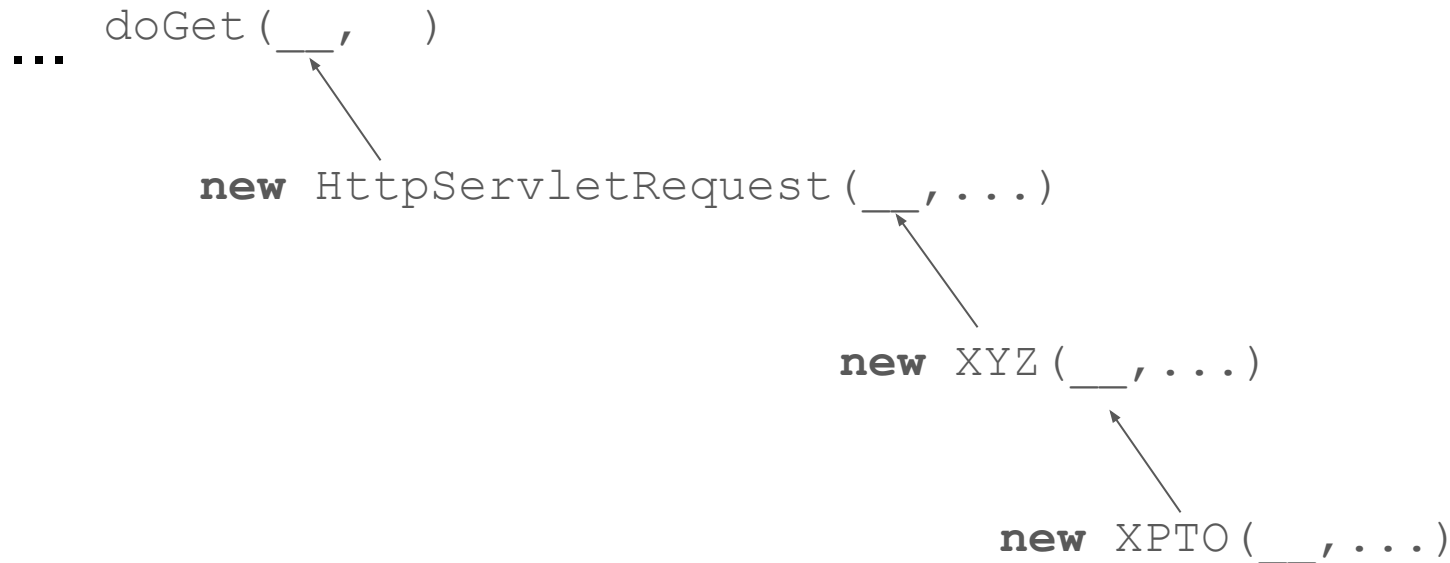
# Example: Servlet

```
public class BMIServlet extends HttpServlet {  
    public void doGet(HttpServletRequest req,  
                      HttpServletResponse res) {  
        res.setContentType("text/html");  
        PrintWriter out = res.getWriter();  
        String weight = req.getParameter("weight");  
        String height = req.getParameter("height");  
        try {  
            double w = Double.parseDouble(weight);  
            double h = Double.parseDouble(height);  
            double bmi = w / (h * h);  
            out.println("Body Mass Index (BMI): " + bmi);  
        }  
        catch (NumberFormatException e) {  
            out.println("Data must be numeric");  
        }  
    }  
}
```

Difficult to test because it has dependencies (parameters) on the Java Servlets package

# Problem: calling doGet(...) is not straightforward...

```
... doGet ( __ ,  )  
      ^  
      |  
new HttpServletRequest ( __ , ... )  
      ^  
      |  
new XYZ ( __ , ... )  
      ^  
      |  
new XPTO ( __ , ... )
```




Testability of doGet() is low

```
class BMIModel {  
    public double calculateBMI(String w1, String h1)  
        throws NumberFormatException {  
        double w = Double.parseDouble(w1);  
        double h = Double.parseDouble(h1);  
        return w / (h * h);  
    }  
}
```

```
public class BMIServlet extends HttpServlet {  
    BMIModel model = new BMIModel();  
  
    public void doGet(HttpServletRequest req,  
        HttpServletResponse res) {  
        res.setContentType("text/html");  
        PrintWriter out = res.getWriter();  
        String weight = req.getParameter("weight");  
        String height = req.getParameter("height");  
        try {  
            double bmi = model.calculateBMI(weight, height);  
            out.println("Body Mass Index (BMI): " + bmi);  
        }  
        catch (NumberFormatException e) {  
            out.println("Data must be numeric");  
        }  
    }  
}
```

Solution: Extract the domain rule to  
a separate, more easily testable  
class



# Mocks

# Motivating Example

```
import org.json.JSONObject;

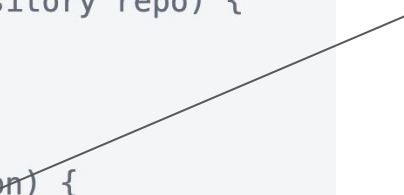
public interface BookRepository {
    String search(int isbn);
}

public class BookSearch {
    private BookRepository repo;

    public BookSearch(BookRepository repo) {
        this.repo = repo;
    }

    public Book getBook(int isbn) {
        String json = repo.search(isbn);
        JSONObject obj = new JSONObject(json);
        String title = (String) obj.get("title");
        return new Book(title);
    }
}
```

Method that searches for a book in a remote service

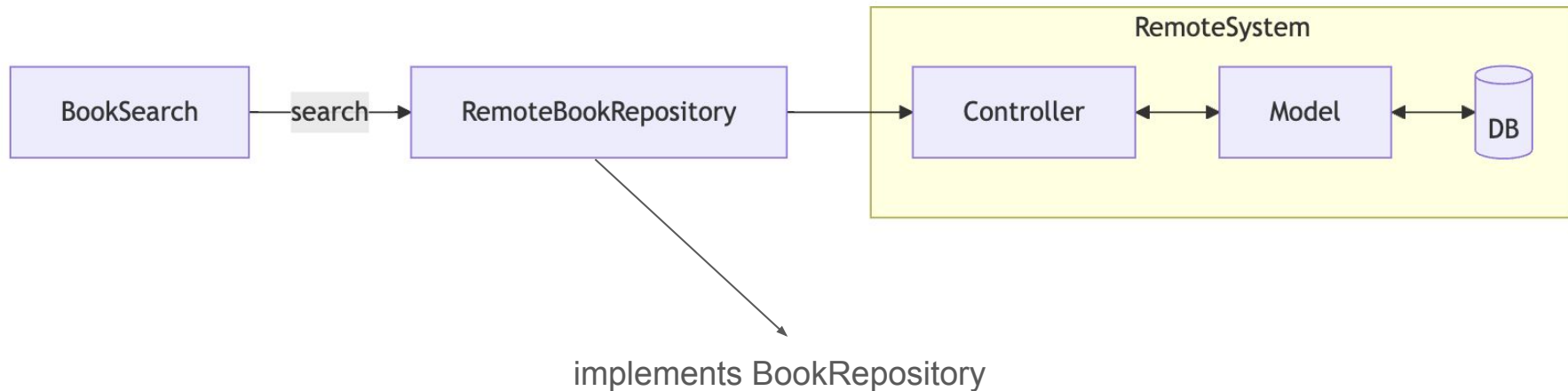


Problem: Unit tests should be fast!

## Solution: Mocks

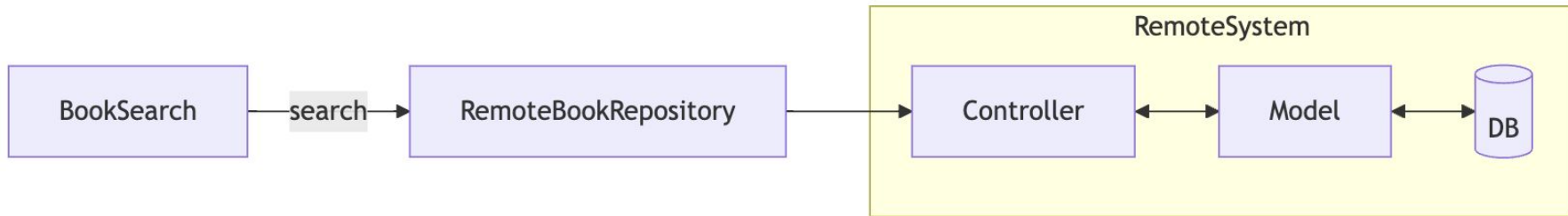
- An object that emulates a real object, but is much simpler than the original

## Solution without mocks (slow test as it accesses a remote server)





## Solution without mocks (slow test as it accesses a remote server)



## Solution with a mock



Only works with two books:  
SOFTENG and NULLBOOK

```
class BookConst {  
    public static String SOFTENG =  
        "{ \"title\": \"Software Engineering\" }";  
    public static String NULLBOOK =  
        "{ \"title\": \"NULL\" }";  
}
```

```
class MockBookRepository implements BookRepository {  
    public String search(int isbn) {  
        if (isbn == 1234)  
            return BookConst.SOFTENG;  
        return BookConst.NULLBOOK;  
    }  
}
```

Search for a single ISBN



```
public class BookSearchTest {
```

```
    private BookSearch bs;
```

```
    @BeforeEach
```

```
    public void init() {
```

```
        bs = new BookSearch(new MockBookRepository());  
    }
```

the test uses the mock



```
    @Test
```

```
    public void testGetBook() {
```

```
        String title = bs.getBook(1234).getTitle();  
        assertEquals("Software Engineering", title);  
    }
```

```
}
```

# Mock Frameworks



## Example: Mockito

- Facilitates the implementation of mocks via a domain-specific language
- Eliminates the need for manual mock implementation

```
public class BookSearchTest {
```

```
    private BookSearch bs;
```

```
    @BeforeEach
```

```
    public void init() {
```

```
        BookRepository mockRepo = mock(BookRepository.class);
```

```
        when(mockRepo.search(anyInt()))
```

```
            .thenReturn(BookConst.NULLBOOK);
```

```
        when(mockRepo.search(1234))
```

```
            .thenReturn(BookConst.SOFTENG);
```

```
        bs = new BookSearch(mockRepo);
```

```
    }
```

```
    @Test
```

```
    public void testGetBook() {
```

```
        String title = bs.getBook(1234).getTitle();
```

```
        assertEquals("Software Engineering", title);
```

```
    }
```

```
}
```

Creates a mock

Programs the behavior of the mock

## Manual Mock

```
@BeforeEach
public void init() {
    bs = new BookSearch(
        new
MockBookRepository());
}
```

```
class MockBookRepository
    implements BookRepository {

    public String search(int isbn) {
        if (isbn == 1234)
            return BookConst.SOFTENG;
        return BookConst.NULLBOOK;
    }
}
```

---

## Mockito

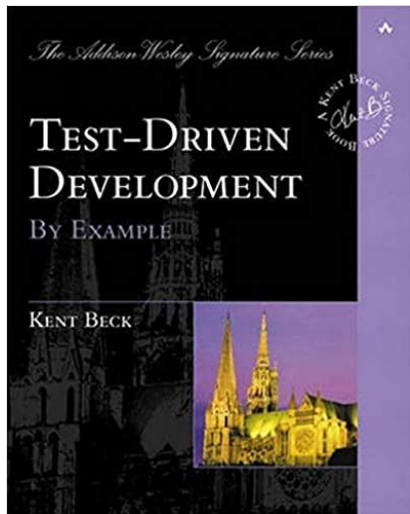
```
@BeforeEach
public void init() {
    BookRepository mockRepo = mock(BookRepository.class);
    when(mockRepo.search(anyInt())) .thenReturn(BookConst.NULLBOOK);
    when(mockRepo.search(1234)) .thenReturn(BookConst.SOFTENG);
    bs = new BookSearch(mockRepo);
}
```

# Test-Driven Development (TDD)



# TDD

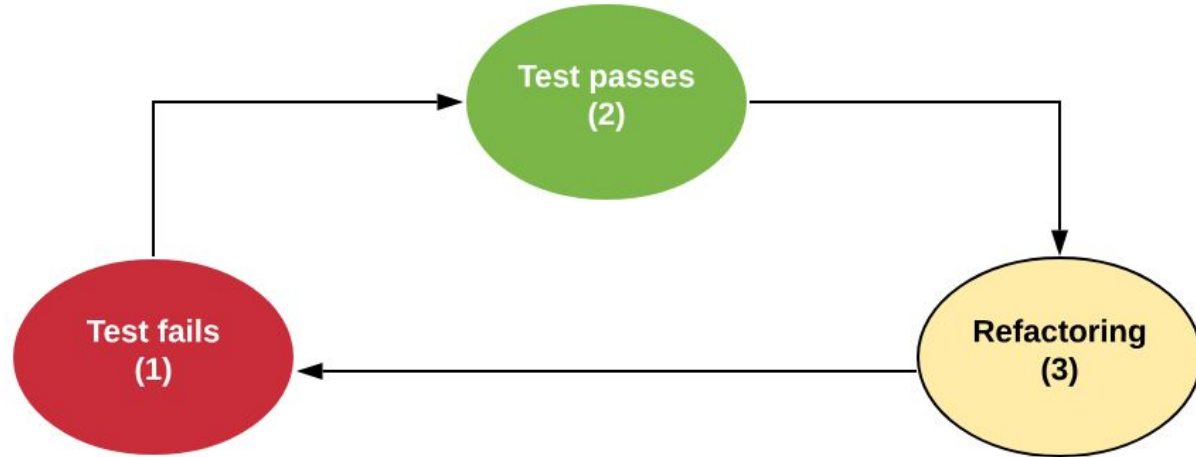
- One of the programming practices proposed by XP
- The idea is to write test T before class C



# Benefits

- Prevents developers from forgetting to write tests
- Encourages writing code with testability in mind
- Improves the design of the code, as the developer becomes the first user of their own code
- Coverage can reach up to 90%

# TDD cycle



# Example of TDD: Shopping Cart

# Red

```
@Test
void testAddGetTotal() {
    Book b1 = new Book("book1", 10, "1");
    Book b2 = new Book("book2", 20, "2");
    ShoppingCart cart = new ShoppingCart();
    cart.add(b1);
    cart.add(b2);
    assertEquals(30.0, cart.getTotal());
}
```

# Still **red**, but at least compiling

```
public class Book {  
    public String title;  
    public double price;  
    public String isbn;  
  
    public Book(String title, double price, String isbn) {  
        this.title = title;  
        this.price = price;  
        this.isbn = isbn;  
    }  
}  
  
public class ShoppingCart {  
  
    public ShoppingCart() {}  
  
    public void add(Book b) {}  
  
    double getTotal() {  
        return 0.0;  
    }  
}
```

Temporary implementations



# First Green

```
public class ShoppingCart {  
    public ShoppingCart() {}  
    public void add(Book b) {}  
    double getTotal() {  
        return 30.0;  
    }  
}
```

Just for a "small victory" ...  
baby steps

```
public class ShoppingCart {  
  
    private ArrayList<Book> items;  
  
    private double total;  
  
    public ShoppingCart() {  
        items = new ArrayList<Book>();  
        total = 0.0;  
    }  
  
    public void add(Book b) {  
        items.add(b);  
        total += b.price;  
    }  
  
    double getTotal() {  
        return total;  
    }  
  
}
```

## Now, a real **green**



**Yellow:**

Can we refactor and improve the code?

```
public class Book {  
    private String title;  
    private double price;  
    private String isbn;  
  
    public Book(String title, double price, String isbn)  
    {  
        this.title = title;  
        this.price = price;  
        this.isbn = isbn;  
    }  
}
```

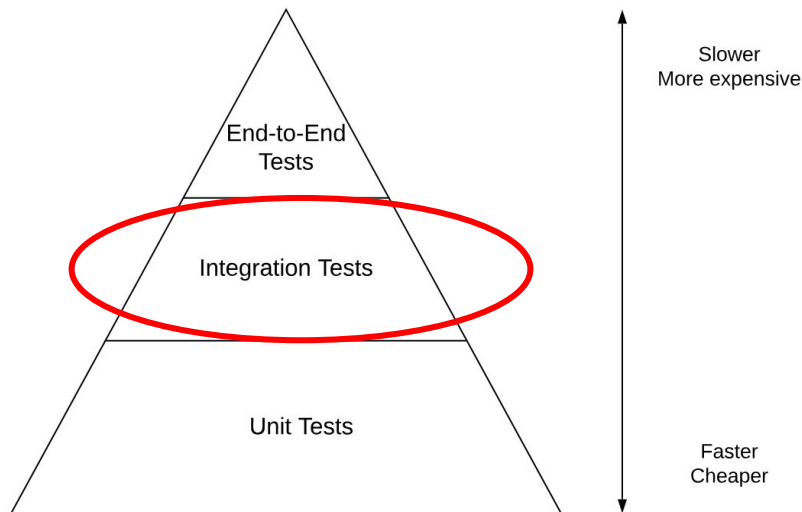
# Next Step

- Do we need more features?
- If yes, new TDD cycle (red-green-yellow)

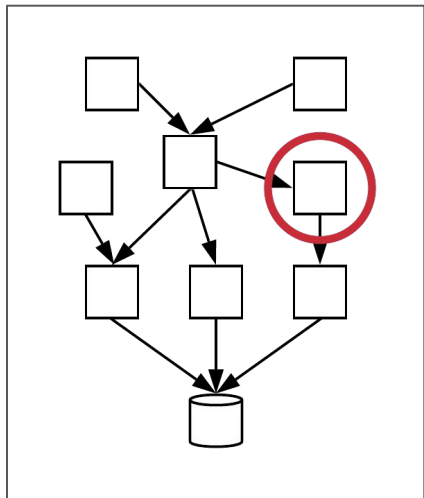
# Integration Tests

# Integration Tests

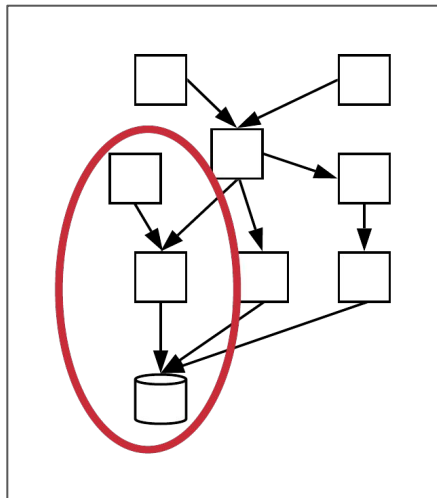
- Test a feature or service
- Including external services (e.g., database)



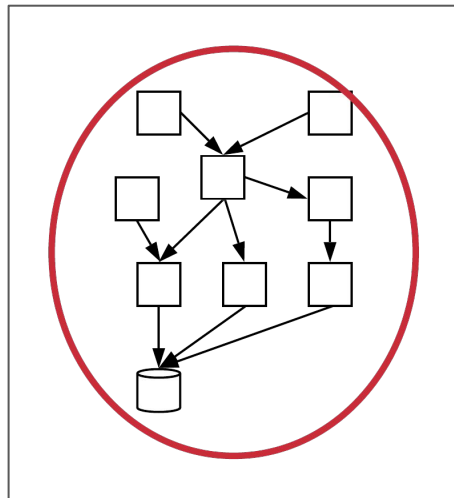
# Reminder...



Unit



Integration



End-to-End

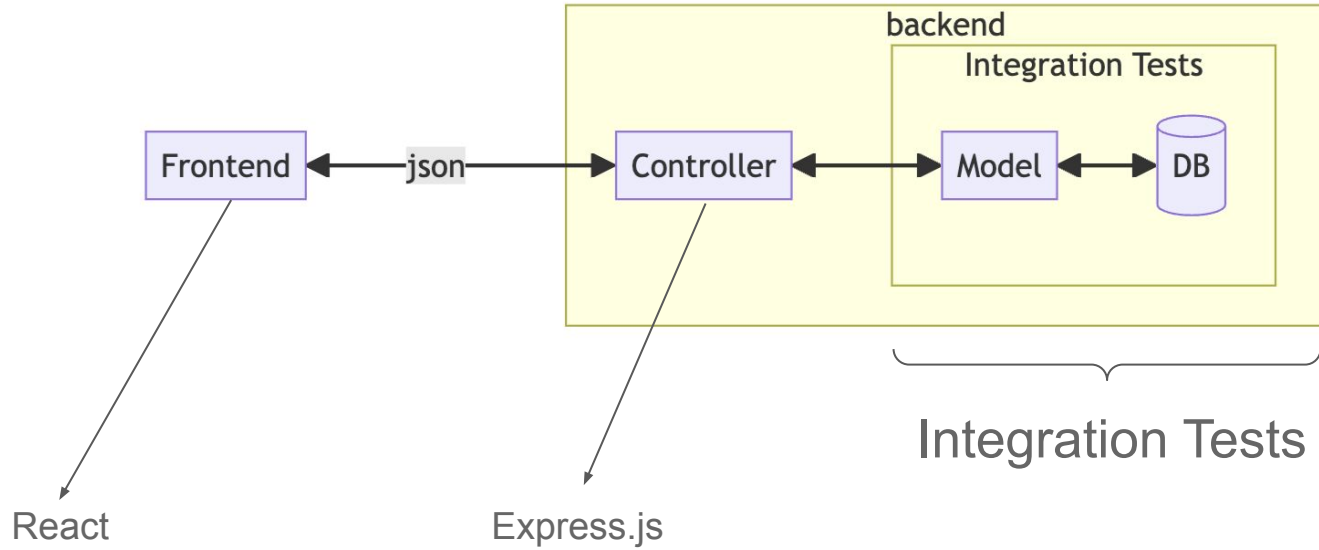
# Example of an Integration Test

Q&A Forum with a frontend (React) and a  
backend (Express.js)

## Questions

ID	Question	# Answers
1	How do you declare a variable in JavaScript?	<a href="#">2</a>
2	What is the difference between let and var in JavaScript?	<a href="#">1</a>
3	How do you create a function in JavaScript?	<a href="#">1</a>
4	What is the purpose of the this keyword in JavaScript?	<a href="#">0</a>
5	How do you add an event listener to a button in JavaScript?	<a href="#">0</a>

Make your question



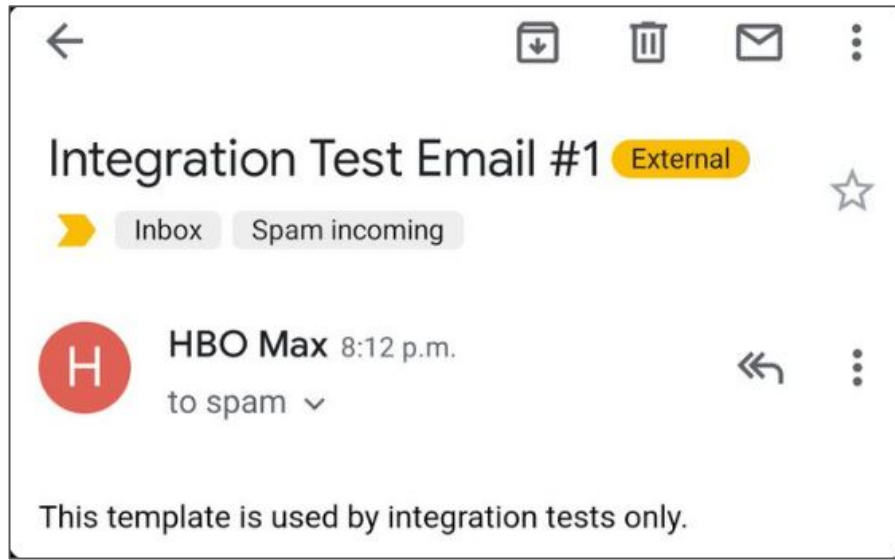


```
beforeEach(() => {  
  bd.reconfig('./db/qa-forum-test.db');  
  
  // clears all tables  
  bd.exec('delete from questions', []);  
  bd.exec('delete from answers', []);  
});  
  
test('Testing empty database', () => {  
  expect(model.list_questions().length).toBe(0);  
});
```

```
test('Creating three questions', () => {  
  model.add_question('1 + 1 = ?');  
  model.add_question('2 + 2 = ?');  
  model.add_question('3 + 3 = ?');  
  const questions = model.list_questions();  
  
  expect(questions.length).toBe(3);  
  expect(questions[0].text).toBe('1 + 1 = ?');  
  expect(questions[1].text).toBe('2 + 2 = ?');  
  expect(questions[2].num_answers).toBe(0);  
  expect(questions[1].id_question).toBe(questions[2].id_question-1);  
});
```

# Exercises

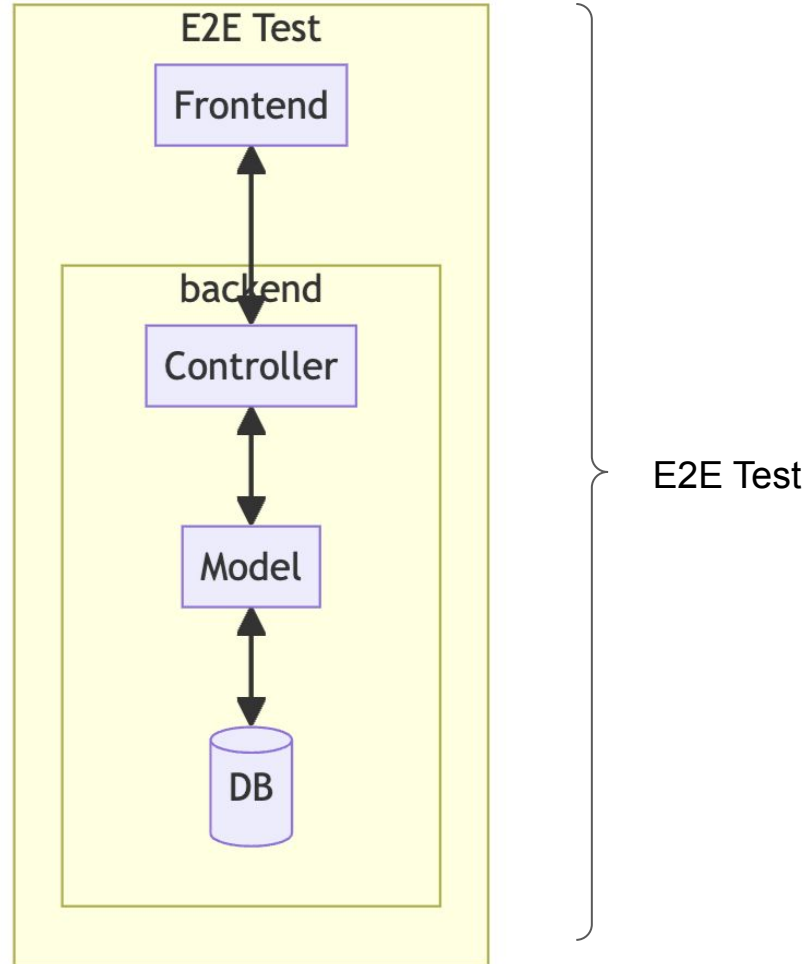
1. In June 2021, the following email was mistakenly sent to thousands of HBO subscribers. What might have caused this email to reach HBO's end users?



# End-to-End Tests

# End-to-End Tests

- Test the entire system via its external interface
- The test simulates a person interacting with the system (filling in data, clicking on buttons, etc.)
- Also known as: system tests, frontend tests, web UI tests



# Example of E2E Test



# todos

▼

What needs to be done?

☐

E2E Testing Practical Assignment

☒

~~Operating Systems Exam~~

×

☐

Compilers Practical Assignment

☐

Calculus III Exam

3 items left

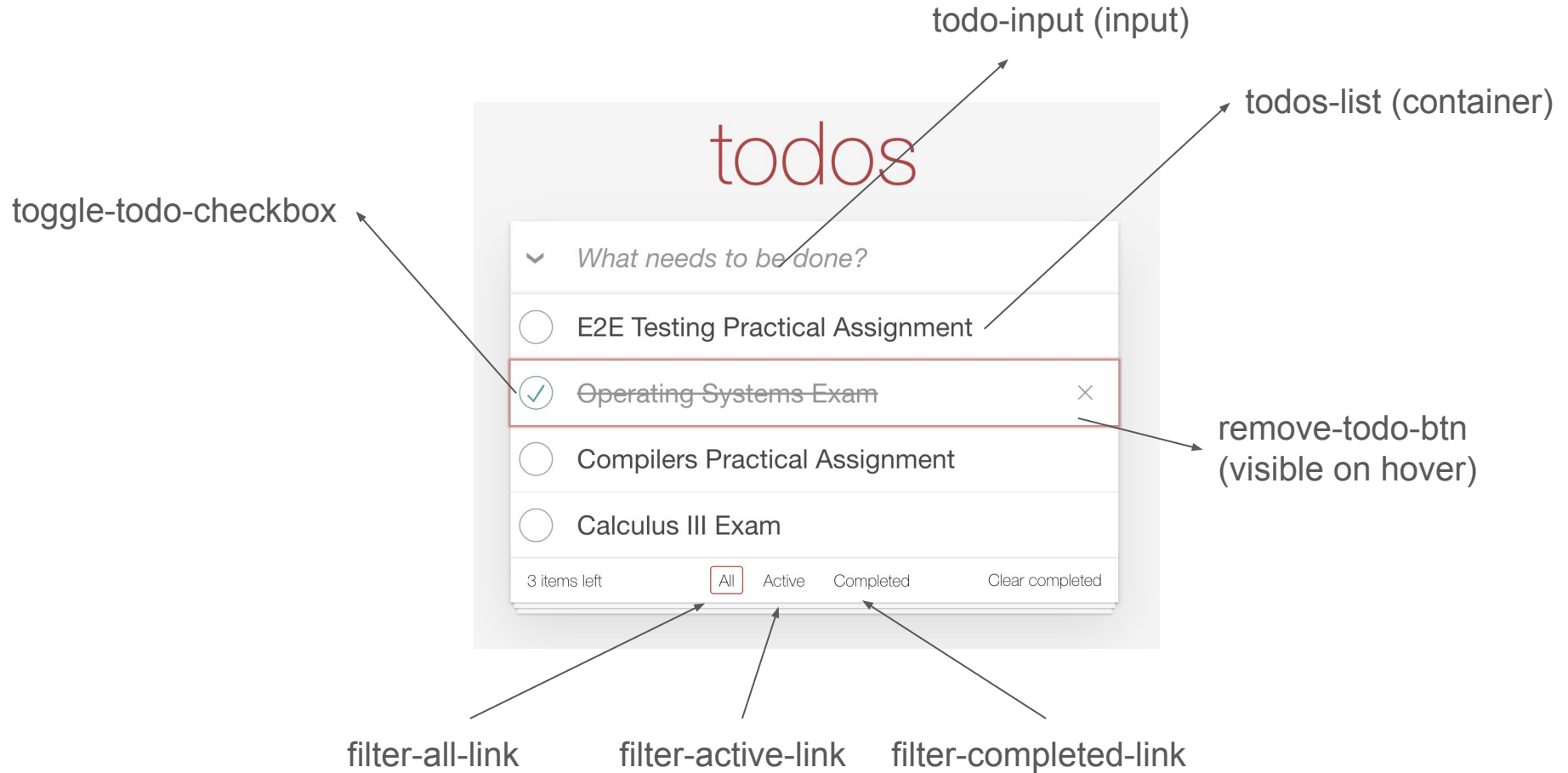
All

Active

Completed

Clear completed

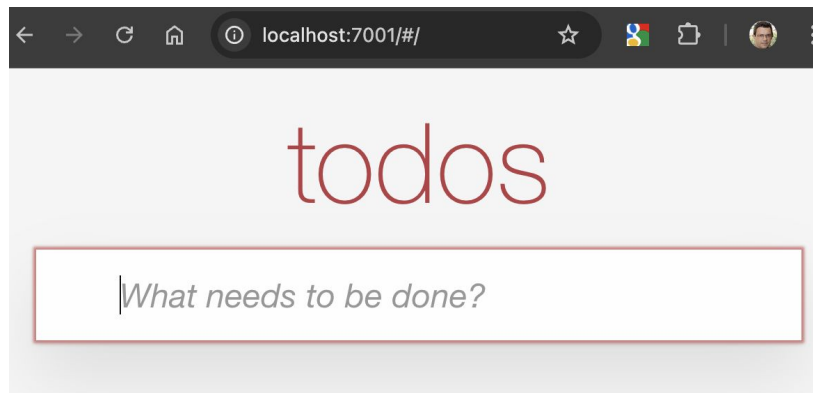
<https://todomvc.com>



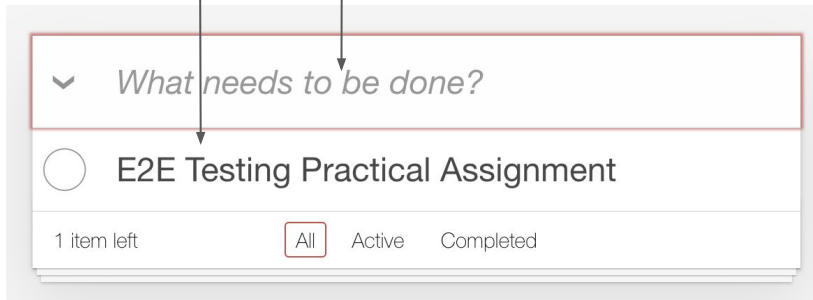
# HTML & data-cy Selectors

```
<input data-cy="todo-input" ... placeholder="What needs to be done?"/>
...
<input data-cy="toggle-todo-checkbox" type="checkbox" {{checked}}>
...
<button data-cy="remove-todo-btn" class="destroy"></button>
...
<ul class="filters">
  <li> <a data-cy="filter-all-link" href="#/" ...>All</a> </li>
  <li> <a data-cy="filter-active-link" href="#/active">Active</a> </li>
  <li> <a data-cy="filter-completed-link"
        href="#/completed">Completed</a> </li>
</ul>
```

```
it('Checking if the app is opening', () => {  
  cy.visit('')  
})
```



```
it('Inserting a task', () => {  
  cy.visit('');  
  
  cy.get('[data-cy=todo-input]')  
    .type('E2E Testing Practical Assignment{enter}');  
  
  cy.get('[data-cy=todos-list]')  
    .children()  
    .should('have.length', 1)  
    .first()  
    .should('have.text', 'E2E Testing Practical Assignment');  
});
```



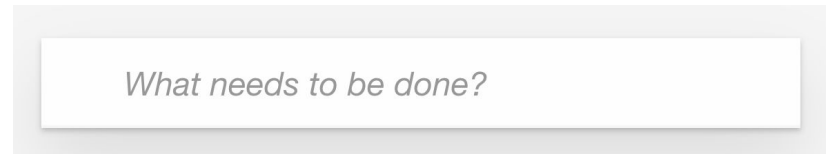
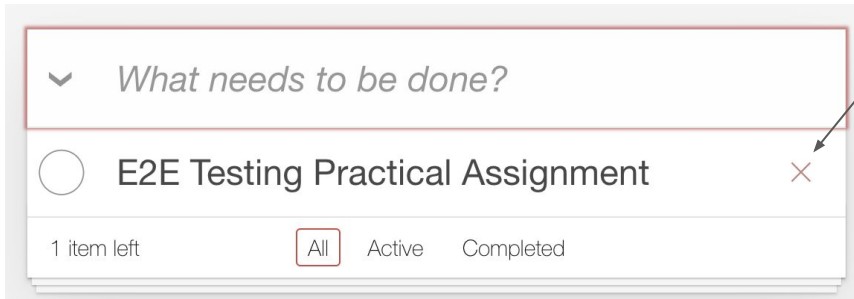
```
it('Inserting and deleting a task', () => {  
  cy.visit('');  
  
  cy.get('[data-cy=todo-input]')  
    .type('E2E Testing Practical Assignment{enter}');  
  
  cy.get('[data-cy=todos-list]')  
    .children()  
    .should('have.length', 1);  
})
```

No news so far for the previous test

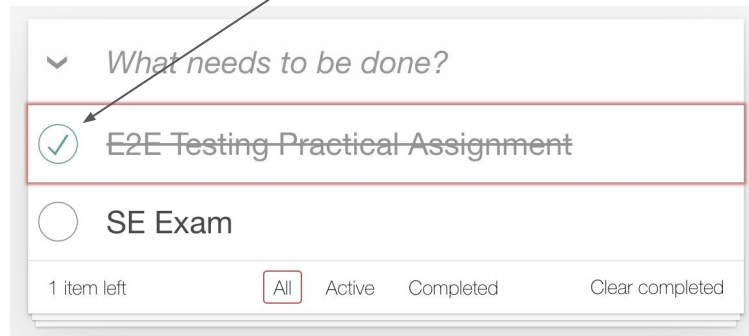
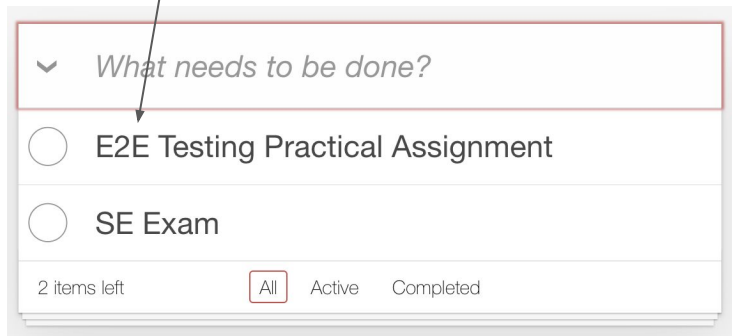
Continue...

```
cy.get('[data-cy=todos-list] > li [data-cy=remove-todo-btn]')  
  .invoke('show')  
  .click();
```

```
cy.get('[data-cy=todos-list]')  
  .children()  
  .should('have.length', 0);  
});
```



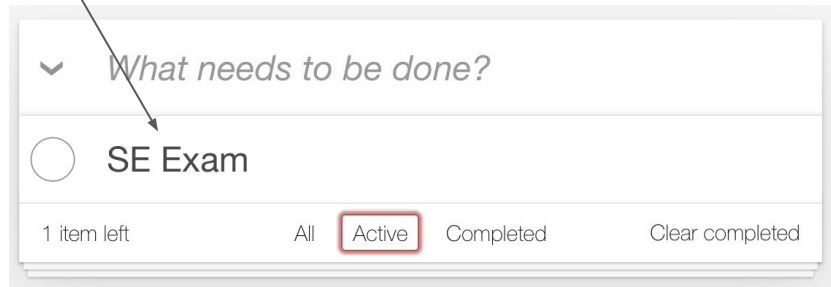
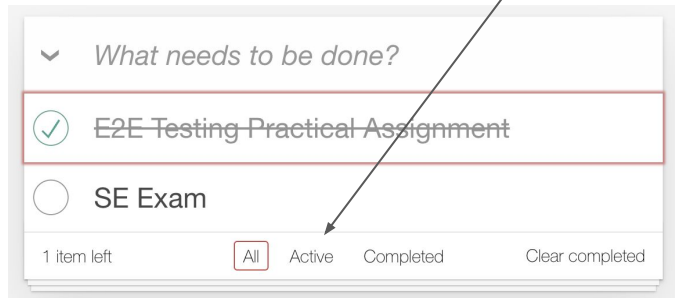
```
it('Selecting completed and active tasks', () => {  
  cy.visit('');  
  
  cy.get('[data-cy=todo-input]')  
    .type('E2E Testing Practical Assignment{enter}')  
    .type('SE Exam{enter}');  
  
  cy.get('[data-cy=todos-list] > li [data-cy=toggle-todo-checkbox]')  
    .first()  
    .click();
```





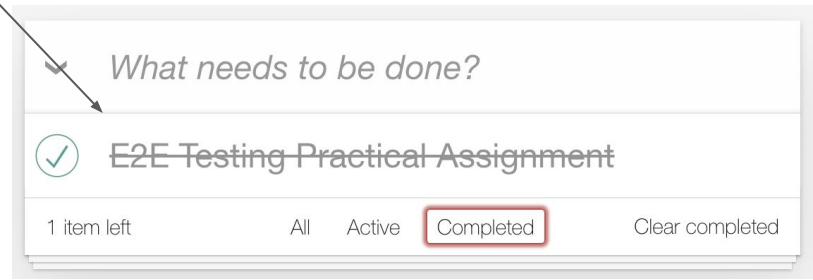
```
cy.get('[data-cy=filter-active-link'])  
  .click();
```

```
cy.get('[data-cy=todos-list]')  
  .children()  
  .should('have.length', 1)  
  .first()  
  .should('have.text', 'SE Exam');
```



Continue...

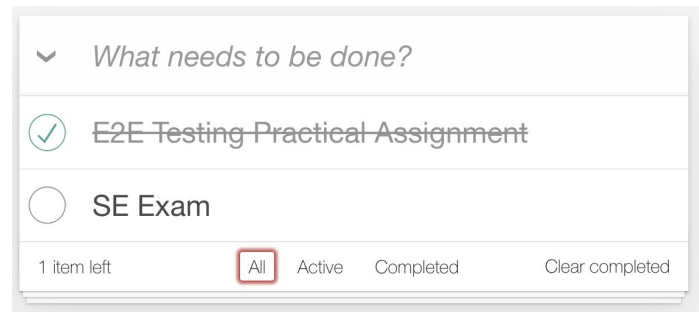
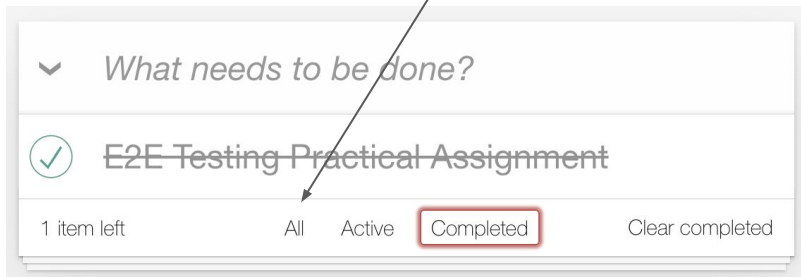
```
cy.get('[data-cy=filter-completed-link'])  
  .click();  
  
cy.get('[data-cy=todos-list]')  
  .children()  
  .should('have.length', 1)  
  .first()  
  .should('have.text', 'E2E Testing Practical Assignment');
```




Continue...


## Continuação...

```
cy.get('[data-cy=filter-all-link'])  
  .click();  
  
cy.get('[data-cy=todos-list]')  
  .children()  
  .should('have.length', 2);  
});  
});
```



## > Selecting Elements

 **Anti-Pattern:** Using highly brittle selectors that are subject to change.

 **Best Practice:** Use `data-*` attributes to provide context to your selectors and isolate them from CSS or JS changes.

<https://docs.cypress.io/app/core-concepts/best-practices#Selecting-Elements>

**End**