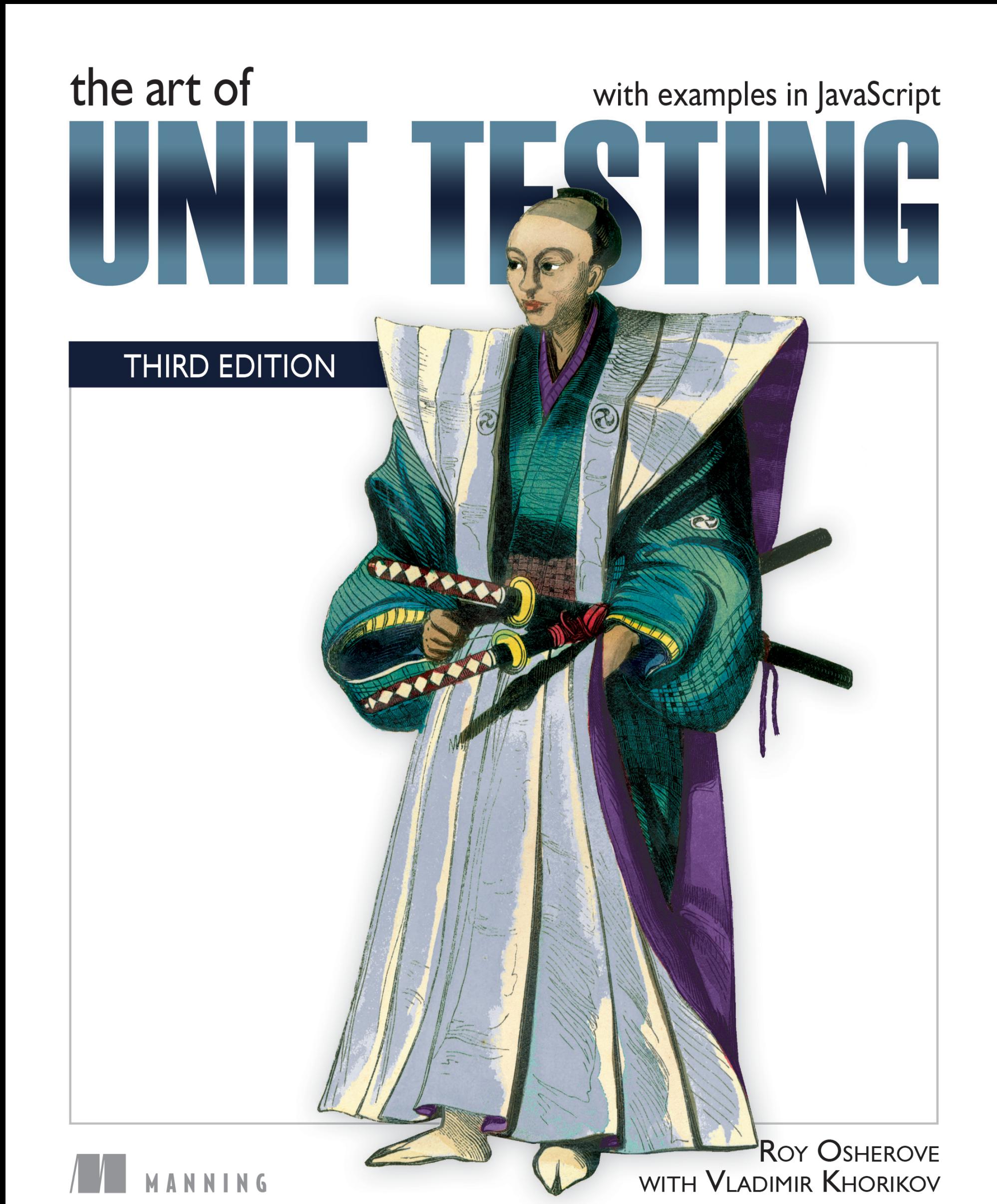


# The Art of Unit Testing

# The Art of Unit Testing

## Study Group in Gogoro



# WebConf 2024

## Flexible Front-end Testing

2024 | 12.27 - 28  
瓶蓋工廠台北製造所

FUTURE PLANS @ TAIWAN

WEBCONF

Summer

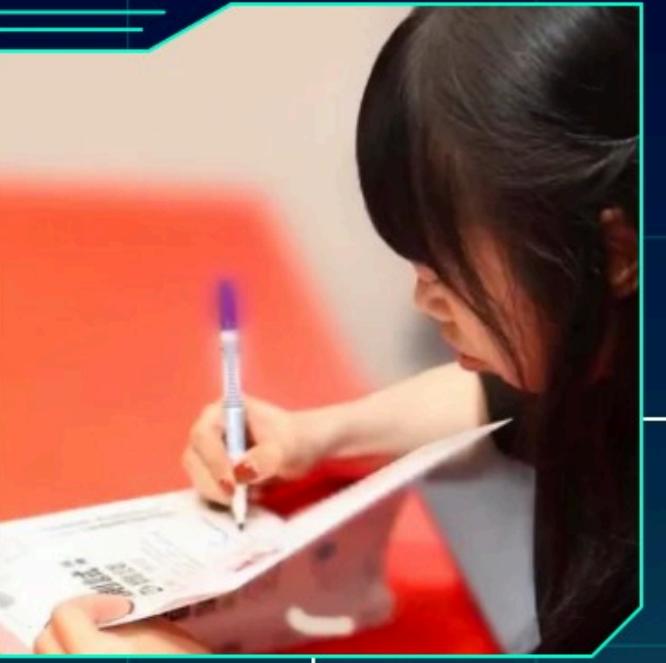
資深工程師

Appier

#講題公布

如何撰寫具彈性的測試程式

NON



# The Art of Unit Testing

- The Story of Google Web Server
- Test Pyramid
- 12 Concepts

# The Story of Google Web Server

Software Engineering at Google

O'REILLY®

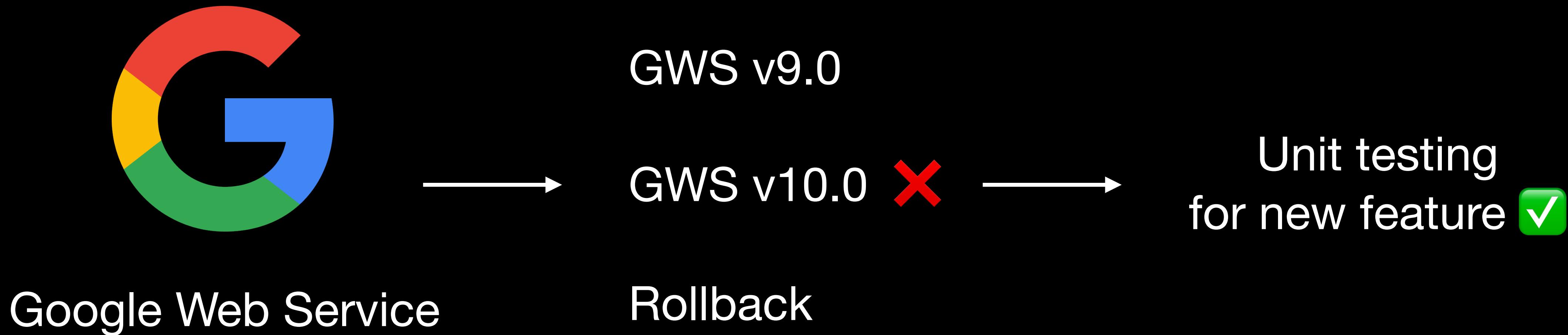
# Software Engineering at **Google**

Lessons Learned  
from Programming  
Over Time



Curated by Titus Winters,  
Tom Mansreck & Hyrum Wright

# The Story of Google Web Server



You can't rely on programmer ability  
alone to avoid product defects

# Reshape Google's engineering culture.

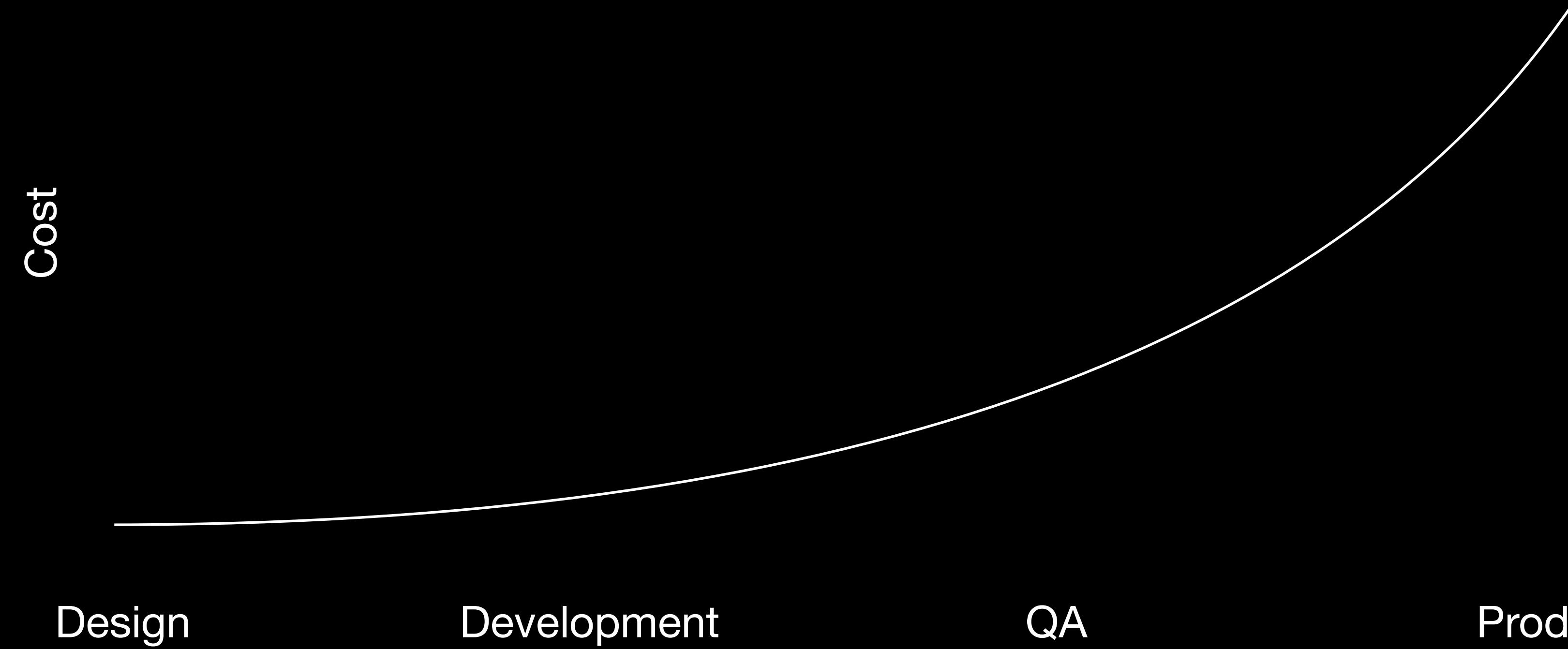
- Orientation Classes
- Test Certified
- Testing on the Toilet
- [Google Testing Blog](#)

Bugs suck.



Tests rock.

# Shift Left



# Beyoncé Rule

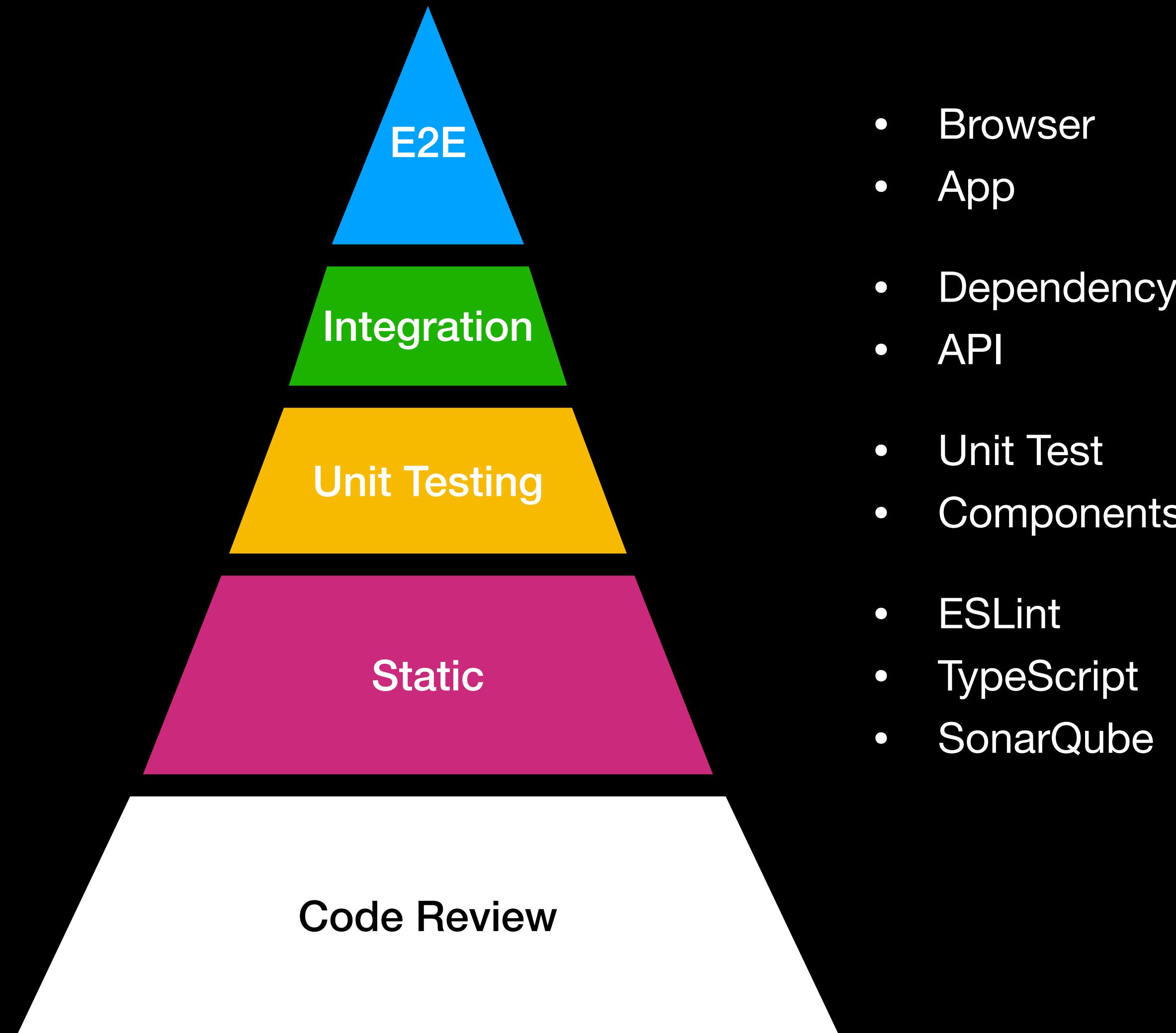
“If you liked it, then you shoulda  
put a test on it.”



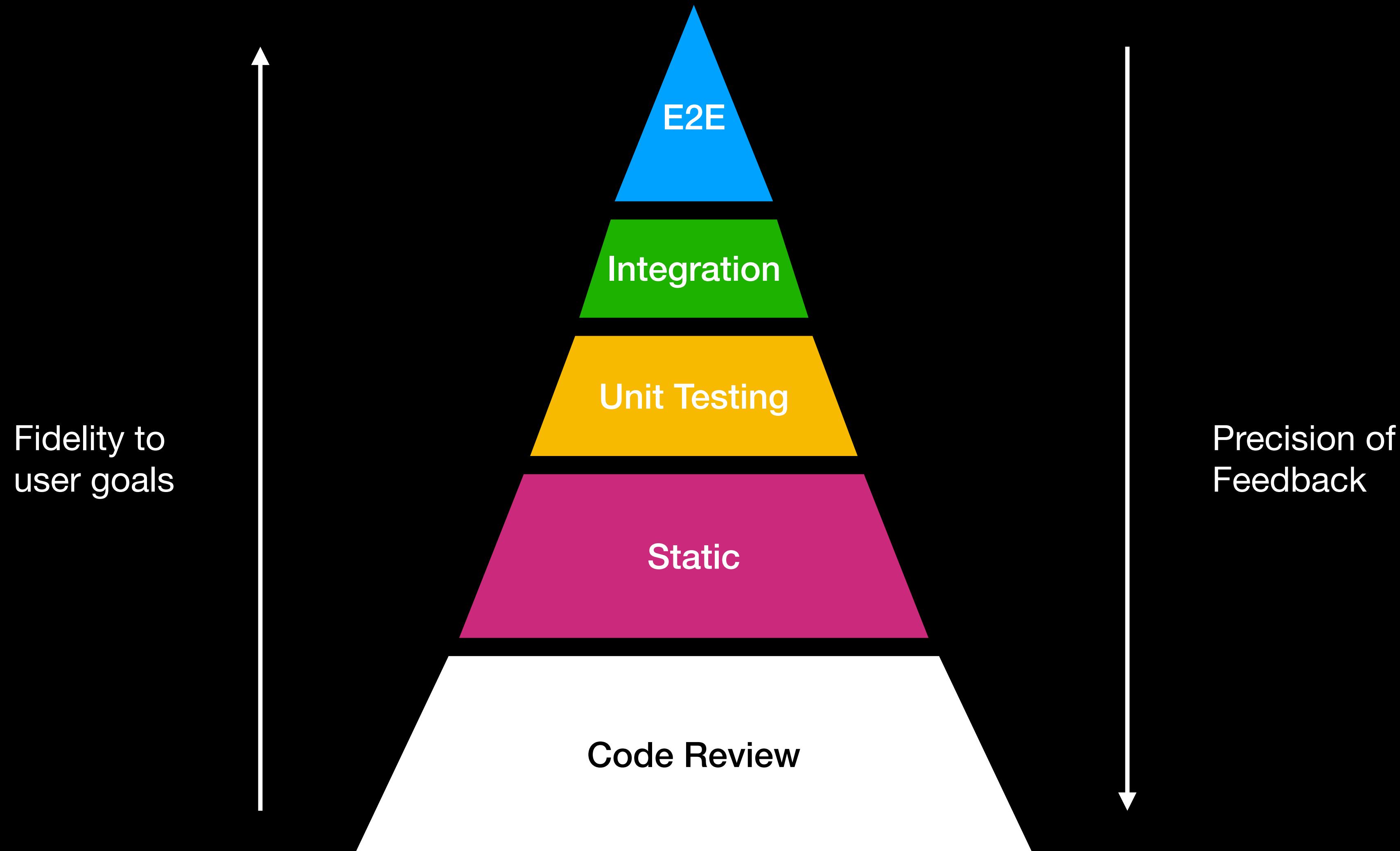
# Other tiers of testing

- Performance testing
- Visual Testing
- Security testing
- Localization testing
- Privacy testing

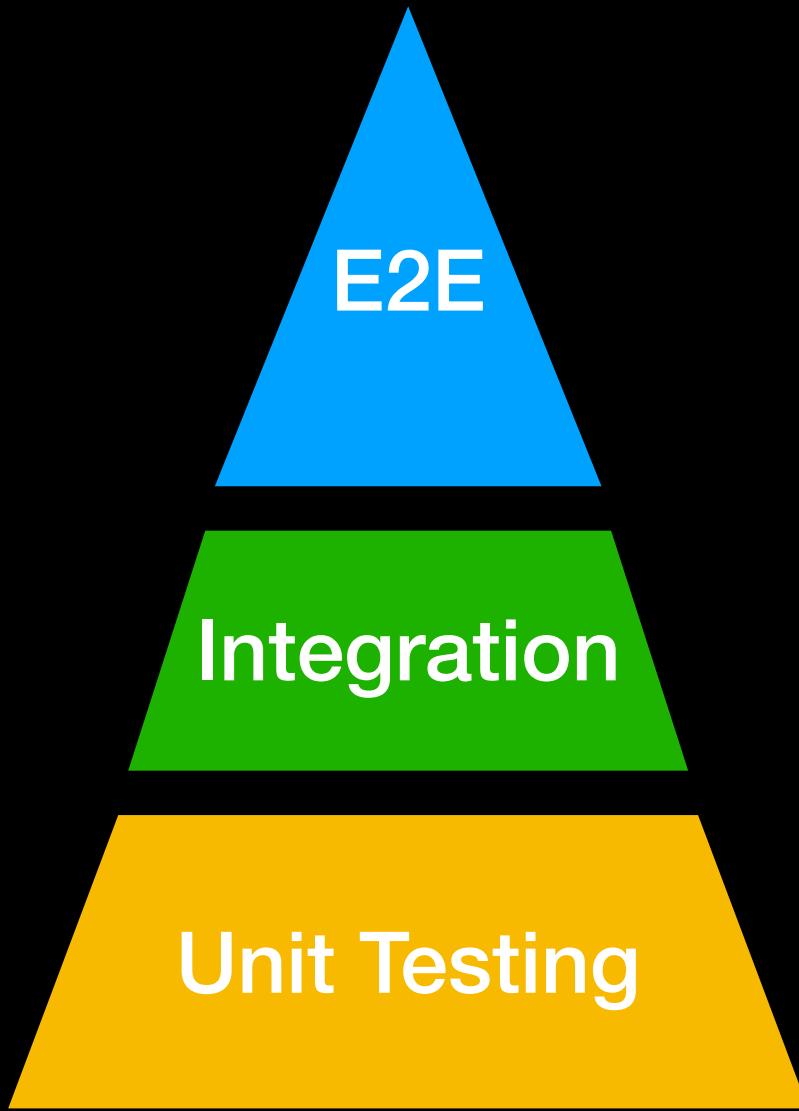
# Test Pyramid



# Test Pyramid



# Test Pyramid in Overdue Page



- Overdue UI
- Playwright Codegen
- Payment API Endpoint
- POST api/my/bill/:billId/pay
- getInvoiceMonths()
- <PaymentMethodModal />



# Playwright Codegen

Generate tests for you as you  
perform actions in the browser



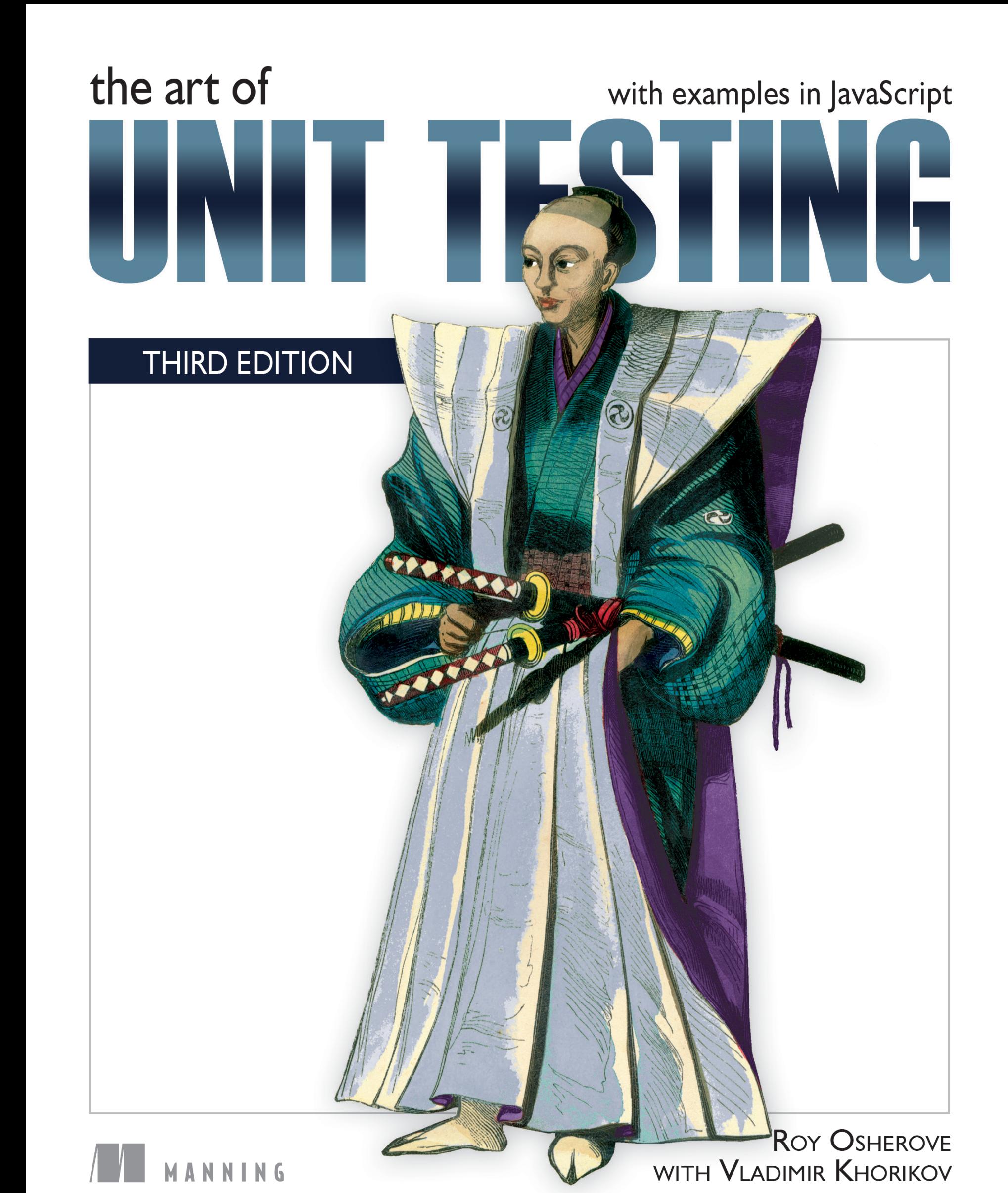
# Swagger Codegen

Generate API integration tests  
from Swagger



# The Art of Unit Testing

## 12 Concepts



# The Art of Unit Testing

- Readability
- Maintainability
- Trust
- Design and Process

# Readability

If we can't read it, then it's hard to maintain, hard to debug,  
and hard to know what's wrong

# Avoiding logic in unit tests

If you have any of the following inside a unit test, your test contains logic

- switch, if, or else statements
- foreach, for, or while loops
- Concatenations (+ sign, etc.)
- try, catch

# Flow control in unit tests X

```
it('should display current time whenever', () => {
  const timeDisplay = new TimeDisplay();

  const result = timeDisplay.getFormatTime();

  const date = new Date();
  let expectedTime = "It's ";
  if (date.getHours() === 0 && date.getMinutes() <= 1) {
    expectedTime += 'midnight';
  } else if (date.getHours() === 12 && date.getMinutes() === 0) {
    expectedTime += 'noon';
  } else {
    expectedTime += `${date.getHours()}:${date.getMinutes()}`;
  }
  expectedTime += '.';
  expect(result).toBe(expectedTime);
});
```

# Remove logic in unit tests

```
it('should display current time whenever', () => {
  jest.useFakeTimers().setSystemTime(new Date('2025-02-17 00:00:00'));
  const timeDisplay = new TimeDisplay();

  const result = timeDisplay.getFormatTime();

  expect(result).toBe("It's midnight.");
  jest.clearAllTimers();
});
```

# Loop in unit tests X

```
it('should pass the valid email list', () => {
  const validEmailList = [
    'simple@example.com',
    'very.common@example.com',
    'disposable.style.email.with+symbol@example.com',
    'other.email-with-hyphen@example.com',
    'fully-qualified-domain@example.com',
    'CAPITAL@example.com',
  ];

  validEmailList.forEach((email) => {
    expect(isValidEmail(email)).toBe(true);
  });
});
```

# it.each(case) for same pattern test cases

```
it.each([
  'simple@example.com',
  'very.common@example.com',
  'disposable.style.email.with+symbol@example.com',
  'other.email-with-hyphen@example.com',
  'fully-qualified-domain@example.com',
  'CAPITAL@example.com',
])('should pass when email is %s', (email) => {
  expect(isValidEmail(email)).toBe(true);
});
```

# Complex test causes problems

- The test is harder to read and understand
- The test is hard to recreate
- The test is more likely to have a bug or to verify the wrong thing

# DRY vs DAMP

- DRY: Don't Repeat Yourself
- DAMP: Descriptive and Meaningful Phrases

# DAMP: Descriptive and Meaningful Phrases

```
it('should stop task progress', () => {
  const task = new Task();
  task.setTitle('Do the laundry');
  task.setStatus('inProgress');
  task.setAssigneeId(1);

  task.stopProgress();

  expect(task.status).toBe('open');
  expect(task.assigneeId).toBeNull();
});

it('should finish task', () => {
  const task = new Task();
  task.setTitle('Do the laundry');
  task.setStatus('inProgress');
  task.setAssigneeId(1);

  task.finish();

  expect(task.status).toBe('closed');
  expect(task.assigneeId).toBe(1);
});
```

# DRY: Don't Repeat Yourself

```
const createTask = (title, status, assigneeId) => {
  const task = new Task();
  task.setTitle(title);
  task.setStatus(status);
  task.setAssigneeId(assigneeId);
  return task;
};

it('should stop task progress', () => {
  const task = createTask('Do the laundry', 'inProgress', 1);

  task.stopProgress();

  expect(task.status).toBe('open');
  expect(task.assigneeId).toBeNull();
});

it('should finish task', () => {
  const task = createTask('Do the laundry', 'inProgress', 1);

  task.finish();

  expect(task.status).toBe('closed');
  expect(task.assigneeId).toBe(1);
});
```

# beforeEach() and scroll fatigue

```
let task = null;
beforeEach(() => {
  task = new Task();
  task.setTitle('Do the laundry');
  task.setStatus('inProgress');
  task.setAssigneeId(1);
});

it('should stop task progress', () => {
  task.stopProgress();

  expect(task.status).toBe('open');
  expect(task.assigneeId).toBeNull();
});

it('should finish task', () => {
  task.finish();

  expect(task.status).toBe('closed');
  expect(task.assigneeId).toBe(1);
});
```

# Maintainability

If maintaining the test or production code is painful because of the tests, our lives will become a living nightmare

# How often should we change a test after writing it?

It never needs to change  
unless the requirements under test change

# Don't change the test again as

- Refactor the system
- Fix bugs
- Add new features

# Strive for Unchanging Tests

- Avoid testing private or protected methods
- Avoid over specification
- Avoid interaction testing

# Avoid over specification

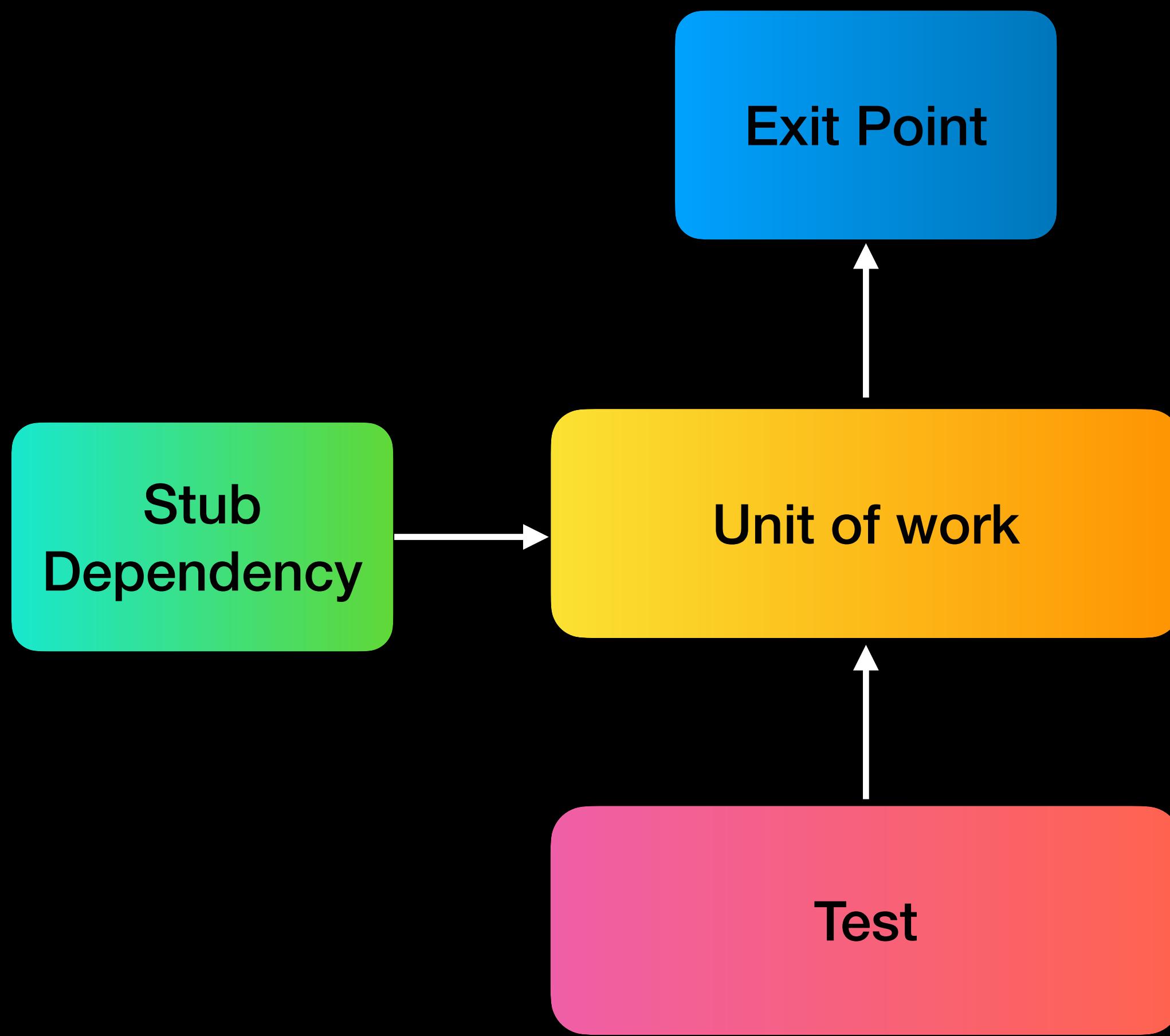
```
<button
  id="main"
  class="btn btn-large"
  name="submission"
  role="button"
  data-testid="submit"
>
  Submit
</button>

// ✗ Highly subject to change
cy.get('.btn.btn-large')

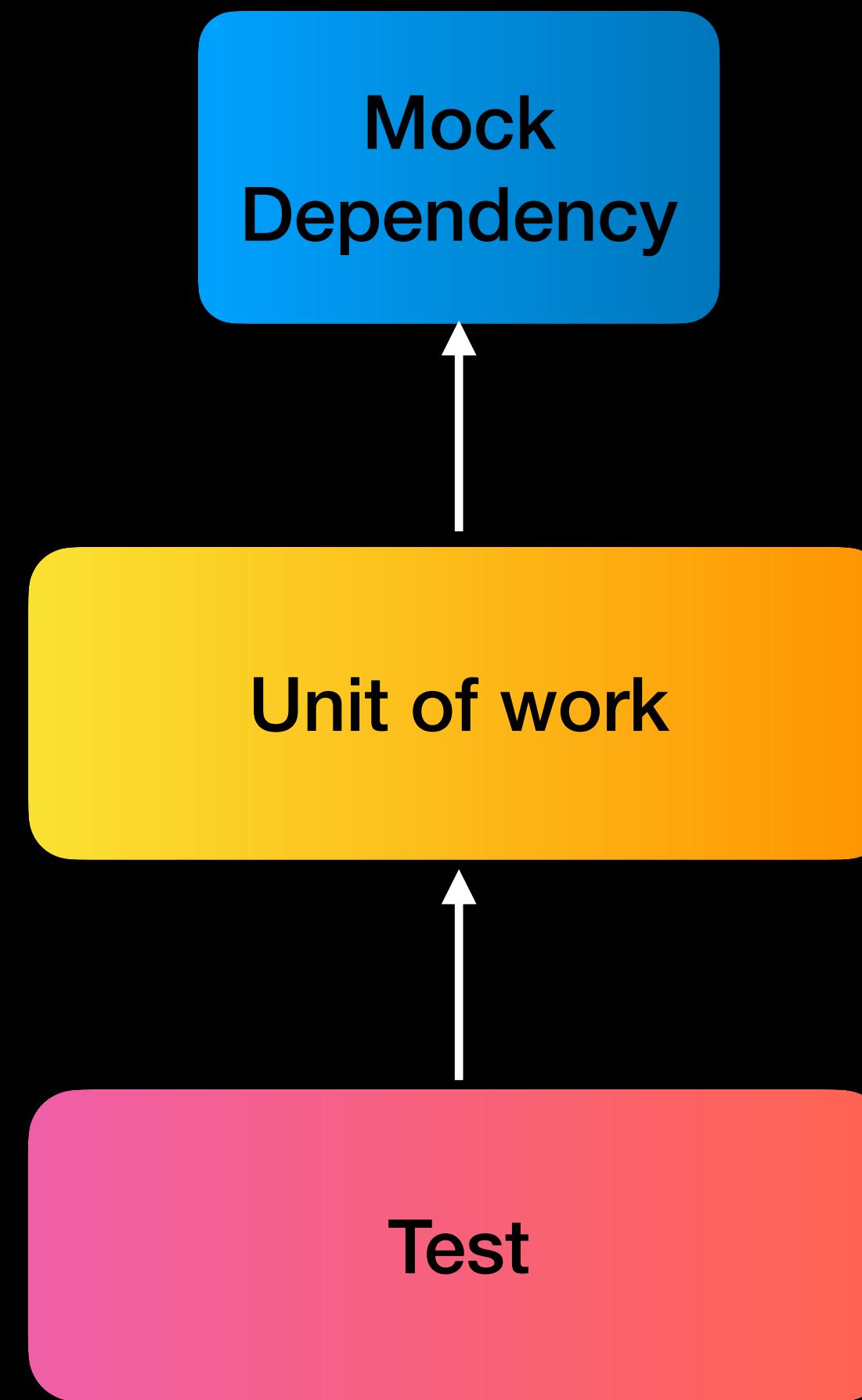
// !! Highly subject to change
cy.contains('Submit')

// 👍 Isolated from all changes
cy.contains('input:button.submit')
cy.get('[data-testid="submit"]')
```

# Stub



# Mock



# Stub and Mock

Difference	Stub	Mock
Dependency	Incoming	Outgoing
Count	$\geq 1$	Only one for maintainability
Assertion	✗	✓
Example	Time	Logger

# Interaction testing X

```
it('should call requestApplePay after clicking Apple Pay', async () => {
  const requestApplePay = jest.fn();
  const mockApplePayAvailable = {
    requestApplePay,
  };
  server = startMirage({}, 'test');
  useApplePay.mockImplementation(() => mockApplePayAvailable);
  const { user } = myGogoroRender(
    <PaymentMethodComponent isPayOpen bill={bill} />,
    State
  );
  const applePay = screen.getByTestId('apple-pay');

  await act(async () => user.click(applePay));

  expect(requestApplePay).toHaveBeenCalledWith(bill);
});
```

# Test state, not interactions

```
it('should popup payment success modal after apple pay success', async () => {
  const mockApplePaySuccess = {};
  useApplePay.mockImplementation(() => mockApplePaySuccess);
  server = startMirage({}, 'test');

  myGogoroRender(<PaymentMethodComponent isPayOpen />, State);

  expect(
    await screen.findByText('my-plan:bill.pay-success')
  ).toBeInTheDocument();
});
```

# Trust

If we don't trust the results of our tests when they fail, we'll start manually testing again, losing all the time benefit the tests are supposed to provide

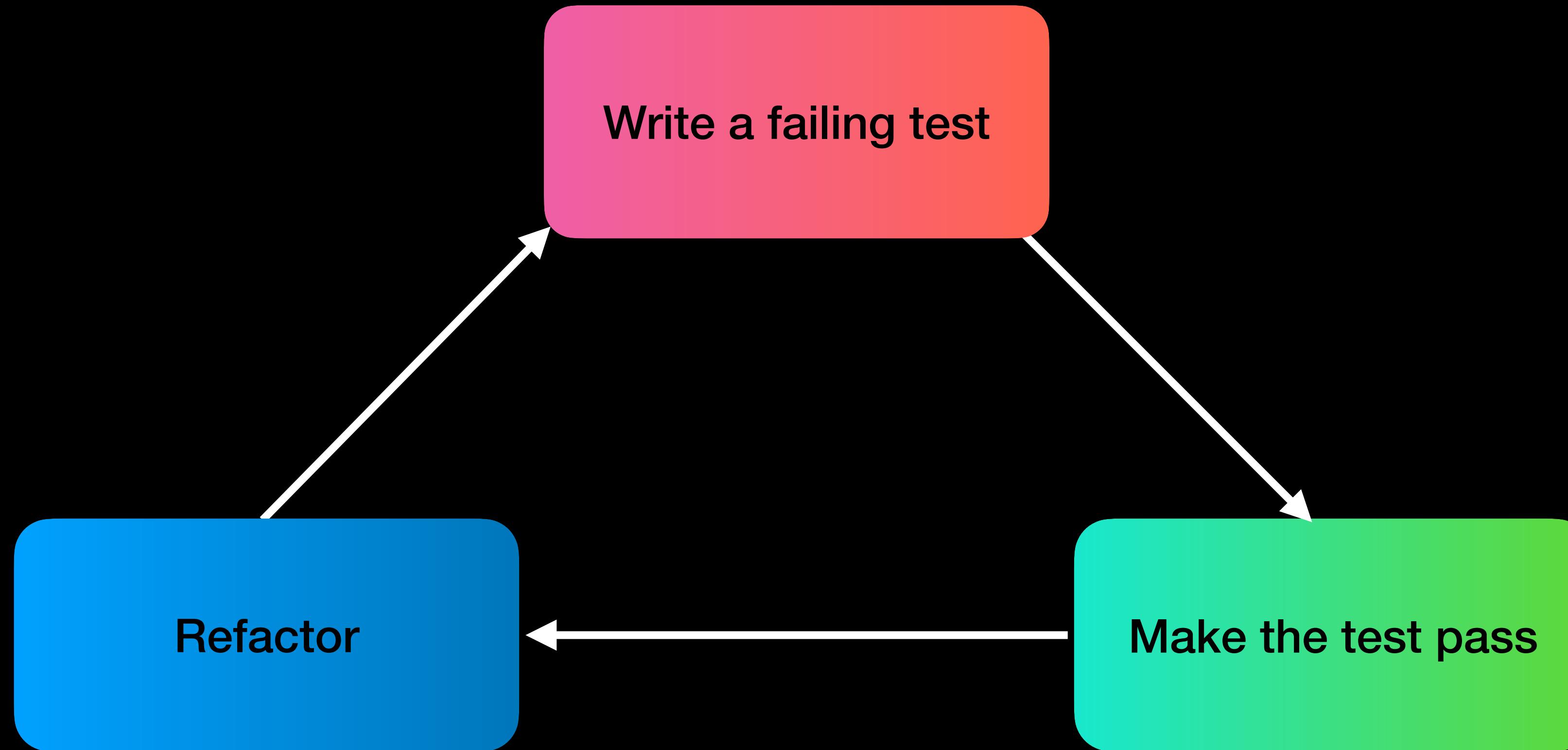
# You might not trust a test if

- It fails and you're not worried
- You feel like it's fine to ignore the results of this test
- It passes and you are worried
- You still feel the need to manually debug or test
- You skip your tests

# TDD

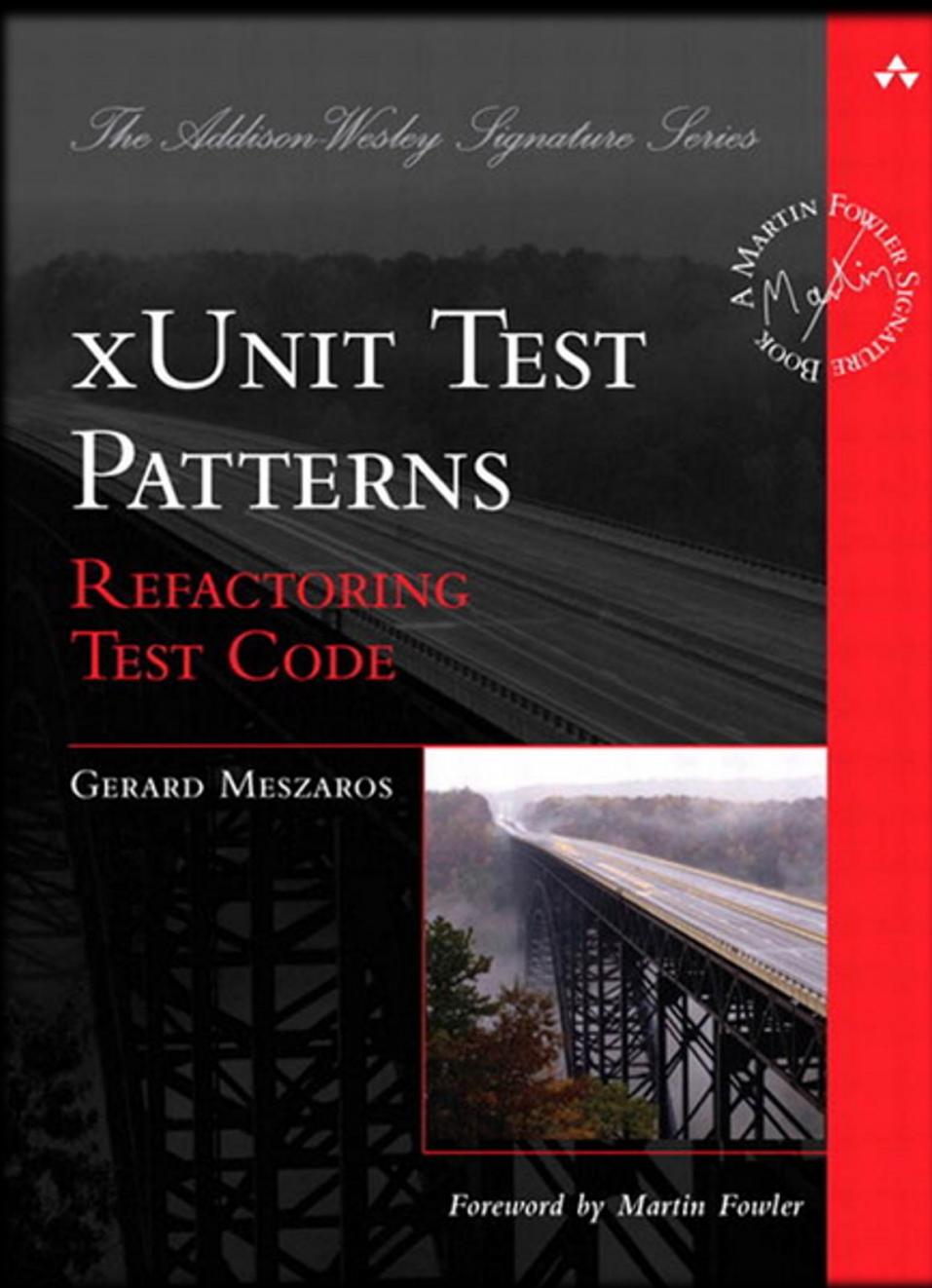
- it fails when it should and it passes when it should
- Use TDD to give **AI** more context and hint

# TDD with AI



# Test Smells

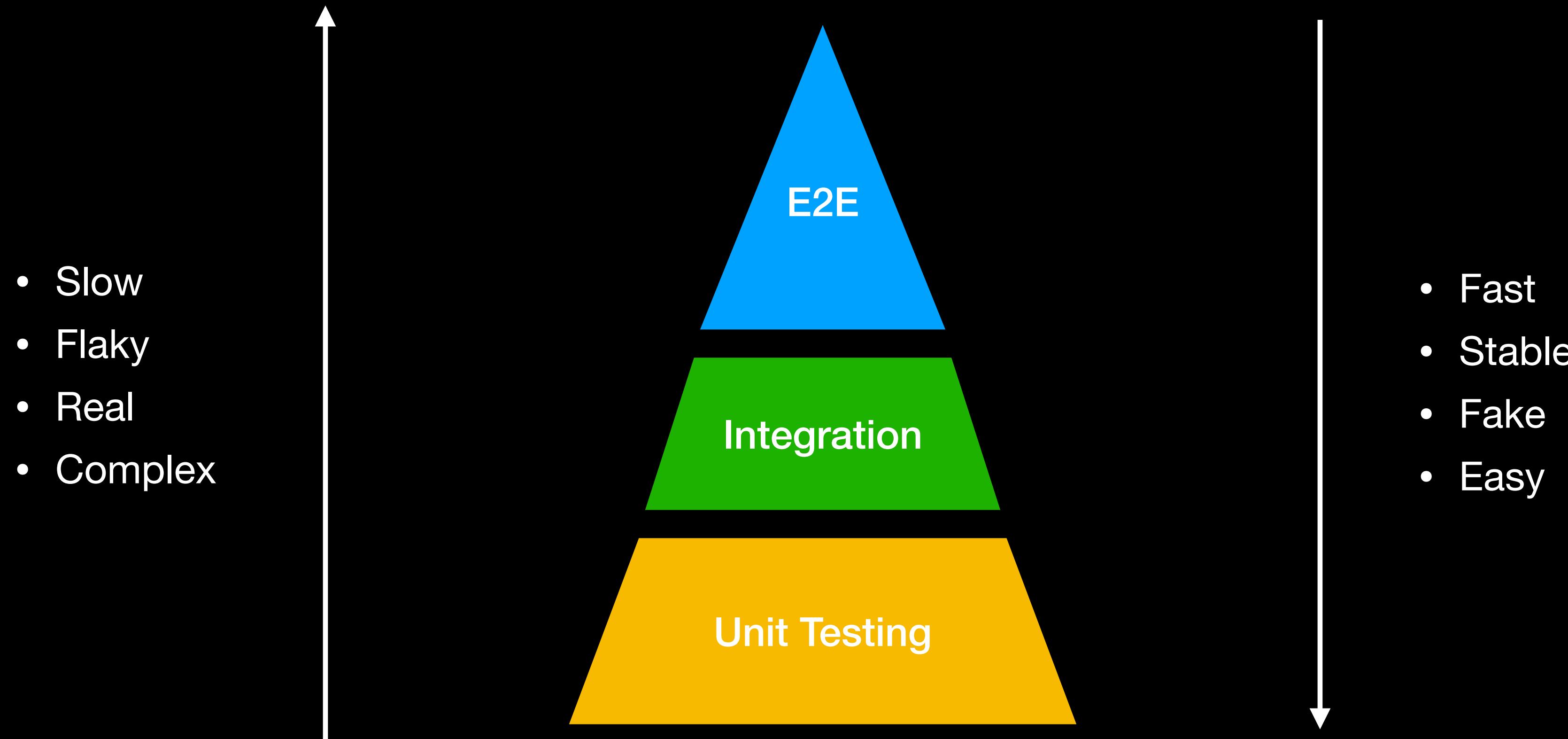
- Hard-to-Test Code
- Fragile Test
- Test Code Duplication
- Test Logic in Production
- Erratic Test
- Obscure Test
- Conditional Test Logic
- Slow Tests



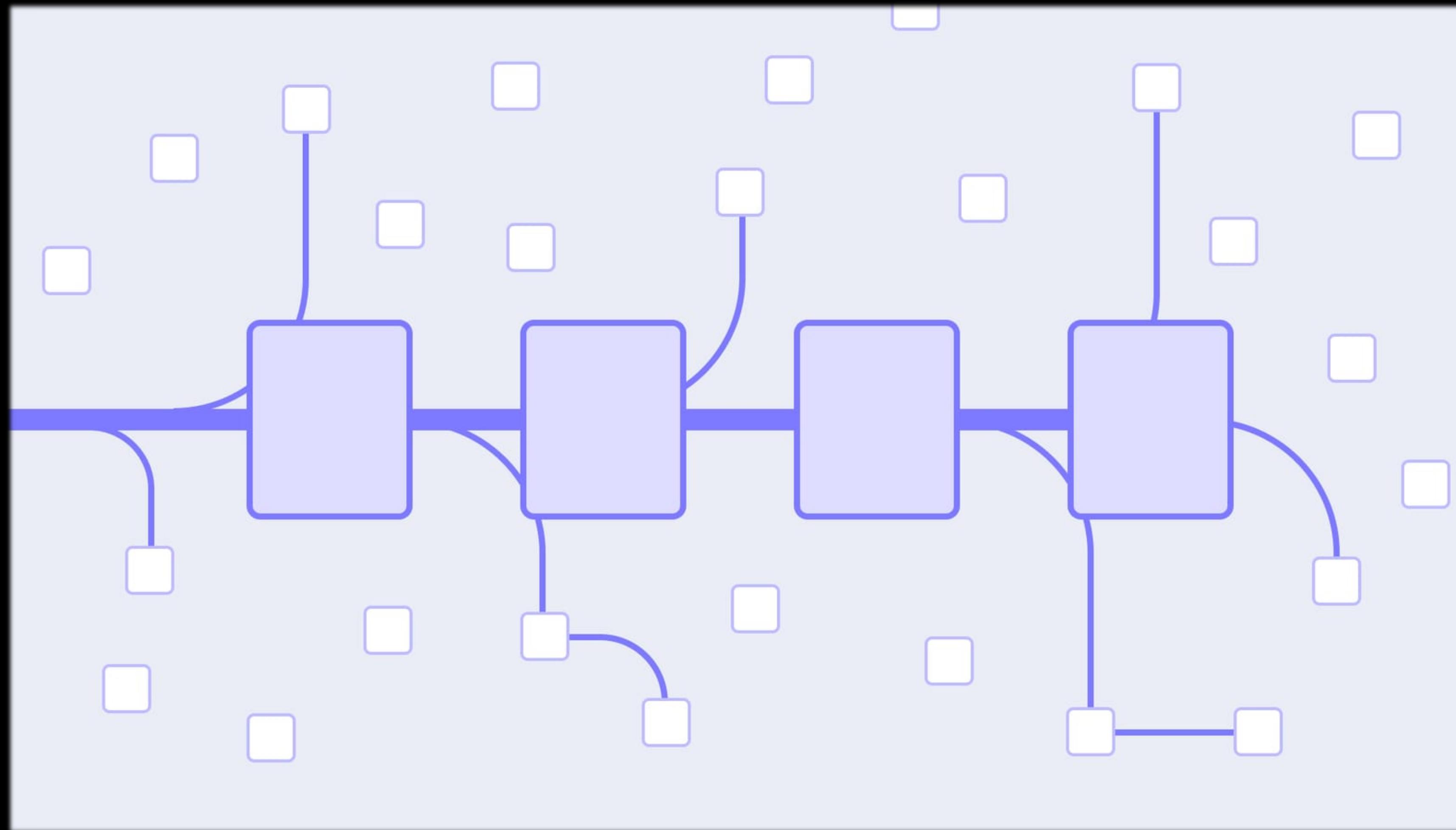
# Design and Process

- Developing a testing strategy with PA team
- Integrating unit testing into the organization

# Developing a testing strategy with PA team

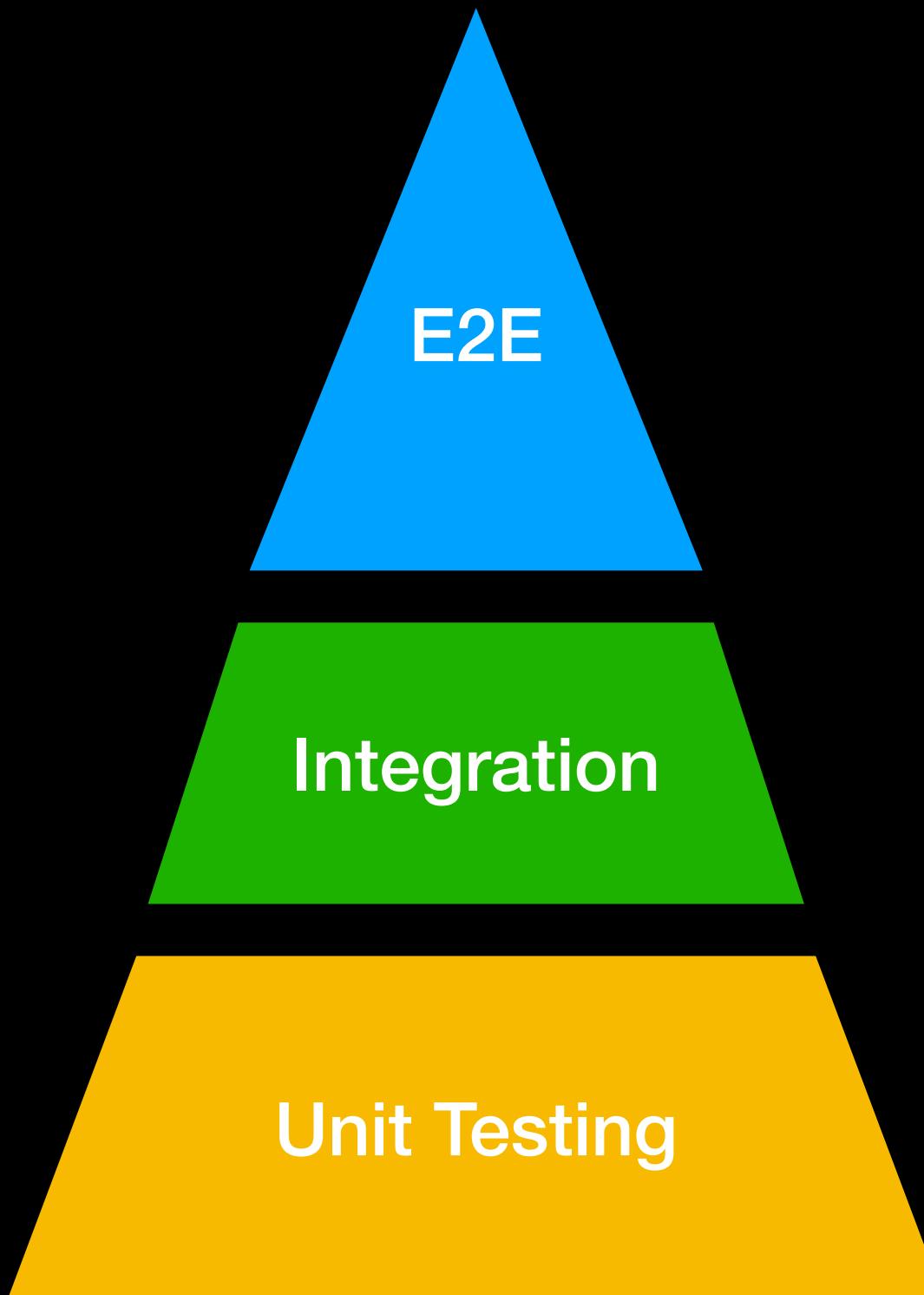


# Prevent overlap testing



# Maintenance

- Review failed test cases
- Trace issues from large system
- Another CI pipeline for flaky testing



# Integrating unit testing into the organization

We have lots of code without tests: Where do we start?

**80% of bugs are found in  
20% of the code.**

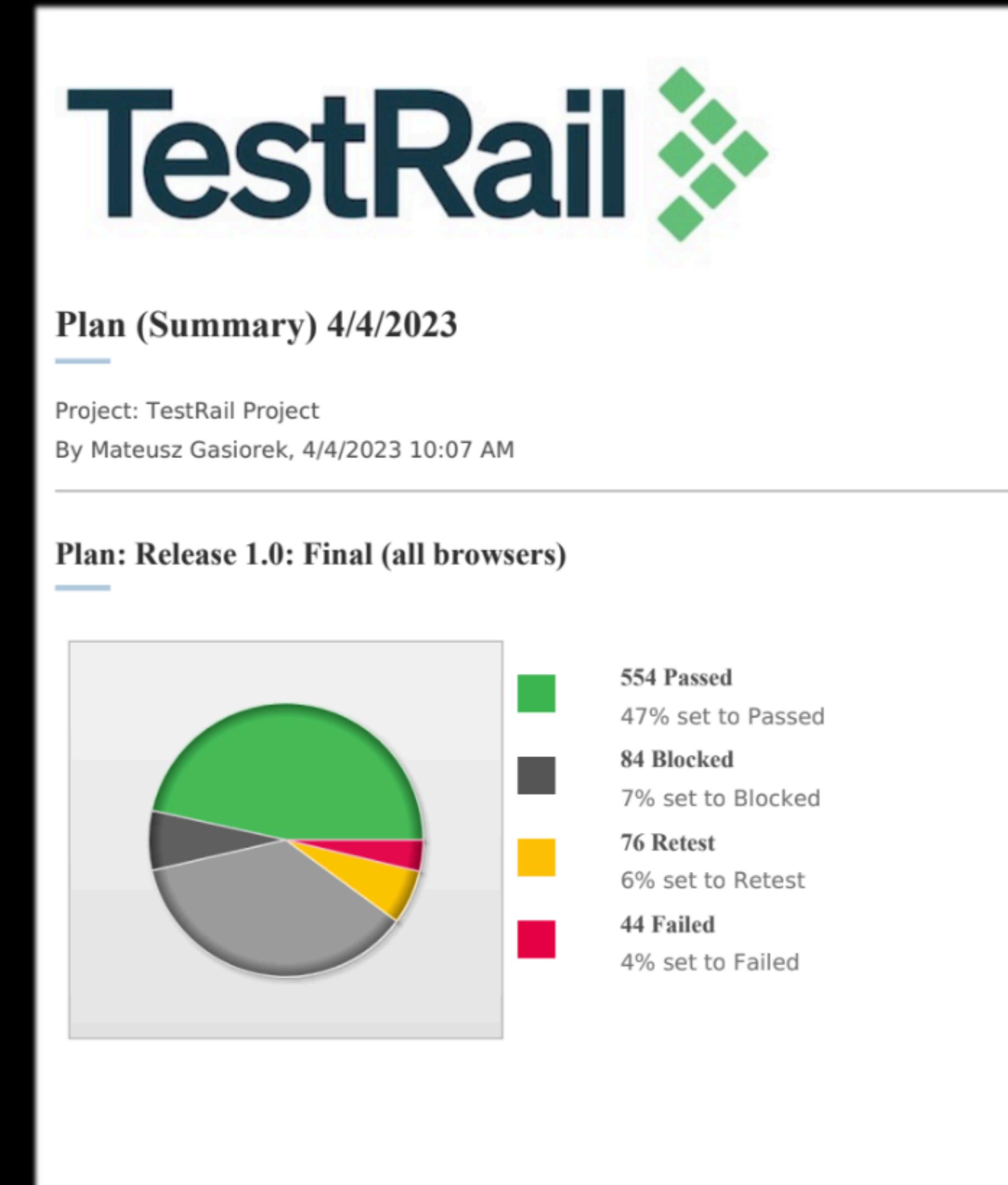
Find the code that has the most problems and test them

# Easy-First Strategy

Test Feasibility: Logical complexity, Dependency level  
and Priority

# Make Progress Visible

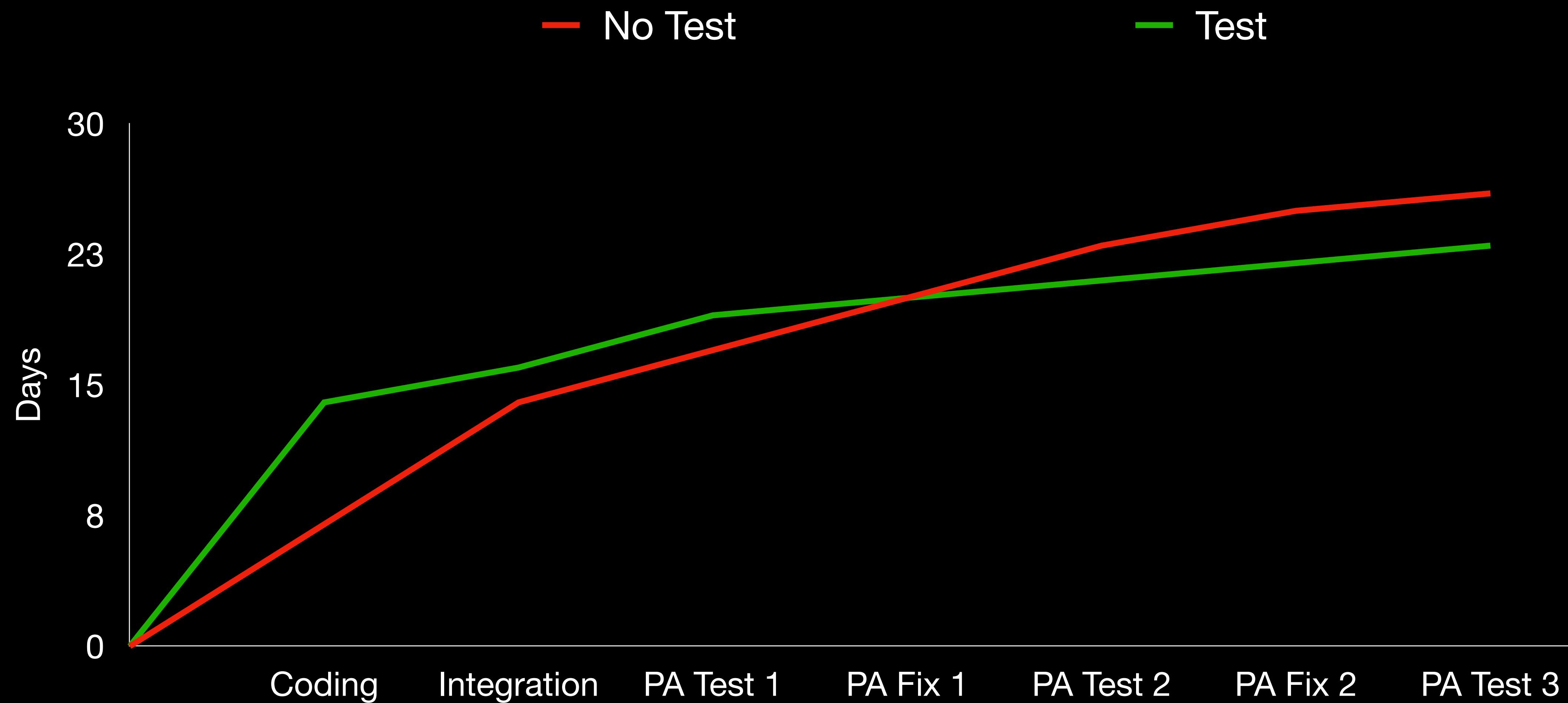
Code coverage is **NOT** quality



# Create metrics and follow the trends

- Bugs found in production
- Deployment frequency
- The amount of time it takes a feature request to get into production
- Time to restore service
- Engineering confidence

# Team progress with test and without test



# Bugs found in production



# Team progress with test and without test

Stage	No Test	Test
Coding	7 days	14 days
Integration	7 days	2 days
Testing and bug fixing	Testing, 3 days Fixing, 3 days Testing, 3 days Fixing, 2 days Testing, 1 day, Total 12 days	Testing, 3 days Fixing, 1 day Testing, 1 days Fixing, 1 day Testing, 1 day, Total, 7 days
Overall Release Time	26 days	23 days
Bugs found in production	71	11

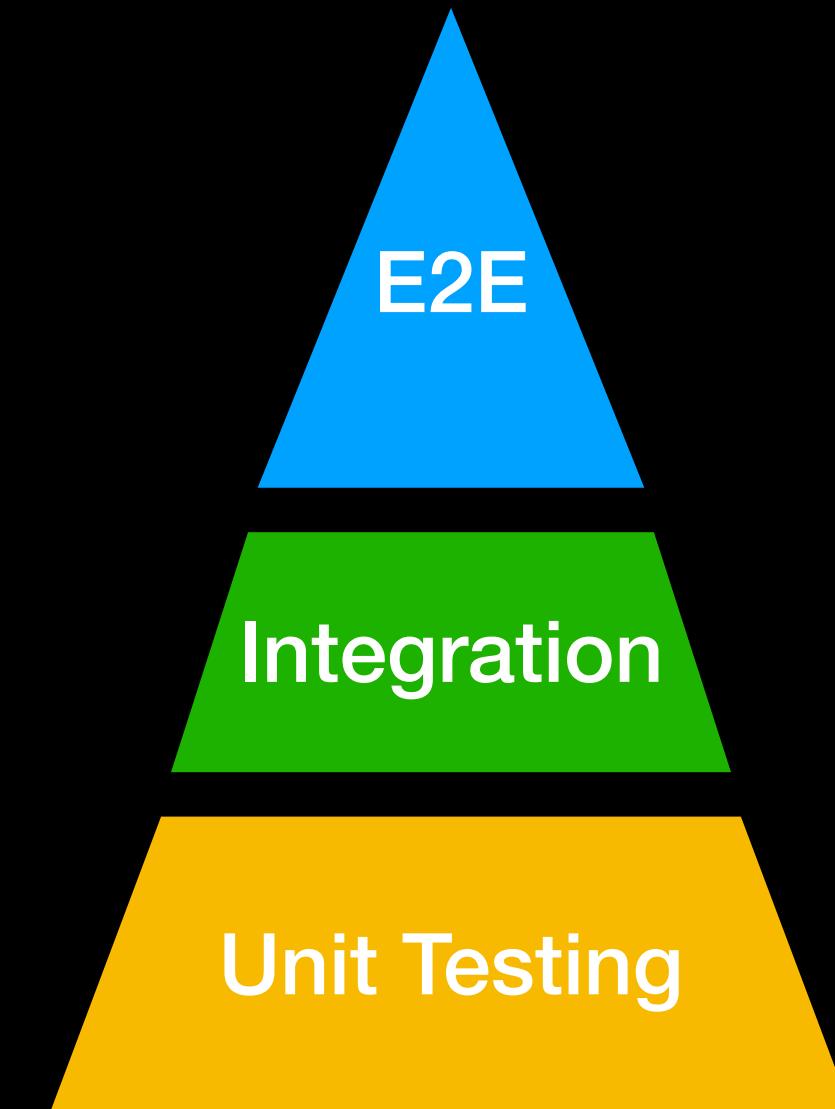
# Reshape Gogoro engineering culture.

- Develop a testing strategy with PA team
- Create metrics and follow the trends
- Add integration and e2e testing in sprint





## Playwright Codegen



- Readability
- Maintainability
- Trust
- Design and Process

**Shift Left**

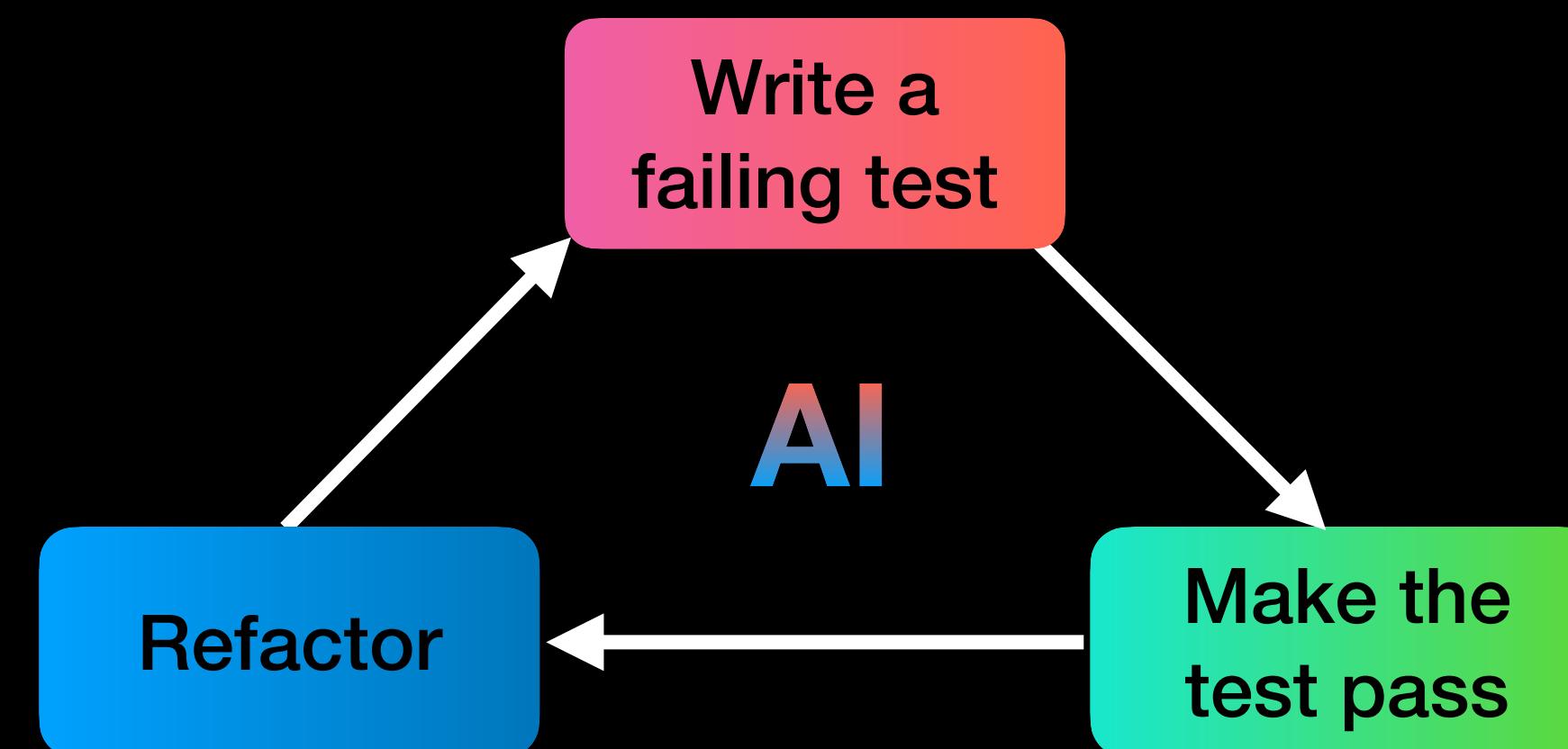
Cost

Design Dev QA Prod

**Visible Progress**

Bug

2025 2026 2027



# Reference

- 如何撰寫具彈性的測試程式 | 2024 WebConf Taiwan
- Component testing in Storybook
- Test Smells
- Software Engineering at Google, Testing Overview
- Google Testing Blog
- The Art of Unit Testing, Third Edition

# Q & A

Thank you for listening