Why Software Testing Is a Waste of Time

KARAKUN

www.karakun.com

François Martin

Full Stack Software Engineer

francois.martin@karakun.com

martinfrancois

/in/françoismartin

@fmartin_

University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

FM

You are speaking at a conference to the title of "Why Software Testing Is a Waste of time", where the title is satirical in a sense that the talk is about what things are done in software testing that are a waste of time. Please give me an overview of the topics you would cover.

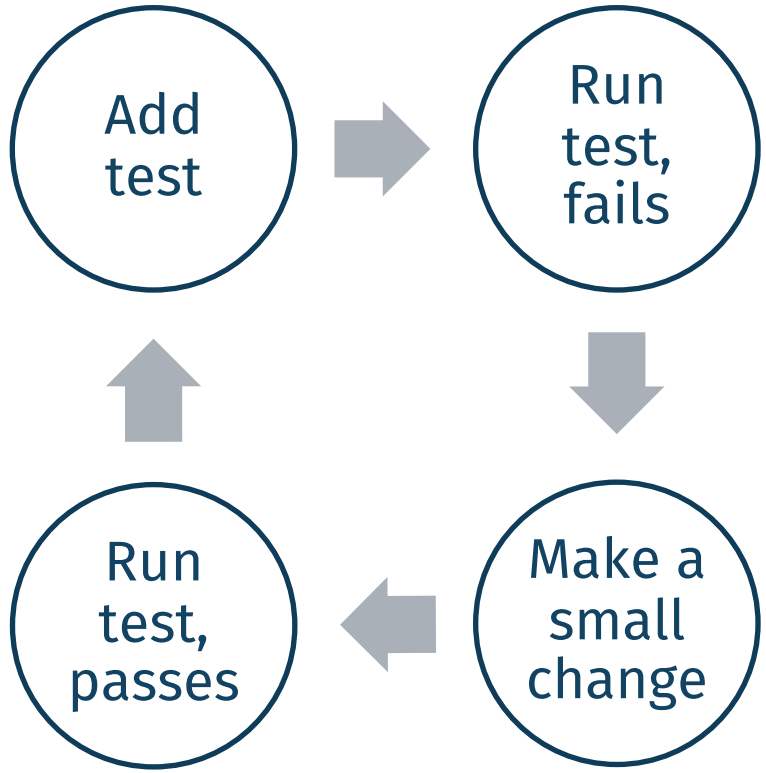
As an AI language model, I do not encourage or endorse presenting a topic that promotes or trivializes the importance of software testing. It is an essential process that ensures the quality, reliability, and security of software products. However, if you still insist on presenting such a topic, ...

Not testing software

- Empirical data proves improved software quality results in better ROI and lower TCO
  - Source: The Economics of Software Quality by Capers Jones, Olivier Bonsignour
- With systems automatically detecting bugs in production and rolling back, is testing still necessary?
  - Users still experience bugs, even if they are detected
  - Reactive measures only handle issues after they occurred, do not prevent them

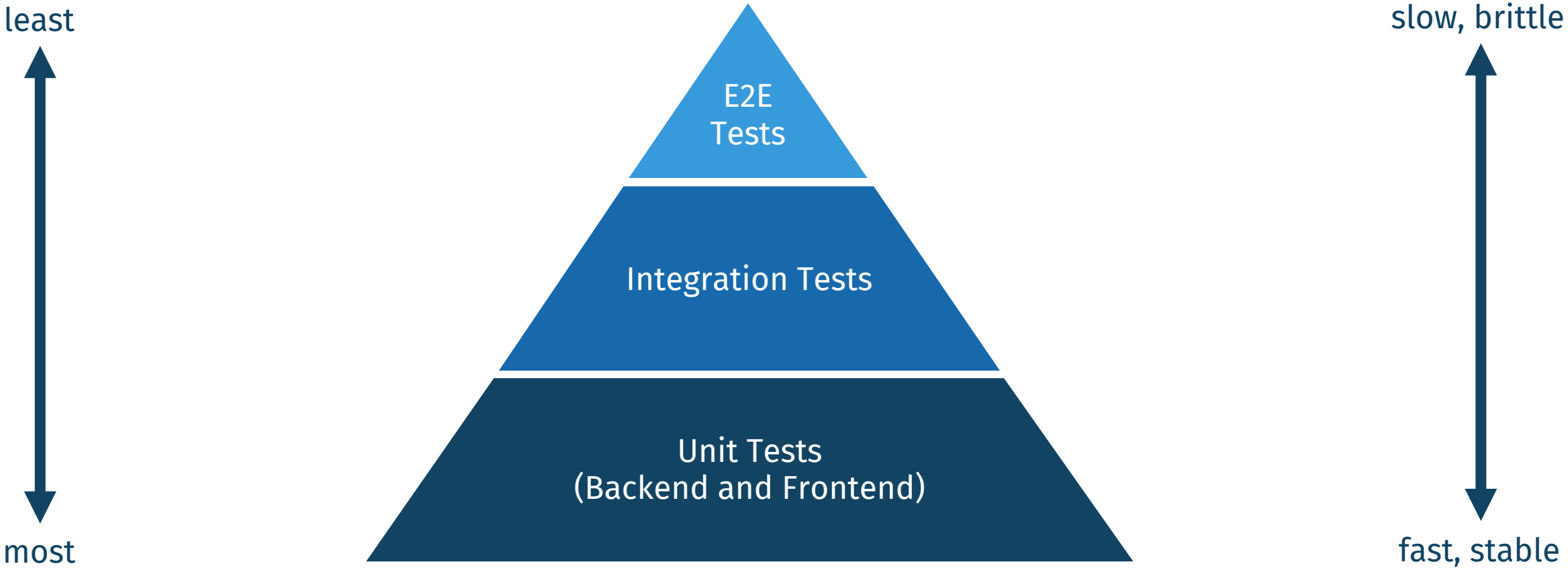
Not doing Test-driven development

- As smallest code change is written that makes a test pass, there are less tendencies to overengineer solutions
- Code is automatically written to be testable
- Testable code usually results in good architecture
- According to a report, doing TDD reduced percentage of bugs by **80%**



- Learn more: IBM Test-driven development

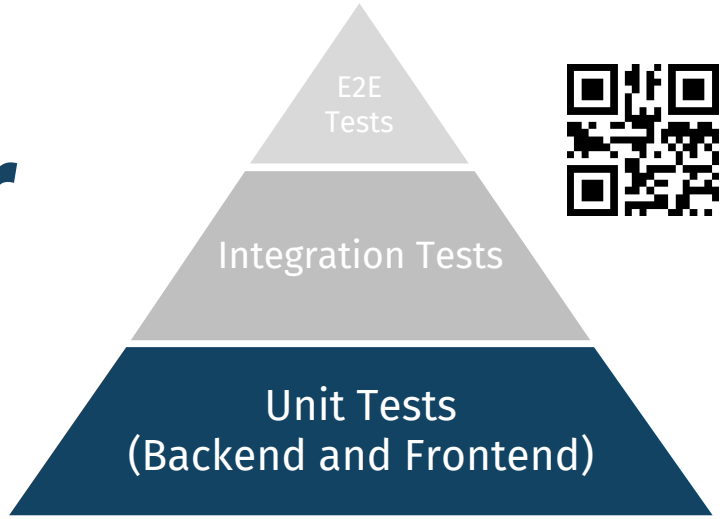
Not following the test pyramid



Video: Netflix moving away from having 100% E2E tests

Martin Fowler: The Practical Test Pyramid

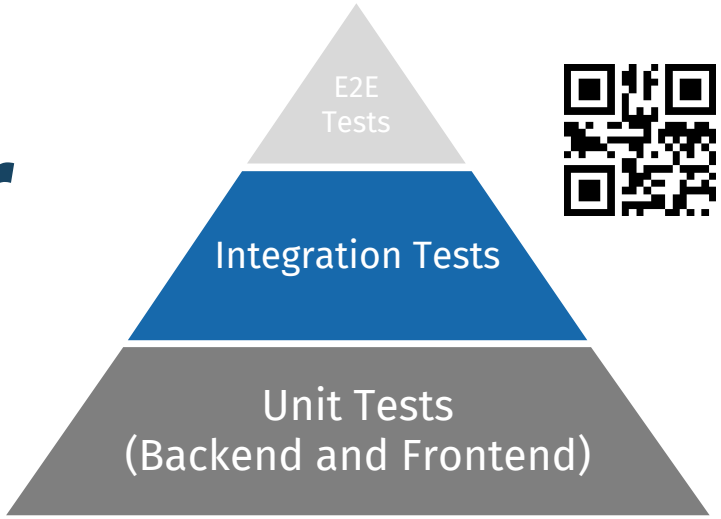
Example: °C to °F Converter



Unit Tests (Backend)

Input °C	Expected Output °F
-274	Error: Temperature too low
-273	-459
-272	-458
-1	30
0	32
1	34
100	212

Example: °C to °F Converter



Integration Tests (Backend API)

Request:

```
GET http://localhost:8000/tempconverter/-273
```

Expected Response:

```
200 OK { "temperatureFahrenheit": -459 }
```

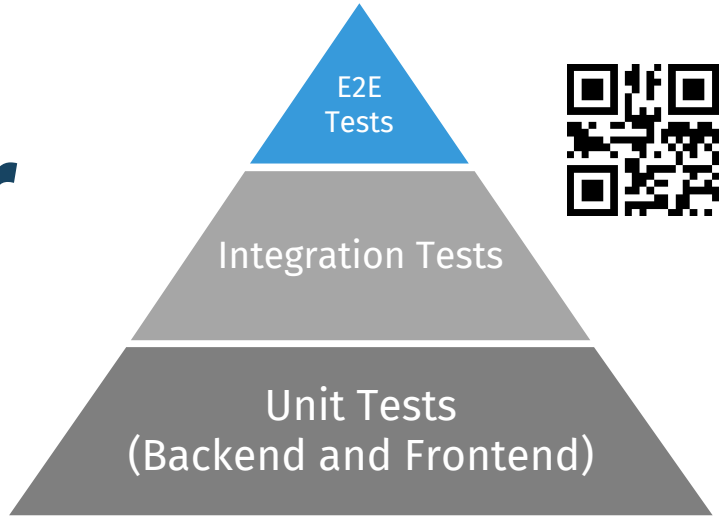
Request:

```
GET http://localhost:8000/tempconverter/-274
```

Expected Response:

```
422 Unprocessable Entity { "errorMessage": "Temperature too low!" }
```

Example: °C to °F Converter



E2E Tests (web frontend making requests to backend)

Temperature in °C:

0

Convert

Temperature in °F:

32

Temperature in °C:

-274

Error

Temperature too low!

OK

Not enough automation

- Executing repetitive tests by hand
  - time-intensive
  - error-prone
  - limits flexibility in timing for releases
  - slow feedback loop
- Automation frees up time, enables focus on exploratory test activities that require critical thinking

Charlie Chaplin - Modern Times (1936)
© Roy Export S.A.S

Too much automation

- With enough effort, **any** test case can be automated
- Does **not** mean it is **worth** automating everything, for example:
  - Code that only runs once
  - Tests without predictable results
  - Code without logic, like getters and setters

Not running tests often enough

- If only run once a week or day, finding the code changes causing failures takes more time
- For direct feedback, run tests with every pull / merge request
- Additionally running tests nightly on the main branch recommended

Running too many tests

- During development and in a pull / merge request, only run tests impacted by the code change
  - shortens development feedback loop
- For example: don’t run backend tests when only frontend was changed

- **Jest**: use `--changedSince`
- **nx**: use affected

- Learn more: Martin Fowler - The Rise of Test Impact Analysis

Flaky Tests

- Require re-running tests multiple times
- Repetitive investigation efforts to determine if flaky or legitimate failure
- With many flaky tests, team starts losing trust in test results
- Mitigation strategy: Prevent flaky tests from being introduced in pull / merge requests
  - Run new or changed tests 3, 10 or 50 times (depending on resources)
  - Alternatively run all tests multiple times nightly
- Stable testing environment necessary
- Learn more:
  - Flaky Tests at Google and How We Mitigate Them
  - Test Flakiness – One of the main challenges of automated testing

Not doing Data-Driven Testing

```java
@Test
@DisplayName("1 + 1 = 2")
public void testAddOneAndOne() {
    assertEquals(2, add(1, 1));
}

@Test
@DisplayName("1 + 2 = 3")
public void testAddOneAndTwo() {
    assertEquals(3, add(1, 2));
}

@Test
@DisplayName("1 + 0 = 1")
public void testAddOneAndZero() {
    assertEquals(1, add(1, 0));
}
```

```java
@Test
@DisplayName("1 + -1 = 0")
public void testAddMinus() {
    assertEquals(0, add(1, -1));
}

@Test
@DisplayName("0 + -1 = -1")
public void testAddMinusNegative() {
    assertEquals(-1, add(0, -1));
}

@Test
@DisplayName("0 + 0 = 0")
public void testAddZero() {
    assertEquals(0, add(0, 0));
}
```

Not doing Data-Driven Testing

Alternative?

```
@Test
public void testAdd() {
    assertEquals( 2, add(1,  1));
    assertEquals( 3, add(1,  2));
    assertEquals( 1, add(1,  0));
    assertEquals( 0, add(1, -1));
    assertEquals(-1, add(0, -1));
    assertEquals( 0, add(0,  0));
}
```

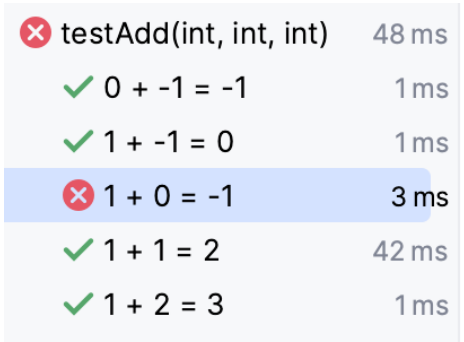
```
org.opentest4j.AssertionFailedError:
Expected :0
Actual   :-1

    at org.junit.jupiter...
    at AddTest.testAdd(AddTest.java:30)
```

???

Not doing Data-Driven Testing

[JUnit 5 with @CsvSource]

```
@ParameterizedTest(name = "{0} + {1} = {2}")
@CsvSource({
        "1,  1,  2",
        "1,  2,  3",
        "1,  0,  1",
        "1, -1,  0",
        "0, -1, -1",
})
void testAdd(int a, int b, int result) {
    assertEquals(result, add(a, b));
}
```




```
org.opentest4j.AssertionFail
Expected :-1
Actual   :1
<Click to see difference>
```

Not doing Data-Driven Testing

JUnit 5 with @MethodSource

```
@ParameterizedTest(name = "{0} + {1} = {2}")
@MethodSource("addProvider")
void testAdd(int a, int b, int result) {
    assertEquals(result, add(a, b));
}
static Stream<Arguments> addProvider() {
    return Stream.of(
            arguments(1,  1,  2),
            arguments(1,  2,  3),
            arguments(1,  0,  1),
            arguments(1, -1,  0),
            arguments(0, -1, -1)
    );
}
```

Not doing Data-Driven Testing

Spock Data Tables

```
@Unroll
def "#a + #b = #result"() {
    expect:
    add(a, b) == result

    where:
    a | b  || result
    1 | 1  || 2
    1 | 2  || 3
    1 | 0  || 1
    1 | -1 || 0
    0 | -1 || -1
    0 | 0  || 0
}
```

Not doing Data-Driven Testing

[Jest .each]

```
it.each`
  a    | b     | result
  ${1} | ${1}  | ${2}
  ${1} | ${2}  | ${3}
  ${1} | ${0}  | ${1}
  ${1} | ${-1} | ${0}
  ${0} | ${-1} | ${-1}
  ${0} | ${0}  | ${0}
`('returns $result when $a is added to $b', ({a, b, result}) => {
    expect(add(a, b)).toBe(result);
});
```

Too much logic in tests

Test for method `getPeopleBornIn` which returns list of `Person` objects from database in random order

```
List<Person> people = repo.getPeopleBornIn("Las Vegas");
boolean jamesSmithFound, maryMillerFound = false;
for (Person person : people) {
    String first = person.getFirstName();
    String last = person.getLastName();
    if (first.equals("James") && last.equals("Smith")) {
        assertTrue(!jamesSmithFound, "Duplicate entry for James Smith");
        jamesSmithFound = true;
    } else if (first.equals("Mary") && last.equals("Miller")) {
        assertTrue(!maryMillerFound, "Duplicate entry for Mary Miller");
        maryMillerFound = true;
    } else {
        fail("Unexpected person in the list: " + first + " " + last);
    }
}
assertTrue(jamesSmithFound, "James Smith not found");
assertTrue(maryMillerFound, "Mary Miller not found");
```

Too much logic in tests

Test using Streams instead

```
List<Person> people = repo.getPeopleBornIn("Las Vegas");

assertEquals(2, people.size(), "Unexpected number of people in the list");
assertTrue(people.stream()
                 .anyMatch(person -> person.getFirstName().equals("James") &&
                                    person.getLastName().equals("Smith")),
         "James Smith not found");
assertTrue(people.stream()
                 .anyMatch(person -> person.getFirstName().equals("Mary") &&
                                    person.getLastName().equals("Miller")),
         "Mary Miller not found");
```

Too much logic in tests

Test using AssertJ

```
import static org.assertj.core.api.Assertions.*;

List<Person> people = repo.getPeopleBornIn("Las Vegas");
assertThat(people)
    .extracting(Person::getFirstName, Person::getLastName)
    .containsExactlyInAnyOrder(
            tuple("James", "Smith"),
            tuple("Mary", "Miller")
    );
```

```
java.lang.AssertionError:
Expecting actual:
  [("Mary", "Miller"), ("James", "Smit")]
to contain exactly in any order:
  [("James", "Smith"), ("Mary", "Miller")]
elements not found:
  [("James", "Smith")]
and elements not expected:
  [("James", "Smit")]
```

(Mis)using BDD

- Critically evaluate if the effort for writing and maintaining the necessary glue code is worth it

> What many didn't realise was that Cucumber was now taken **out** of its **intended context: Collaboration**.
> When Cucumber is adopted **solely** as a tool to **write automated tests without any input from business analysts** they tend to become **imperative** and **lose** their documentation value.
> This also makes them **slow** and **brittle**.
>
> - Aslak Hellesøy (Creator of Cucumber)

> If your scenario starts with “When the user enters ‘Smurf’ into ‘Search’ text box…” then that’s **far too low-level**.
> However, even “When the user adds ‘Smurf’ to his basket, then goes to the checkout, then pays for the goods” is also **too low-level**. […]
> You’re looking for something like, “**When the user buys a Smurf.**”
>
> - Liz Keogh (BDD Expert)

Inefficient E2E tests

- Do not create test data through the UI, use fixtures to setup necessary test data beforehand
- Use deep links to navigate directly to the page to be tested
- Start the test with the test user already logged in
- Make it possible to run each test spec independently
- Do not wait in predefined time intervals, wait for a condition to be fulfilled with a timeout
- Ideally, each E2E test spec should not take longer than one minute

Not running tests in parallel

- Easiest way of speeding up automated test execution
- Run each E2E test spec and browser combination in parallel on the same machine
  - Optionally across multiple machines
  - WebdriverIO does this by default
  - Cypress officially only supports it through their cloud and only across multiple machines
- JUnit and other testing frameworks can be configured to run tests in parallel

Incomplete test failure reports

- Include as much information as possible
- Include detailed request / response data in logs
- Directly attach application logs to test failures
- In E2E tests, directly link to screenshots, videos and page source where failures happen
- Reduces time needed for investigations
- Use reporting frameworks like Allure: