Software Testing

A (brief) introduction

THOU SHALT TEST

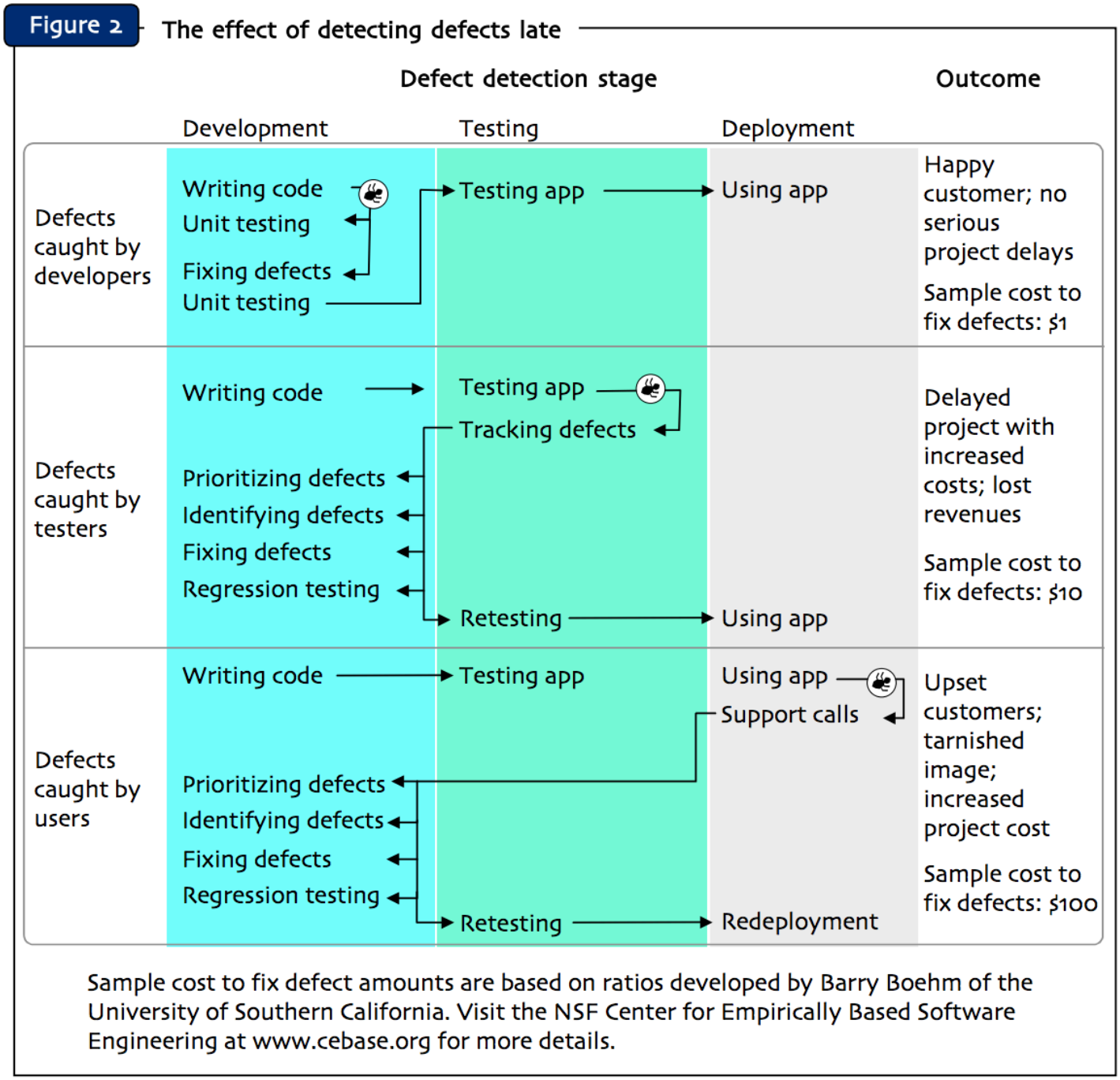
WHY?

Defective Software

- We develop programs that contain defects
  - How many? What kind?
- Hard to predict the future, however… it is highly likely, that the software we (including you!) will develop in the future will not be significantly better.

Defects cause failures

- **Communications:** Loss or corruption of communication media, non delivery of data.
- **Space Applications:** Lost lives, launch delays.
- **Defense and Warfare:** Misidentification of friend or foe.



Credits: “Improving Software By Reducing Coding Defects”, Klockwork Inc. (2004)

Advantages

- Testing accelerates development
- Testing assures quality of software

Debugging !== Testing

Do it right

- Testing needs to be planned
- Can easily cost 40% of your budget
- Shall be “reasonable”, independent and objective
- **Testing is a fundamental aspect of professional software development**

A definition of Testing

The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.

—ISTQB®

Validation and Verification

- **Validation**: are we building the right product?
- **Verification:** are we building the product right?

Static Testing

- Objective: find defects
- Verification only
- Code is not executed

Static analysis

- Examples
  - Syntax checking (your editor)
  - Code smell detection (e.g.: reek)
  - Coding standards compliance (e.g.: Rubocop)
- Typical defects found
  - Syntax errors
  - Unreachable code
  - Overly complicated constructs

Review techniques

- Informal review
- Walkthrough (train the team)
- Technical review (documented & defined process)
- Formal inspection (to gain metrics)

Success Factors

- Clear predefined objectives (what is important and why)
- Defects found are welcomed and presented objectively
- Application of appropriate review techniques
- Review process supported by management
- Emphasis on learning and process improvement

Coding standards

- Increase readability and maintainability
- Enforce a style to try and prevent bugs
- Industry standards + Team standards
- Static analysis to check compliance
- Example: AirBnB style guides http://airbnb.io/projects/styleguides/

Dynamic Testing

- Primarily verification
- The code is executed
- Objective: find failures
- Different methods: black/white box
- Different levels

Black-box vs White-box

- Black-box
  - Based on specifications
  - No knowledge of the code
  - Observation of external behavior
- White-box
  - Structure-based
  - Close examination of procedural level of detail
  - Knowledge of internals required

Unit Testing

- Test individual units (usually functions) in isolation
- Find defects in software components
- Done by developers
- In general, white-box

Ruby example

```
expect(factorial(10)).to

eq(1*2*3*4*5*6*7*8*9*10)
```

Make sure a method does what it's supposed to do (in isolation from the rest of the code)

Integration Testing

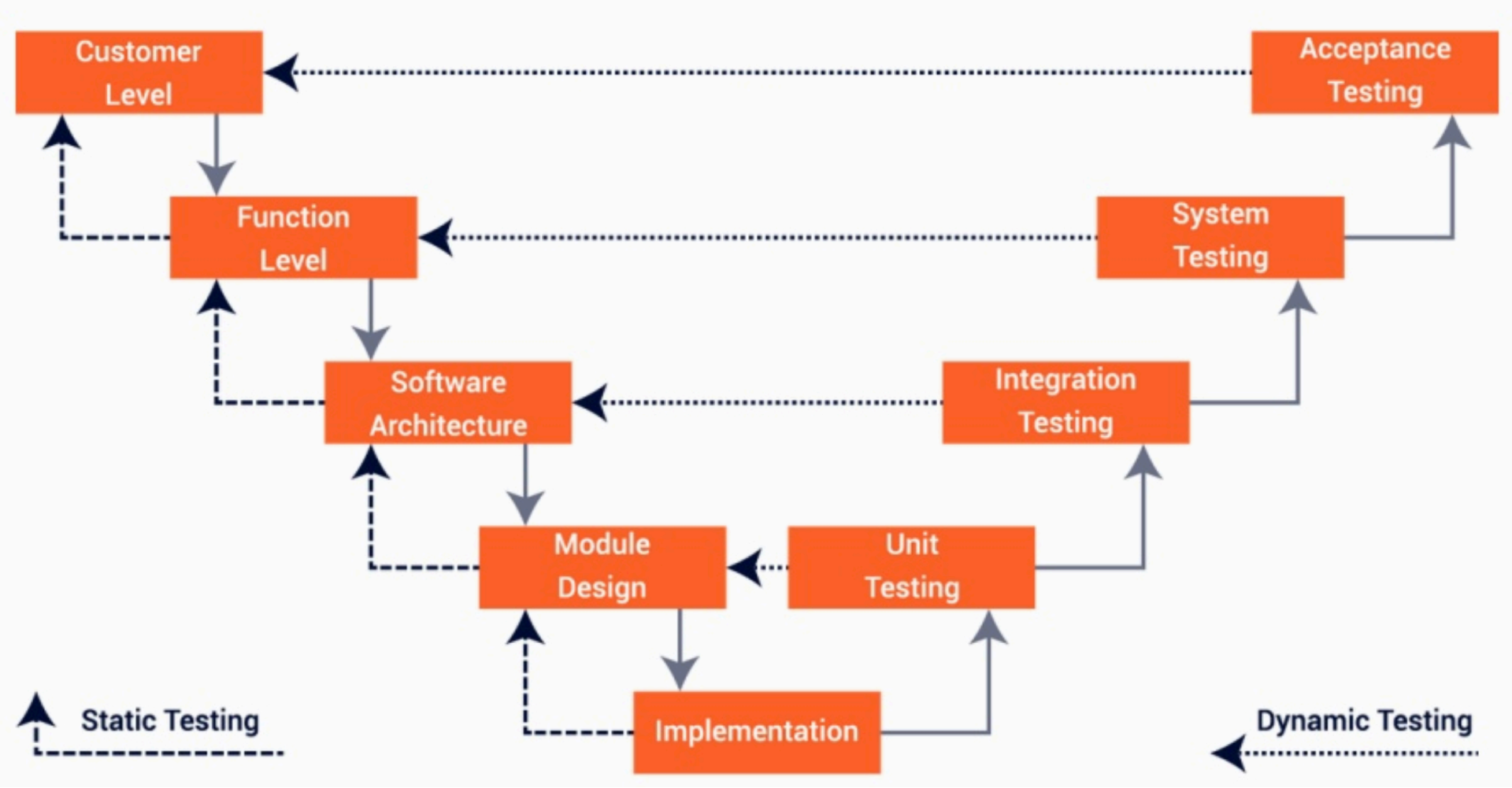
- Test communication/interaction among units
- Unit integration and system integration
- Done by developers
- Mainly white-box but also black-box

System Testing

- Test the complete system
- Evaluate compliance with requirements
- Stress, performance, usability, etc. testing
- Done by (external) testers (involving the customer in Agile processes)
- In general, black-box

Acceptance Testing

- Test complete, integrated system
- Evaluate compliance with acceptance criteria (e.g. critieria set for each user story)
- May be performed at various times (e.g. in SCRUM at each sprint for each US)
- Done by customers, users, product owners, …
- Only black-box

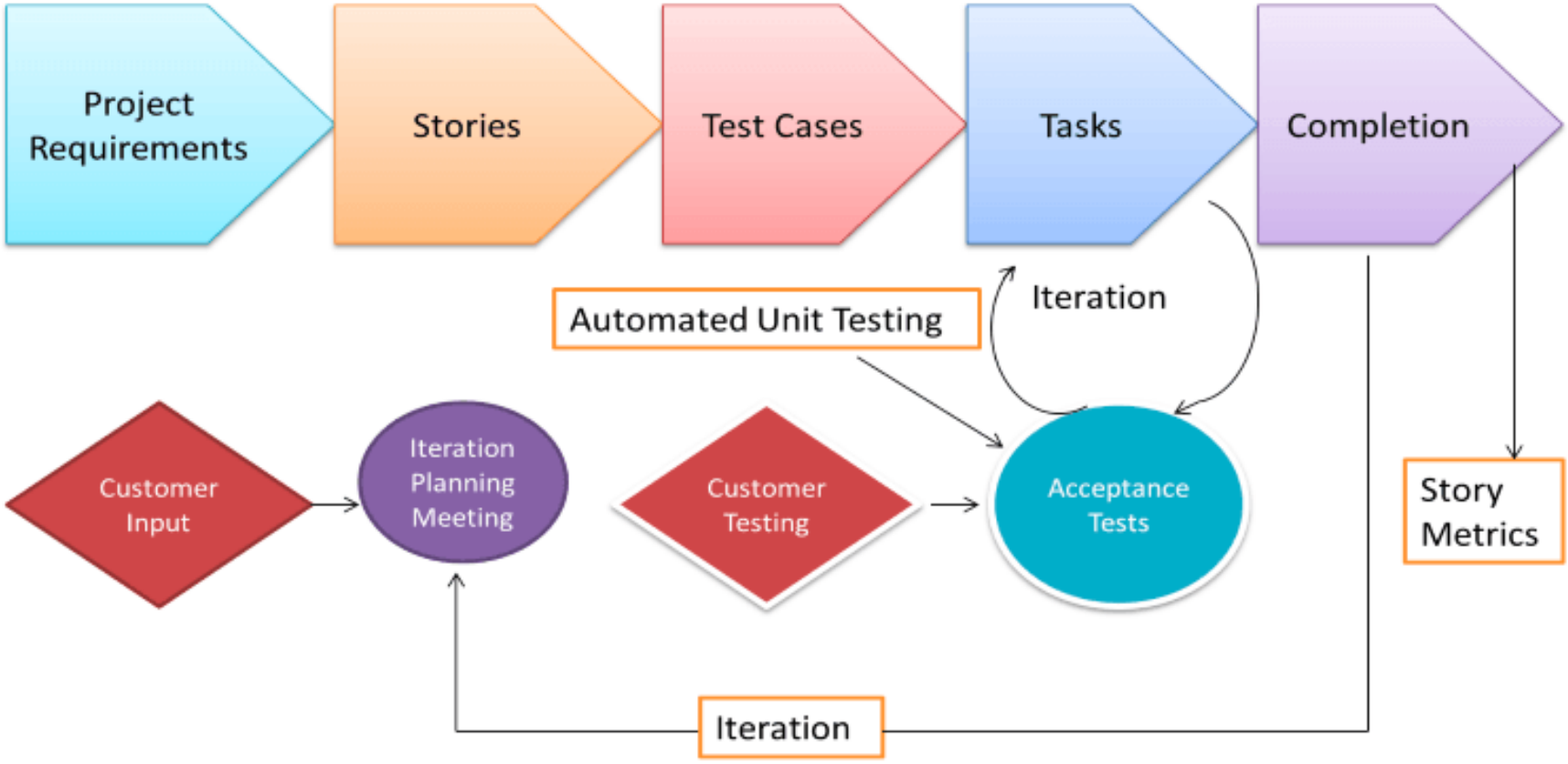

Customer Level
Acceptance Testing
Function Level
System Testing
Software Architecture
Integration Testing
Module Design
Unit Testing
Implementation
Static Testing
Dynamic Testing

Regression Testing

- Problem
  - You implemented some features
  - Then you implemented some more
  - …but you end up breaking the old features!
- Purpose: ensure that new features have not introduced new faults

A simple example

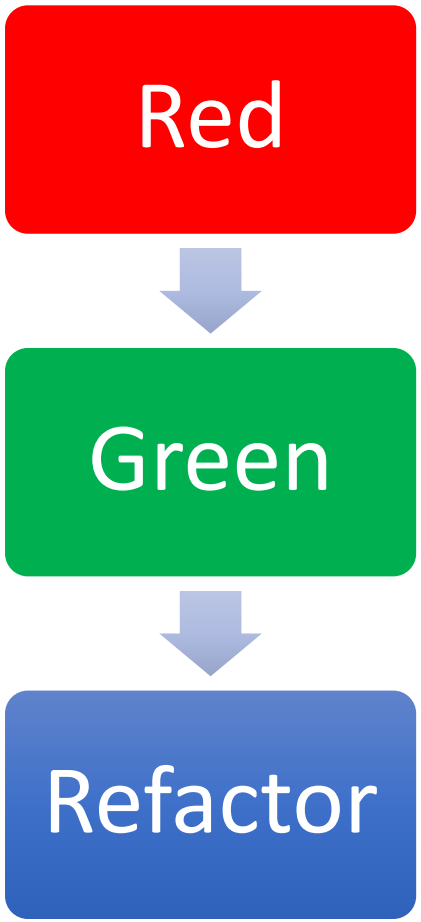
Agile Testing

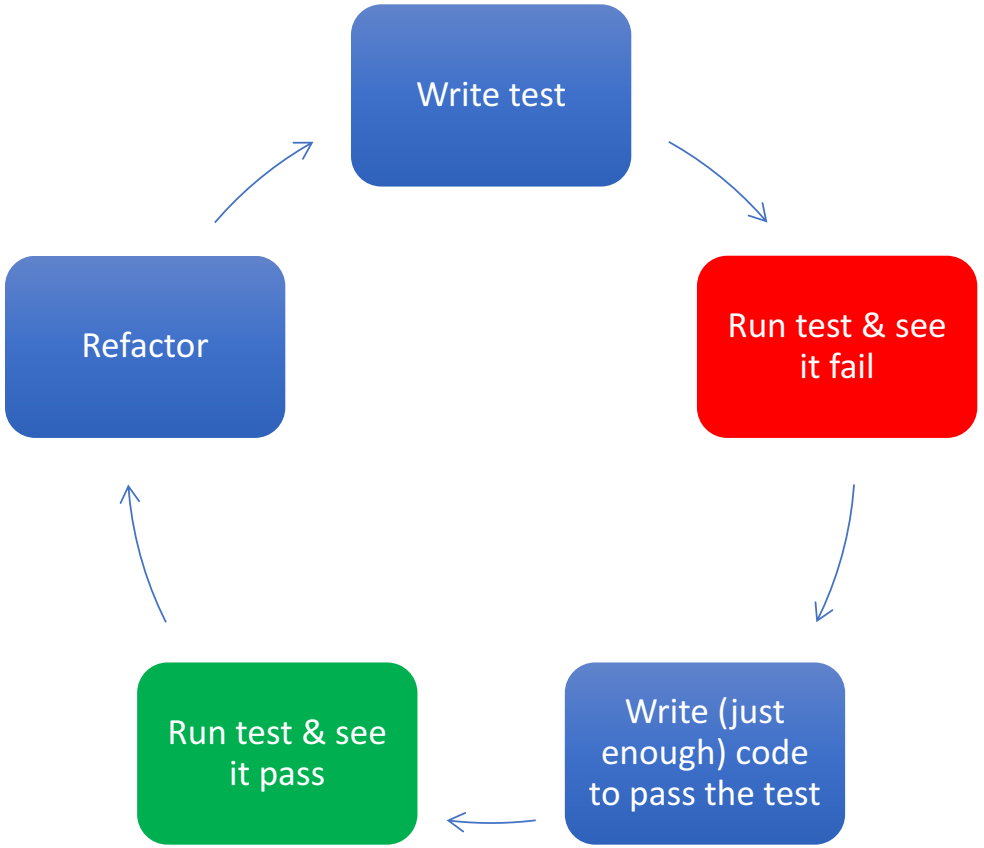


Test-Driven Development (TDD)

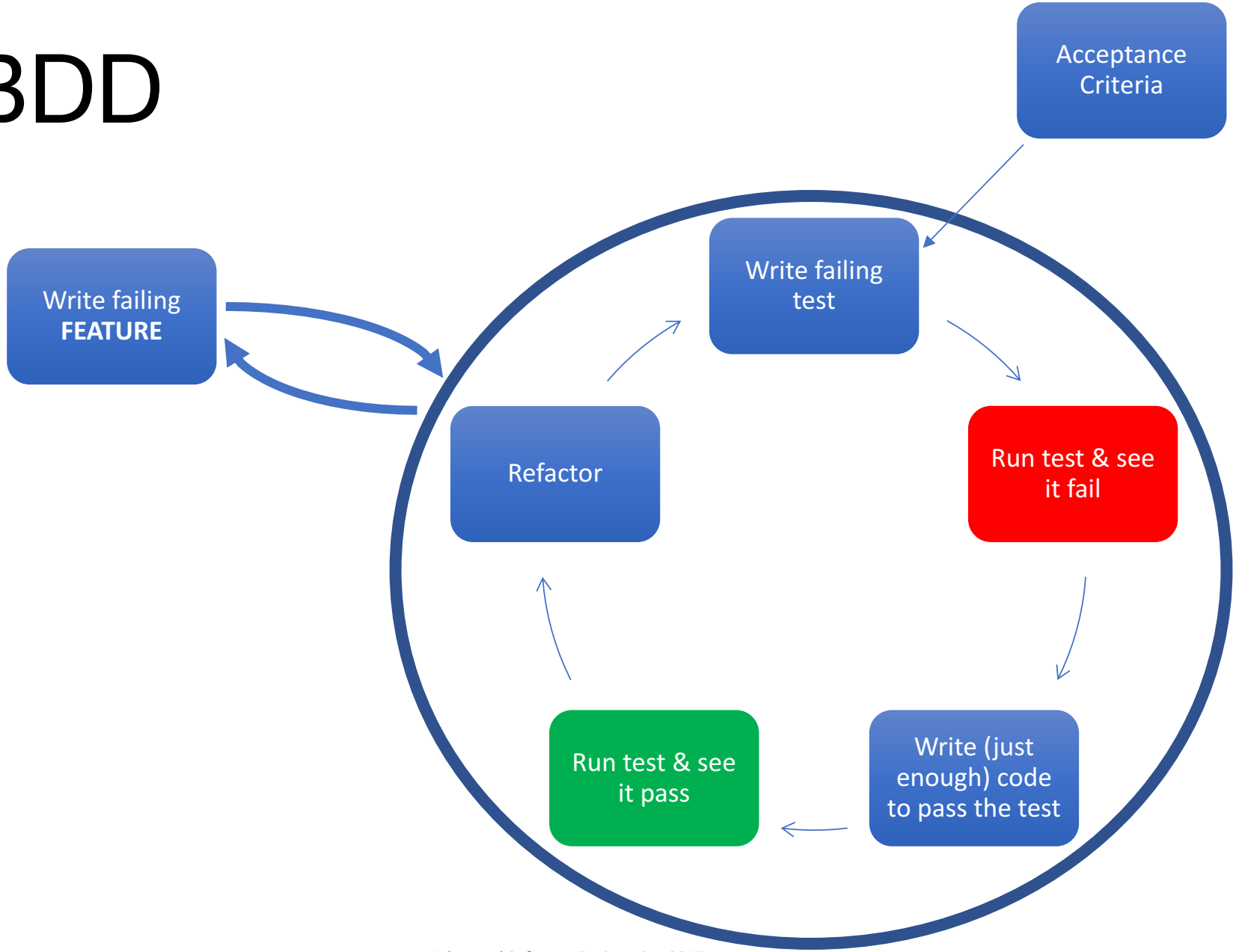
- Test-first approach
- Think about “how to use a component” before “how to implement a component”
- Mainly a design technique

TDD Process


Red
Green
Refactor


Write test
Run test & see it fail
Write (just enough) code to pass the test
Run test & see it pass
Refactor

BDD


Acceptance Criteria
Write failing FEATURE
Write failing test
Run test & see it fail
Write (just enough) code to pass the test
Run test & see it pass
Refactor

Feature

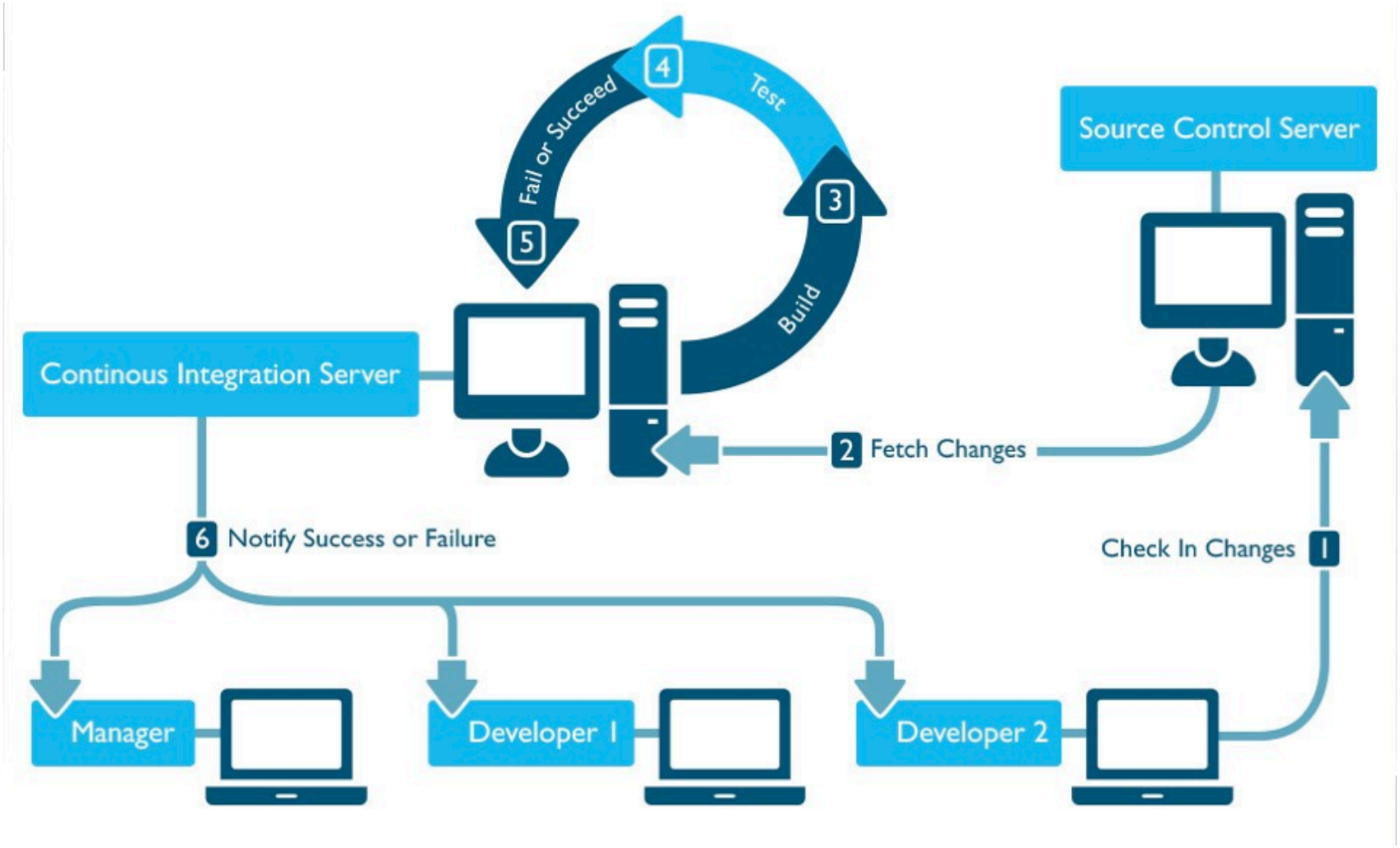
- Format:
  As a _____
  I want to _____
  In order to _____

- Example:
  As an instructor
  I want to grade an assignment
  In order to give feedback to my students

Acceptance criteria

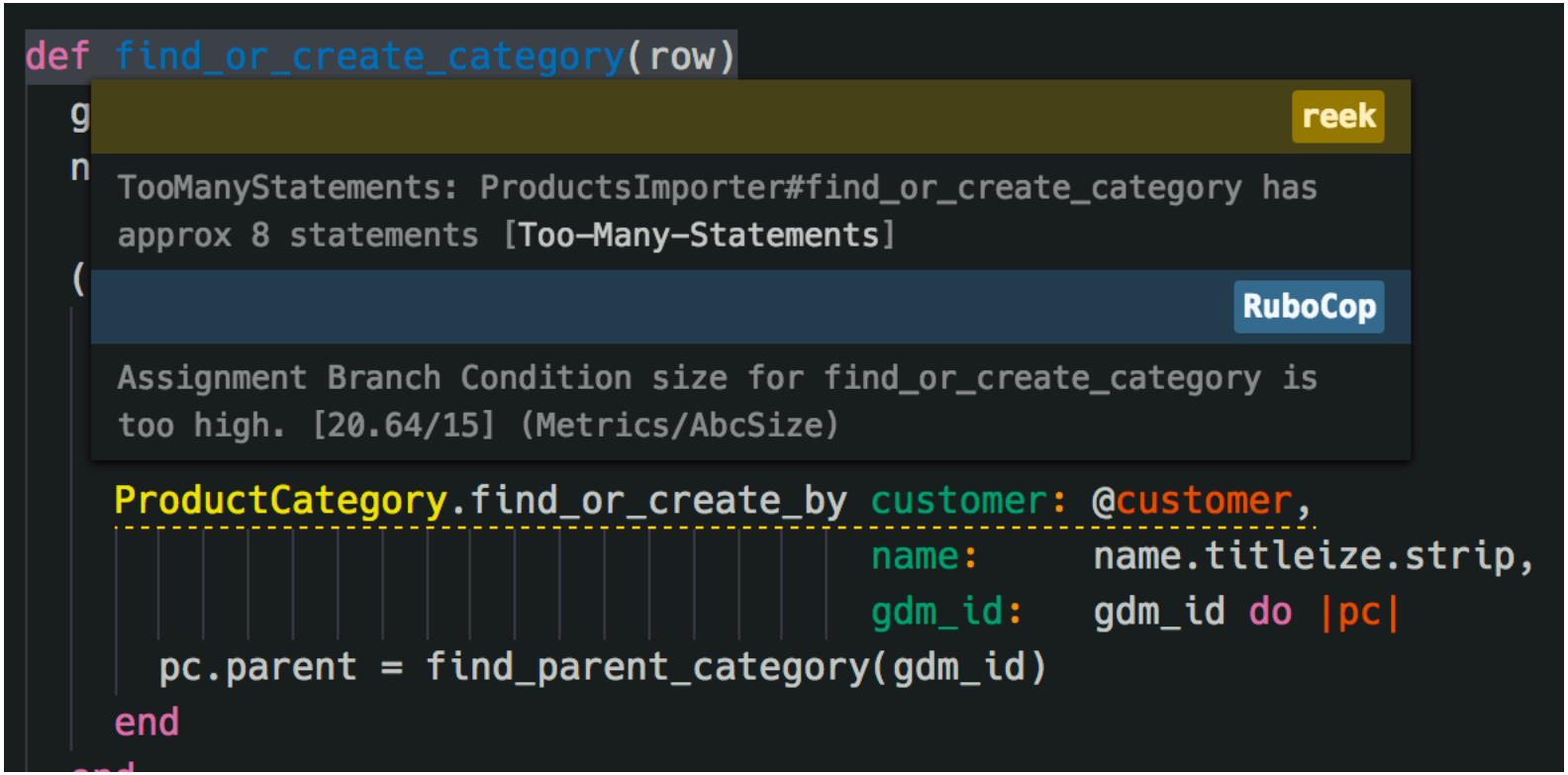
- Describe scenarios that display the expected behavior
- Example:
  Given I am an instructor
  And I am viewing the students submissions
  When I enter a valid grade
  Then the grade is displayed in the gradebook

Continuous Integration/Delivery



Testing Rails apps

- Static Analysis
  - Rubocop: check coding standards
  - Reek: check for code smells
  - Brakeman: detect security vulnerabilities
  - Eslint: lint JS code
  - Sass-lint: lint your stylesheets


def find_or_create_category(row)
reek
TooManyStatements: ProductsImporter#find_or_create_category has approx 8 statements [Too-Many-Statements]
RuboCop
Assignment Branch Condition size for find_or_create_category is too high. [20.64/15] (Metrics/AbcSize)
ProductCategory.find_or_create_by customer: @customer,
name: name.titleize.strip,
gdm_id: gdm_id do |pc|
pc.parent = find_parent_category(gdm_id)
end

Testing Rails apps (2)

- Unit Testing/TDD
  - Minitest (default)
  - RSpec (better, de-facto standard)
- Acceptance Tests/BDD
  - Capybara
  - Cucumber