

TDD



The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO



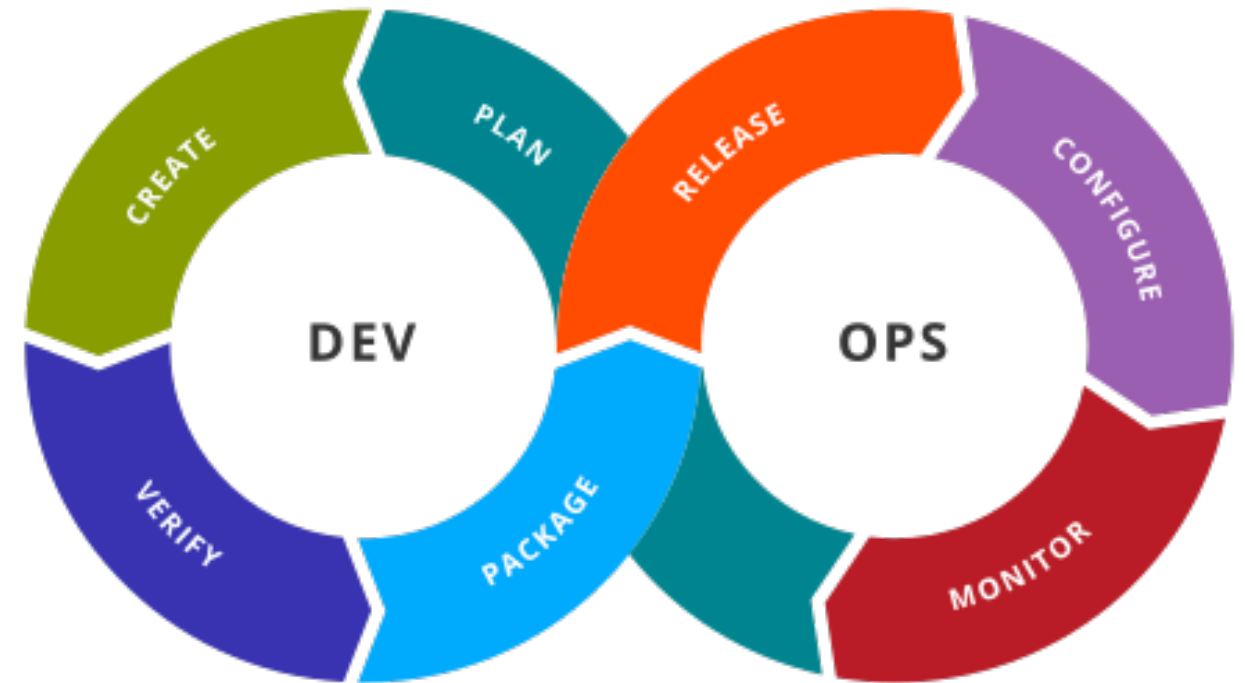
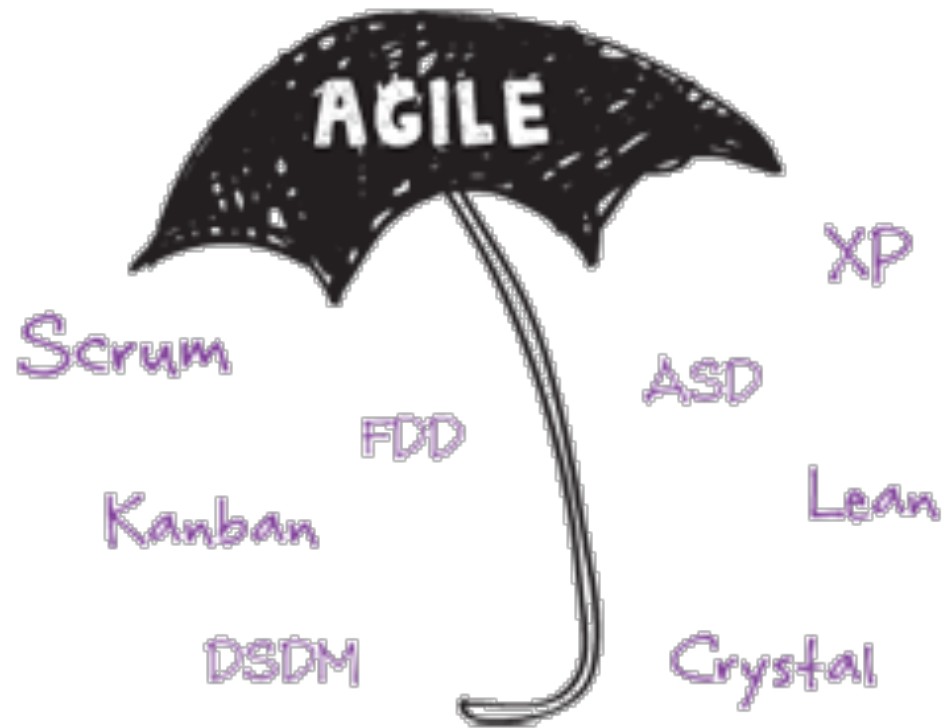
Early Testing

Testing should start
as early as possible in the
Software Development Life Cycle

Learning Goals

- Understand process aspects of QA
- Describe the tradeoffs of QA techniques
- Select an appropriate QA technique for a given project and quality attribute
- Apply testing and test automation for functional correctness
- Understand opportunities and challenges for testing quality attributes; enumerate testing strategies to help evaluate the following quality attributes: usability, reliability, security, robustness (both general and architectural), performance, integration.
- Discuss the limitations of testing

Developers + Operators = DevOps

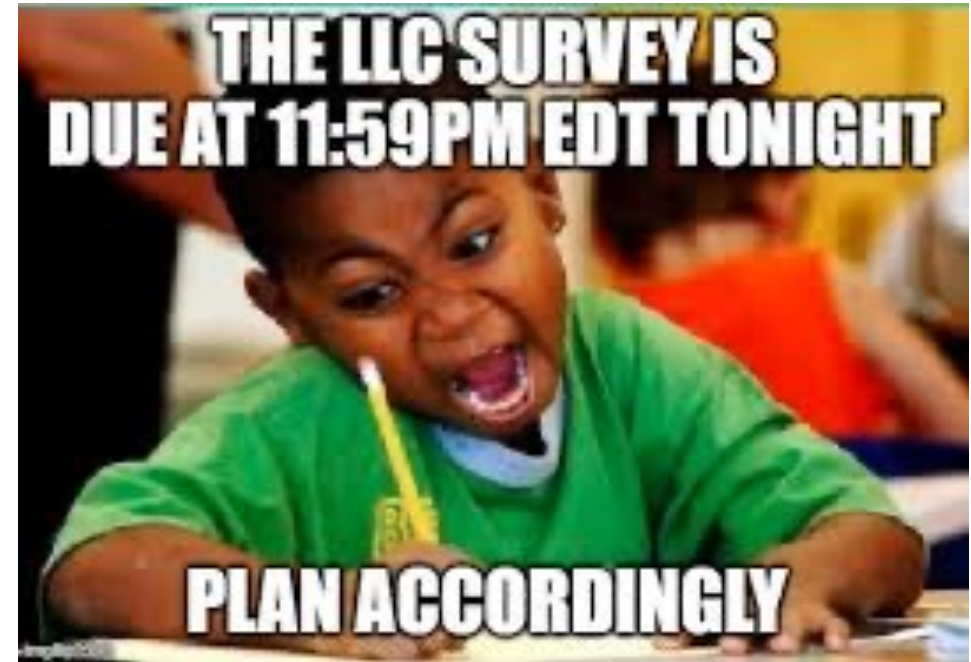


QA is Hard

“We had initially scheduled time to write tests for both front and back end systems, although this never happened.”



“Due to the lack of time, we could only conduct individual pages’ unit testing. Limited testing was done using use cases. Our team felt that this testing process was rushed and more time and effort should be allocated.”



“We failed completely to adhere to the initial [testing] plan. From the onset of the development process, we were more concerned with implementing the necessary features than the quality of our implementation, and as a result, we delayed, and eventually, failed to write any tests.”



Software testing life...

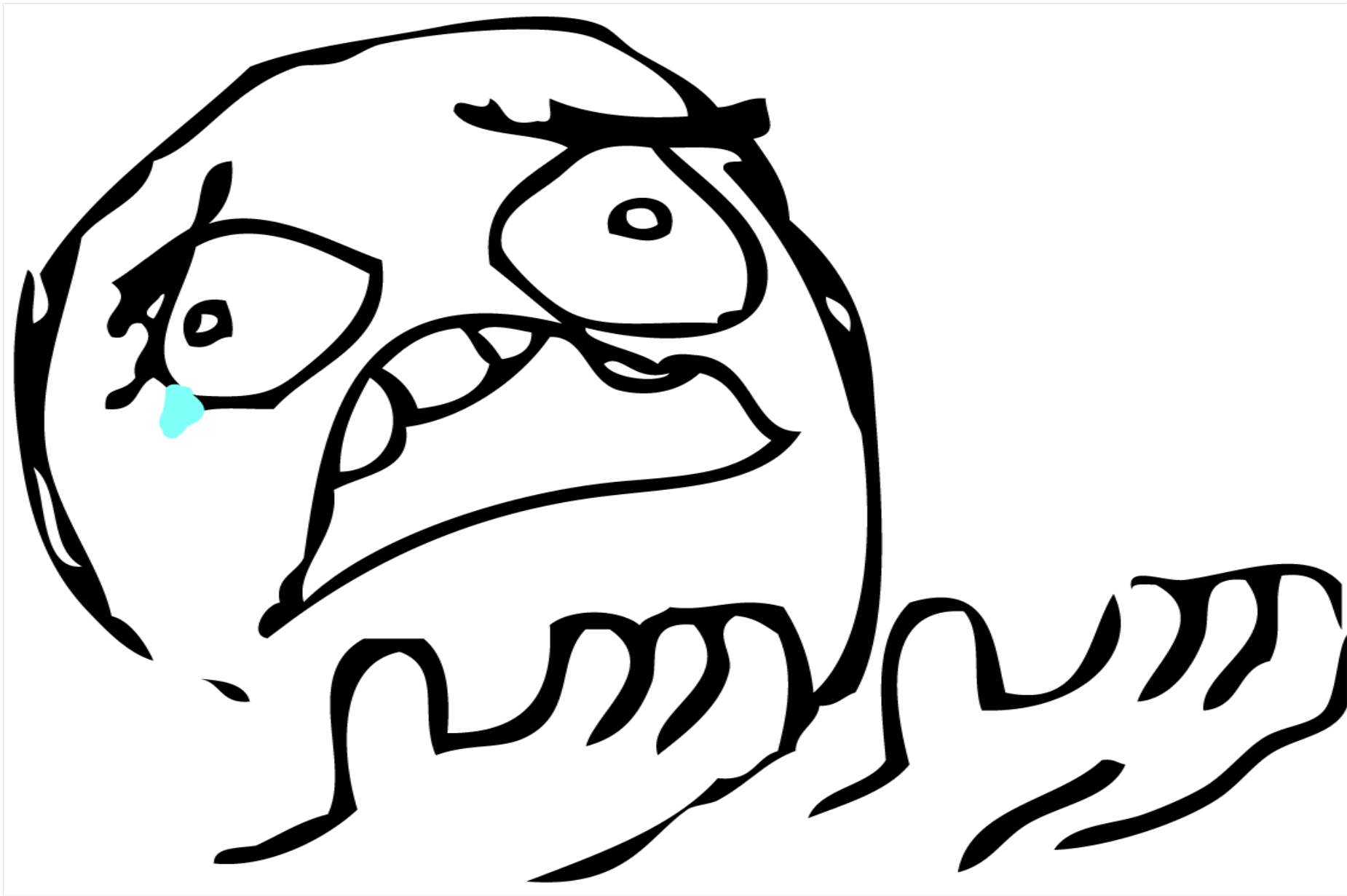
Unit test vs. Integration test



“One portion we planned for but were not able to complete to our satisfaction was testing.”

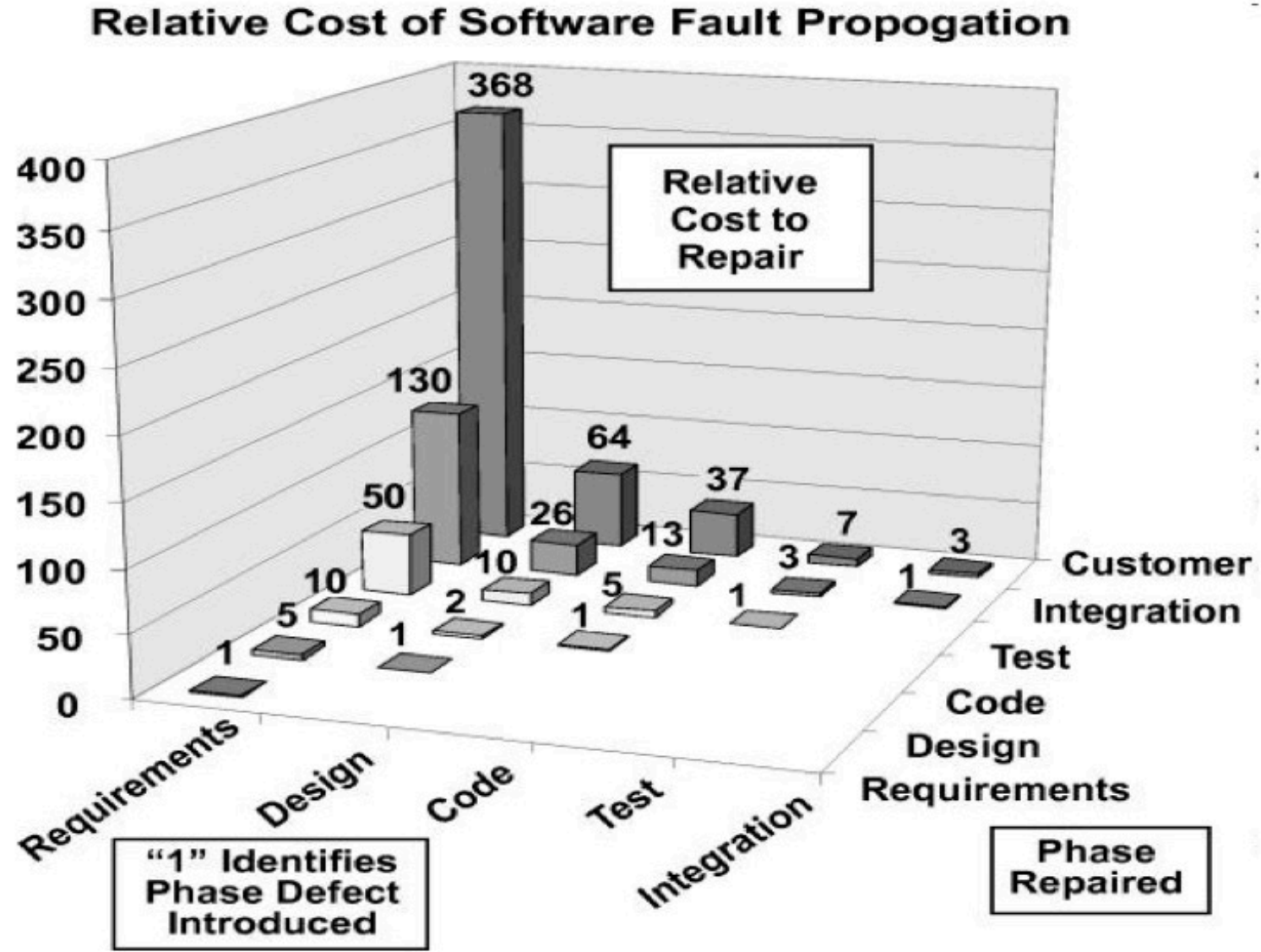
Time estimates (in hours):

Activity	Estimated	Actual
testing plans	3	0
unit testing	3	1
validation testing	4	2
test data	1	1



QA is Important (Duh!)

Cost



DEFECT: It can be simply defined as a variance between expected and actual.

Defect is an error found AFTER the application goes into production. It commonly refers to several troubles with the software products, with its external behavior or with its internal features.

In other words Defect is the difference between expected and actual result in the context of testing. It is the deviation of the customer requirement.

Wrong: When requirements are implemented not in the right way. This defect is a variance from the given specification. It is Wrong!

Missing: A requirement of the customer that was not fulfilled. This is a variance from the specifications, an indication that a specification was not implemented, or a requirement of the customer was not noted correctly.

Extra: A requirement incorporated into the product that was not given by the end customer. This is always a variance from the specification, but may be an attribute desired by the user of the product. However, it is considered a defect because it's a variance from the existing requirements.

ERROR: An error is a mistake, misconception, or misunderstanding on the part of a software developer. In the category of developer we include software engineers, programmers, analysts, and testers.

For example, a developer may misunderstand a de-sign notation, or a programmer might type a variable name incorrectly – leads to an Error. It is the one which is generated because of wrong login, loop or due to syntax. Error normally arises in software; it leads to change the functionality of the program.

BUG: A bug is the result of a coding error. An Error found in the development environment before the product is shipped to the customer. A programming error that causes a program to work poorly, produce incorrect results or crash. An error in software or hardware that causes a program to malfunction. Bug is terminology of Tester.

FAILURE: A failure is the inability of a software system or component to perform its required functions within specified performance requirements. When a defect reaches the end customer it is called a Failure. During development Failures are usually observed by testers.

FAULT: An incorrect step, process or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner. A fault is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification. It is the result of the error.

The software industry can still not agree on the definitions for all the above. In essence, if you use the term to mean one specific thing, it may not be understood to be that thing by your audience.

Cost

theguardian

News | US | World | Sports | Comment | Culture | Business | Money | Environment | Science

News > Technology > Heartbleed

Heartbleed: developer who introduced the error regrets 'oversight'

Submitted just seconds before new year in 2012, the bug 'slipped through' – but discovery 'validates' open source

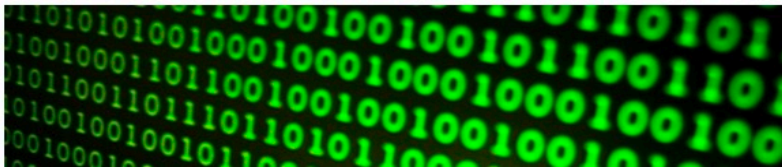
Alex Hern

Follow @alexhern

Follow @guardiantech

theguardian.com, Friday 11 April 2014 03.05 EDT

Jump to comments (108)



Share 430

Tweet 269

+1 27

Share 103

Email



Technology

Heartbleed · Open source · Programming · Software · Internet · Hacking · Data and computer security

More news

More on this story

Heartbleed bug 'will cost millions'

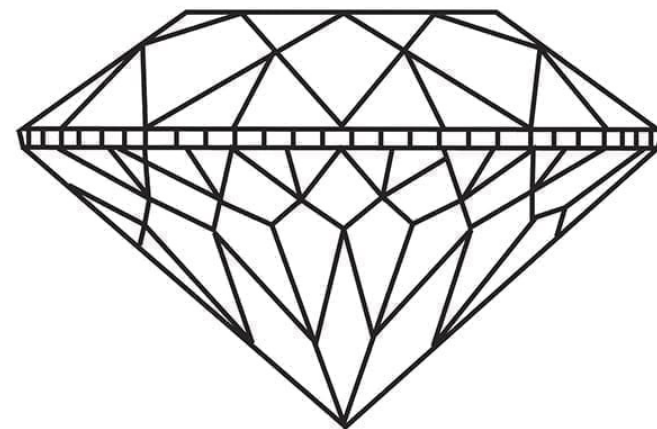
Revoking all SSL certificates leaked by Heartbleed will cost millions of dollars, according to Cloudflare, which provides services to website hosts



▲ Image: Codenomicon



**WHAT IS
HEARTBLEED?**



QA has many facets

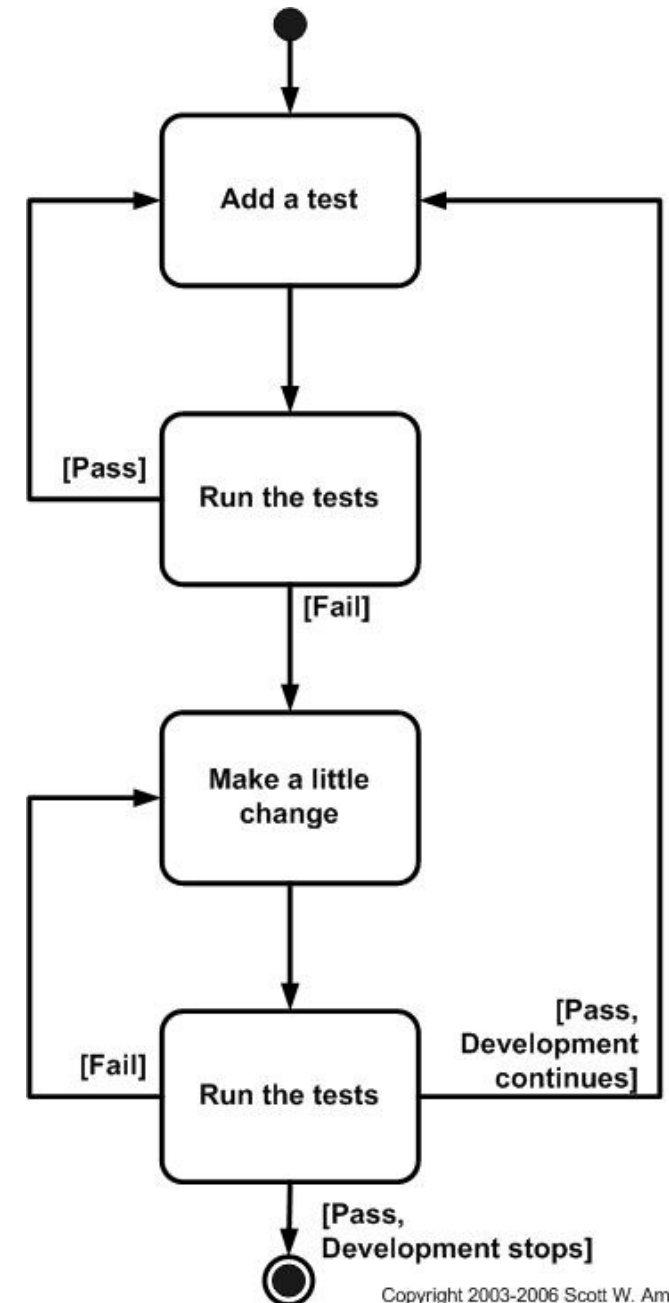
How do you know that your
Program works?

Questions

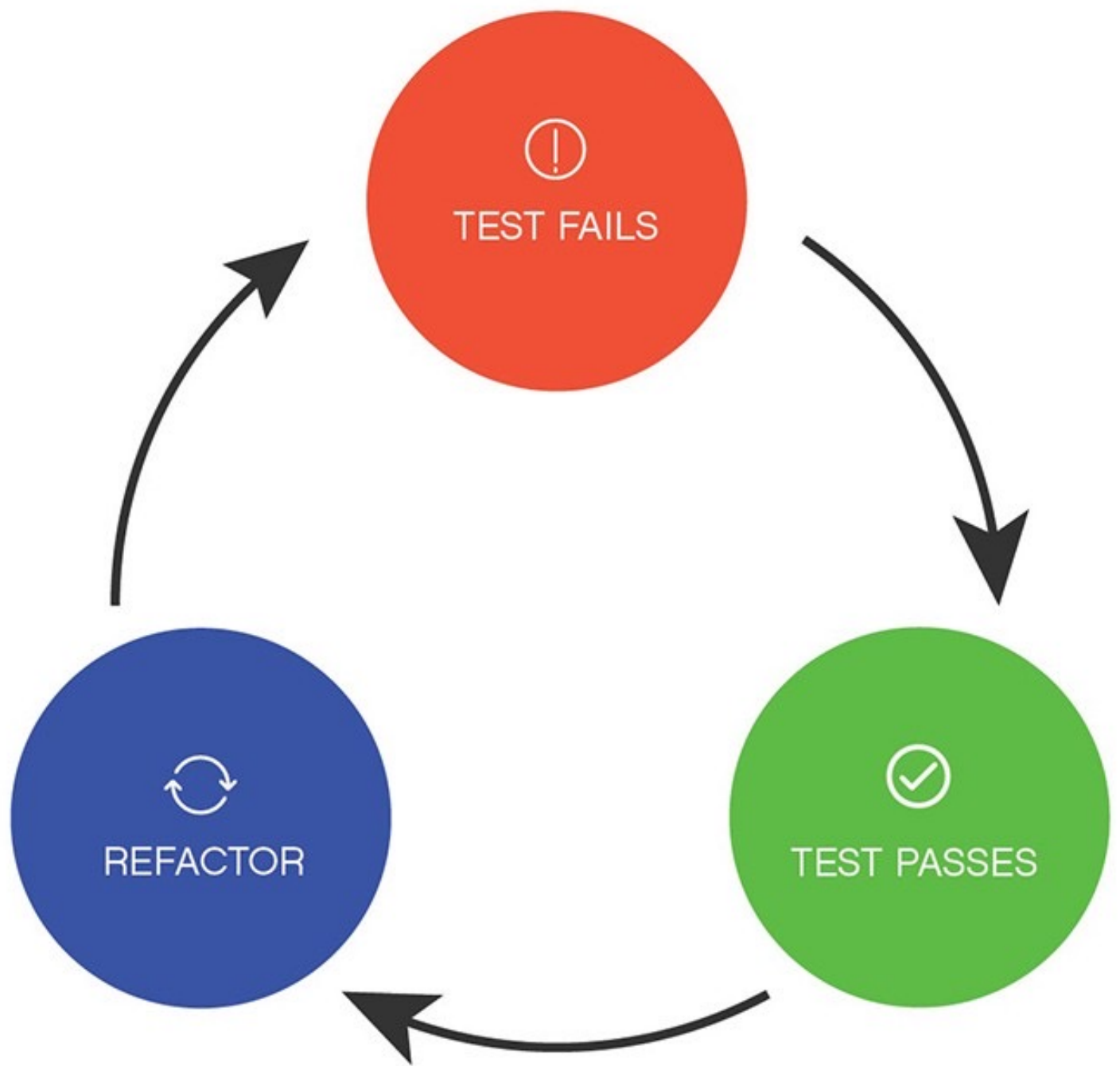
- How can we ensure that the specifications are correct?
- How can we ensure a system meets its specification?
- How can we ensure a system meets the needs of its users?
- How can we ensure a system does not behave badly?

Test Driven Development (TDD)





- Test-driven development (TDD) is an evolutionary approach to development which combines **test-first development**, where you write a test before you write just enough production code to fulfil that test, and **refactoring**.



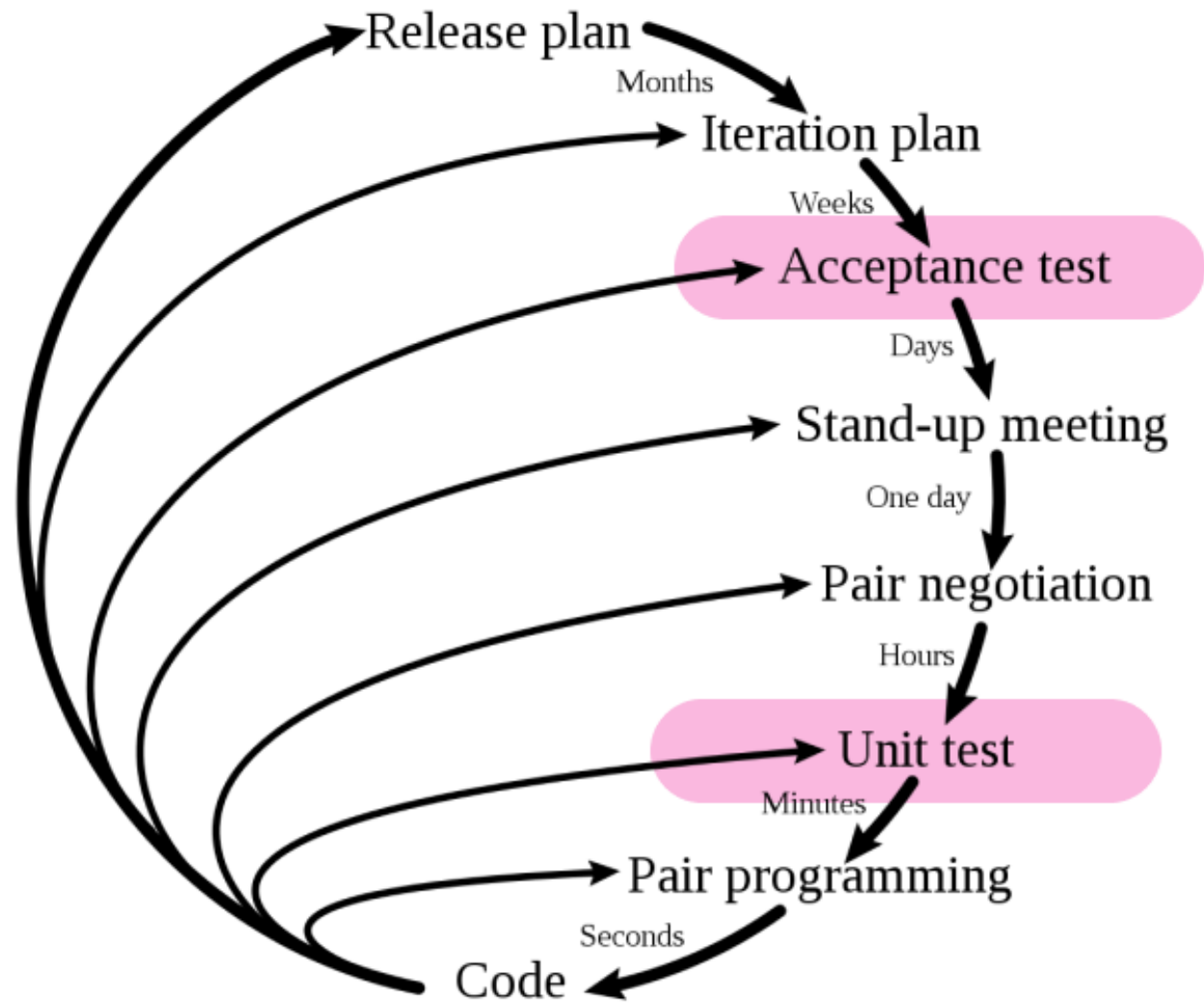
TDD Cycle



Types of tests (a subset)

- Unit  — Test an individual, isolated component
- Integration  — Test that multiple units work together
- End-to-End  — Tests that act as a user *actually using the application*
- Acceptance  — verify a user story works as expected.

Planning/feedback loops

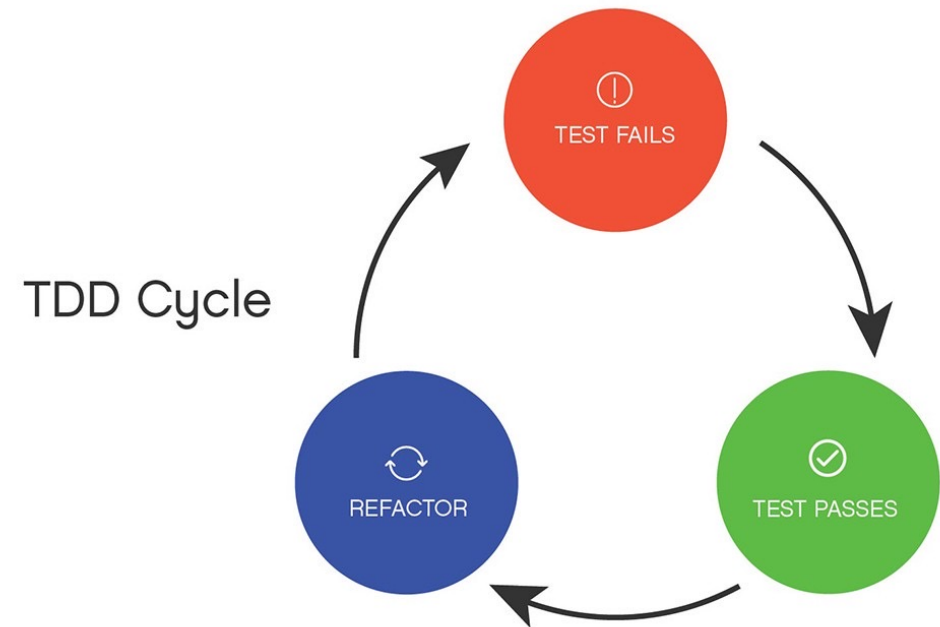


- In Extreme Programming, tests are a mandatory part of planning and feedback loops

TDD demo: Palindrome Example

- **Functional requirement:** detect that a string is a palindrome: that is, it is the same word or phrase in reverse.
 - mom
 - Mom
 - Was It A Rat I Saw
 - Never Odd or Even
 - ...

<https://khalilstemmler.com/articles/test-driven-development/introduction-to-tdd/>



TDD demo: Palindrome Example

- **1. Write the failing test**

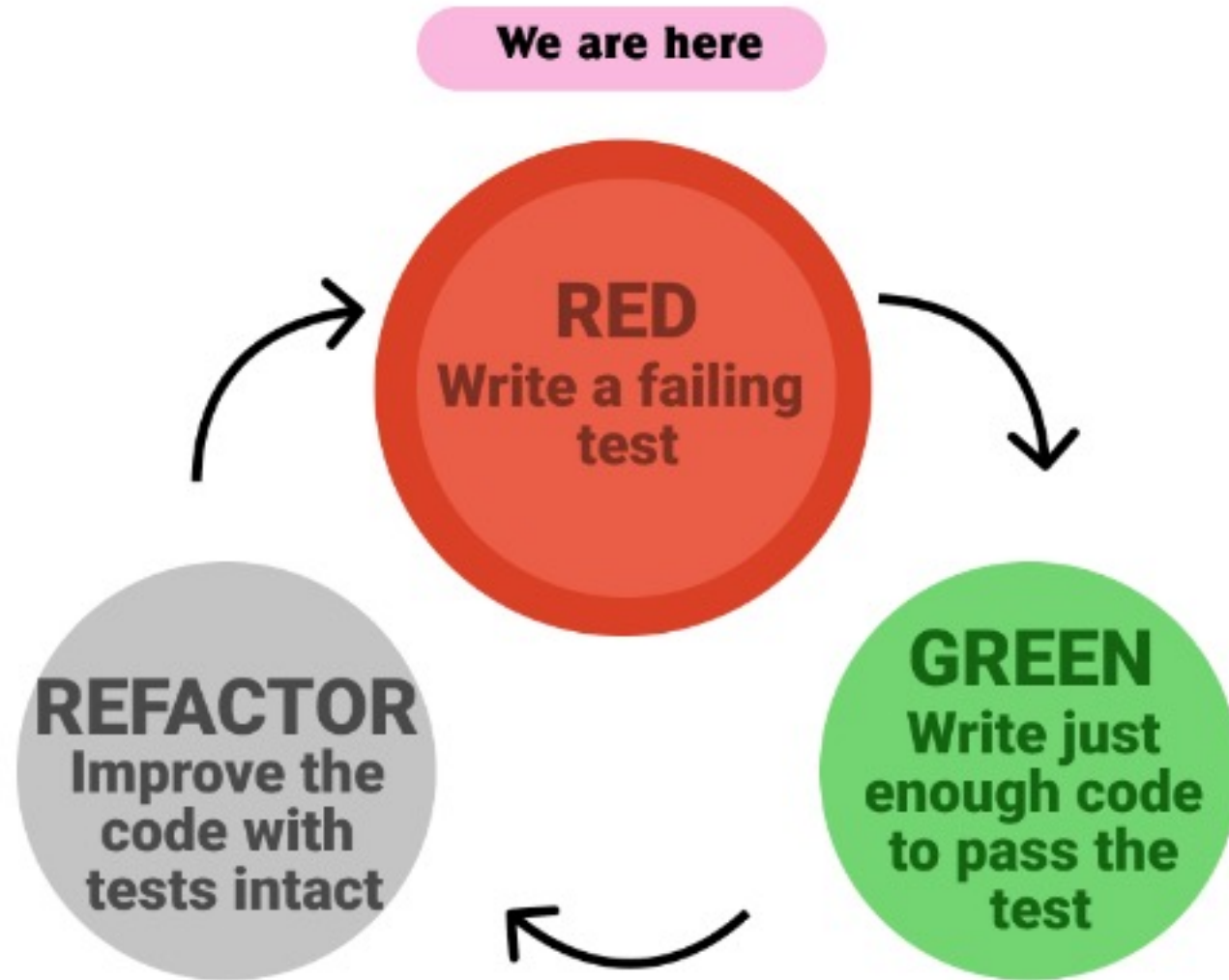
- Write the test name using the requirements
- Pretend that something called `palindromeChecker` exists and that it has an `isAPalindrome` method on it.
- Expect the method to return `true` for `mom` .
- Save

TDD demo: Palindrome Example

- 1. Write the failing test

```
index.spec.ts
```

```
describe('palindrome checker', () => {  
  
  it('should be able to tell that "mom" is a palindrome', () => {  
    expect(palindromeChecker.isAPalindrome('mom')).toBeTruthy(); // ✖  
  });  
  
})
```



TDD demo: Palindrome Example

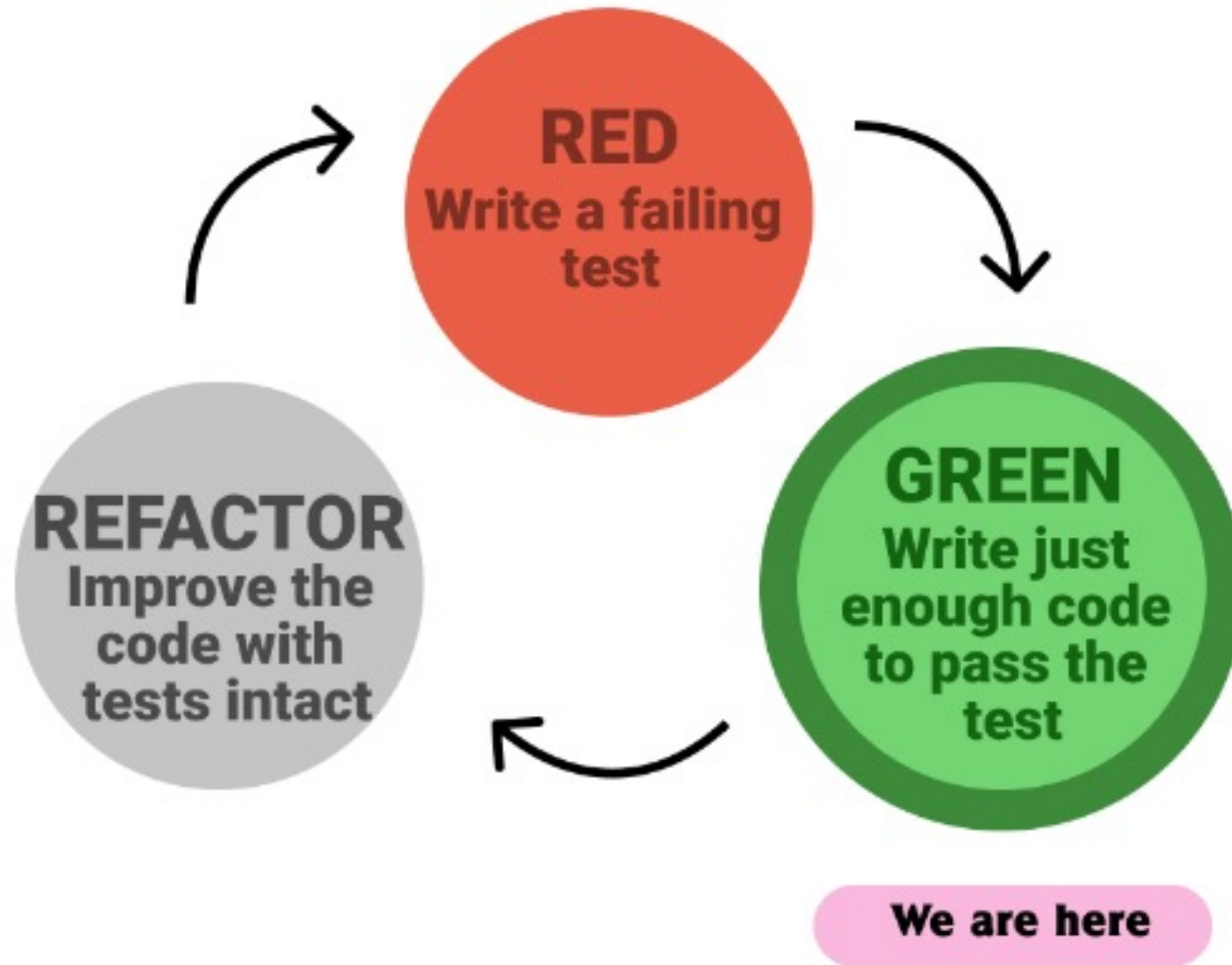
- **2. Write the simplest code to make the test pass**

- - Create a `PalindromeChecker` class
 - Give it an `isAPalindrome` method
 - Return `true` (the simplest thing that would work)
 - and import it in our test

TDD demo: Palindrome Example

- 2. Write the simplest code to make the test pass

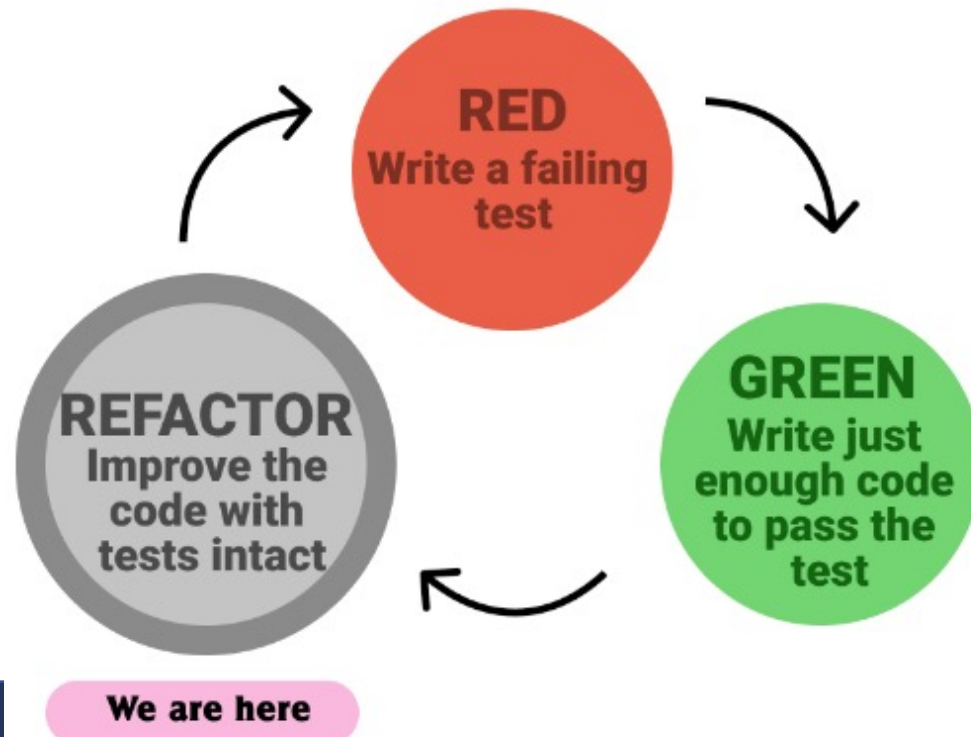
- ```
export class PalindromeChecker {
 isAPalindrome (str: string): boolean {
 return true; // This is the simplest thing!
 }
}
```



# TDD demo: Palindrome Example

- **3. Refactor**

When refactoring, keep a lookout for **duplication (at least three times)** and **code smells**.







# TDD demo: Palindrome Example

- 4. The next failing test -- “test that "bill" *isn't* a palindrome.”

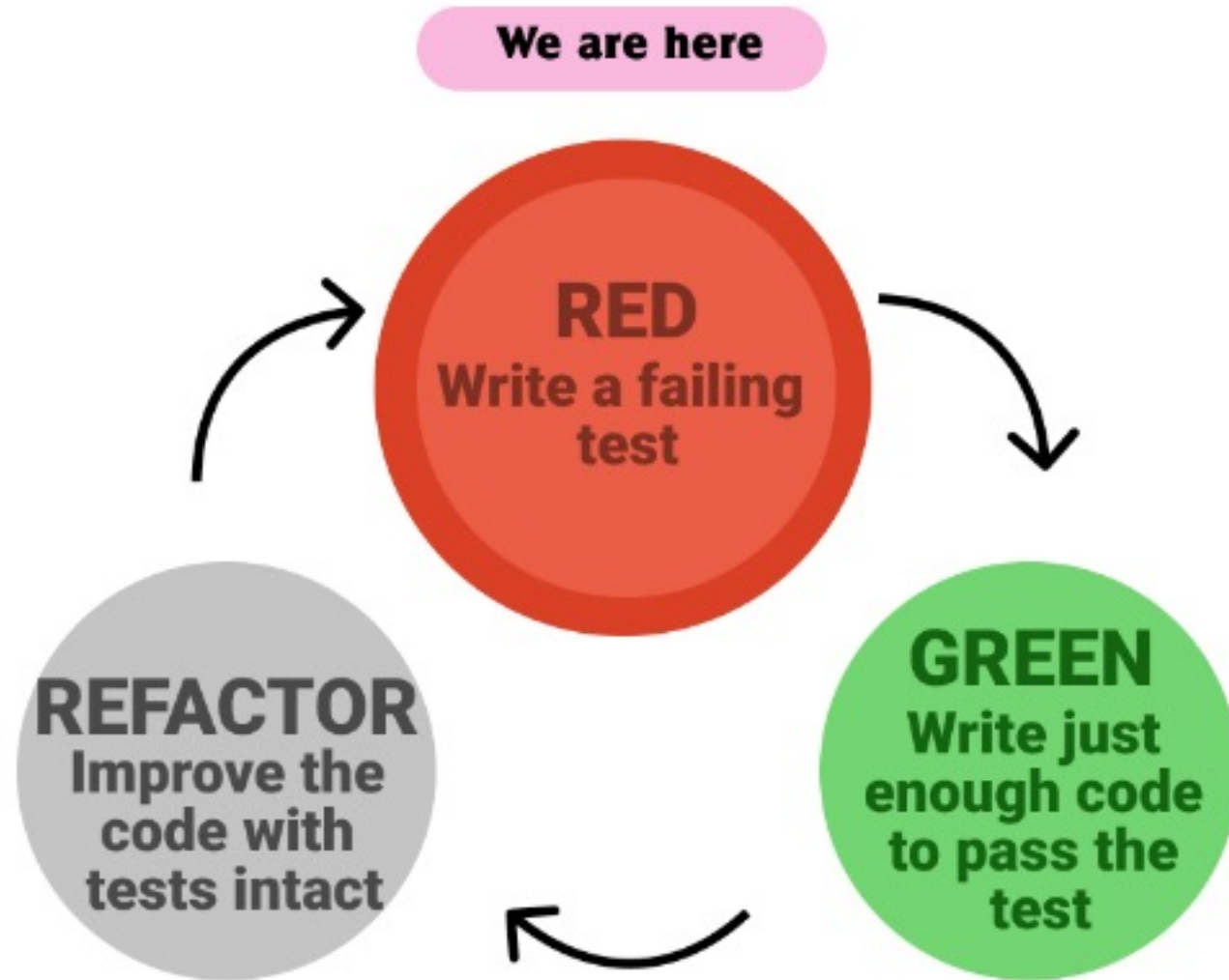
```
import { PalindromeChecker } from './index'

describe('palindrome checker', () => {

 it('should be able to tell that "mom" is a palindrome', () => {
 const palindromeChecker = new PalindromeChecker();
 expect(palindromeChecker.isAPalindrome('mom')).toBeTruthy(); // 
 });

 it('should be able to tell that "bill" isnt a palindrome', () => {
 const palindromeChecker = new PalindromeChecker();
 expect(palindromeChecker.isAPalindrome('bill')).toBeFalsy(); // 
 });
});
```



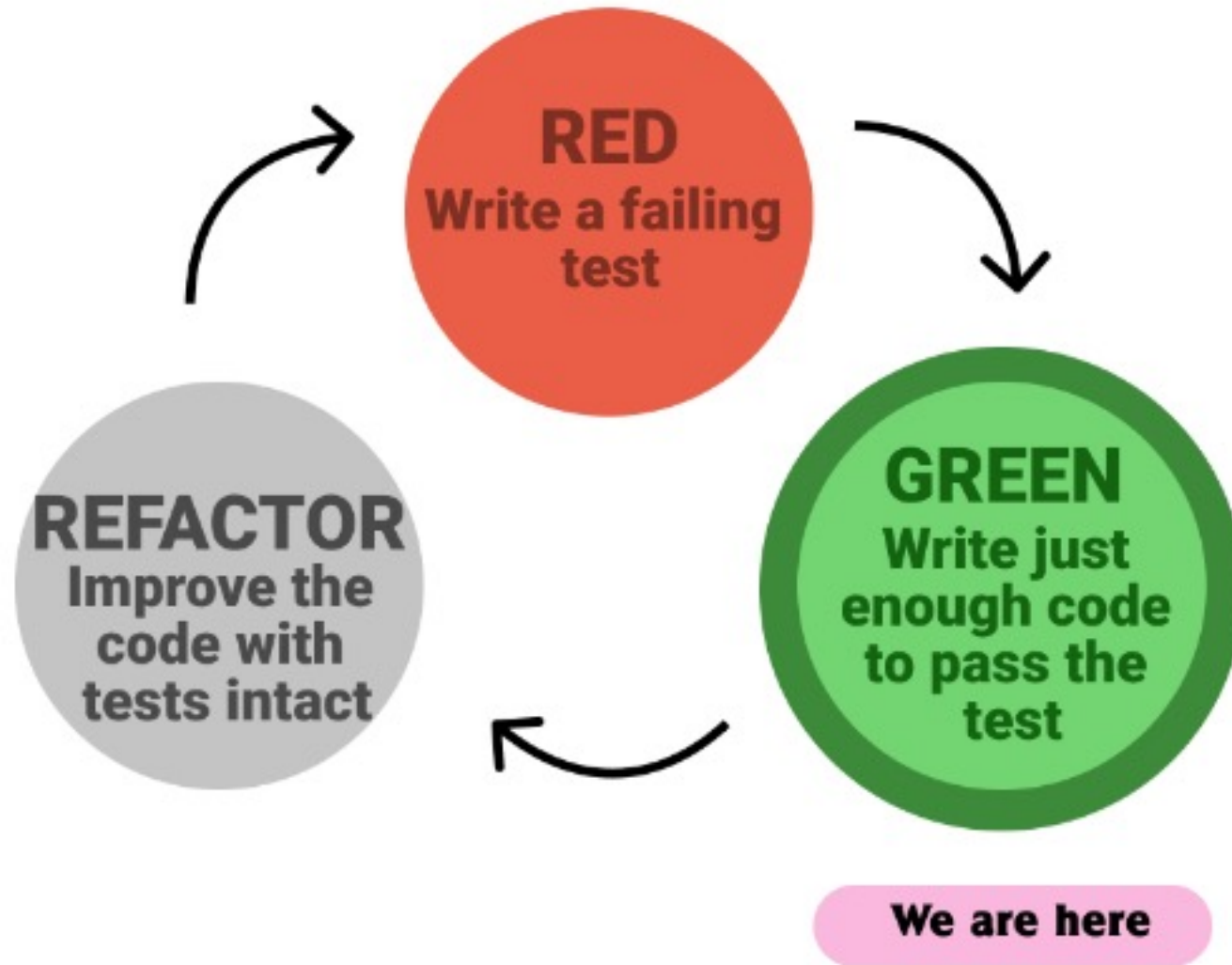


# TDD demo: Palindrome Example

- 5. Write the simplest code to make the test pass

```
export class PalindromeChecker {
 isAPalindrome (str: string): boolean {
 if (str === 'mom') {
 return true;
 } else {
 return false;
 }
 }
}
```

```
export class PalindromeChecker {
 isAPalindrome (str: string): boolean {
 const reversed = str.split("").reverse().join("");
 return reversed === str;
 }
}
```



# TDD demo: Palindrome Example

- **6. Refactor** -- refactoring both production code and test code

```
import { PalindromeChecker } from './index'

describe('palindrome checker', () => {

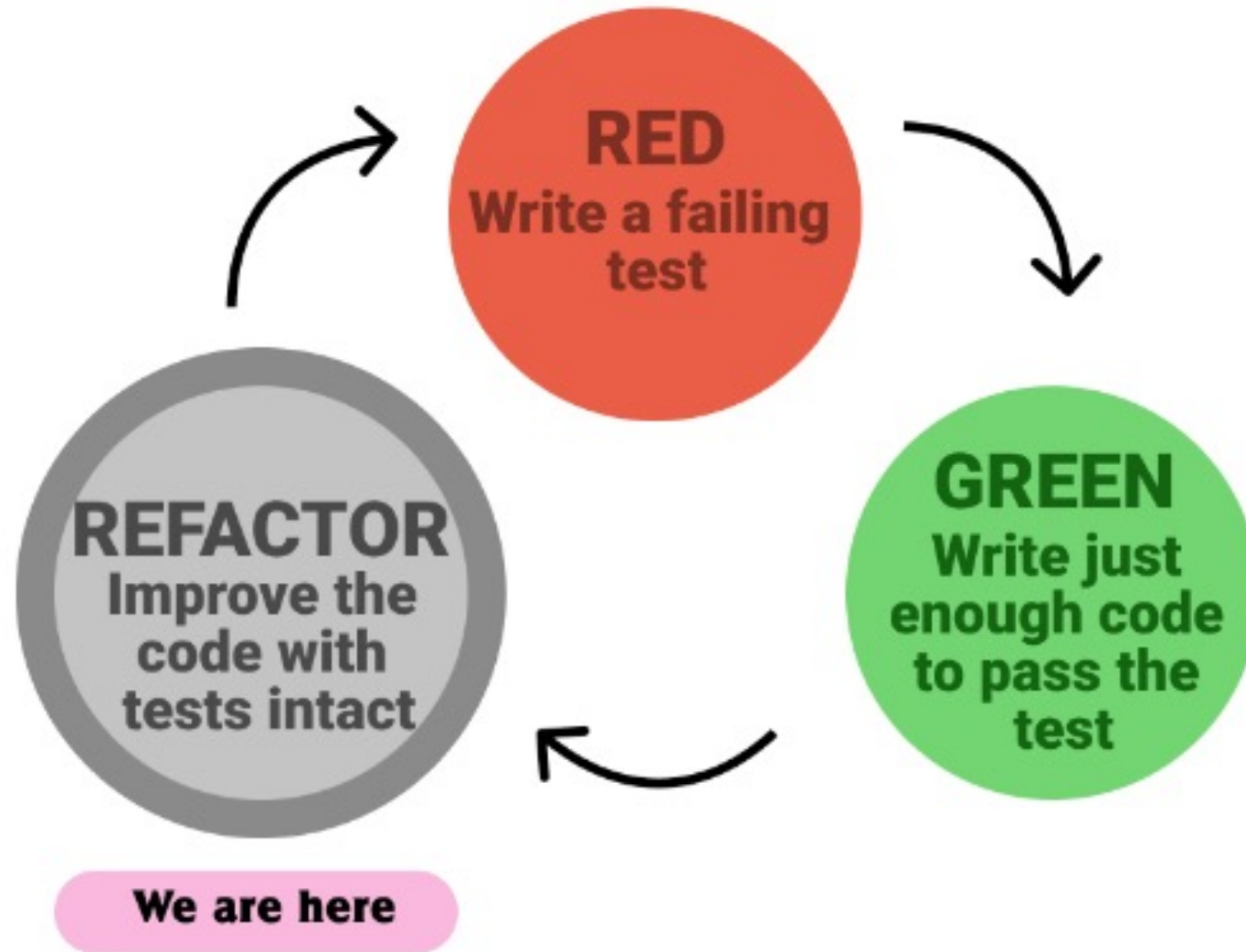
 it('should be able to tell that "mom" is a palindrome', () => {
 const palindromeChecker = new PalindromeChecker();
 expect(palindromeChecker.isAPalindrome('mom')).toBeTruthy(); // ✓
 });

 it('should be able to tell that "bill" isnt a palindrome', () => {
 const palindromeChecker = new PalindromeChecker();
 expect(palindromeChecker.isAPalindrome('bill')).toBeFalsy(); // ✓
 });
});
```

# TDD demo: Palindrome Example

- 4. The next failing test -- “test that “Mom” is a palindrome.”

```
it('should still detect a palindrome even if the casing is off', () => {
 const palindromeChecker = new PalindromeChecker();
 expect(palindromeChecker.isAPalindrome("Mom")).toBeTruthy(); // ✖
});
})
```



# Lab 6: TDD

**Stay  
Tuned!**