# Test Driven Development (TDD)

and

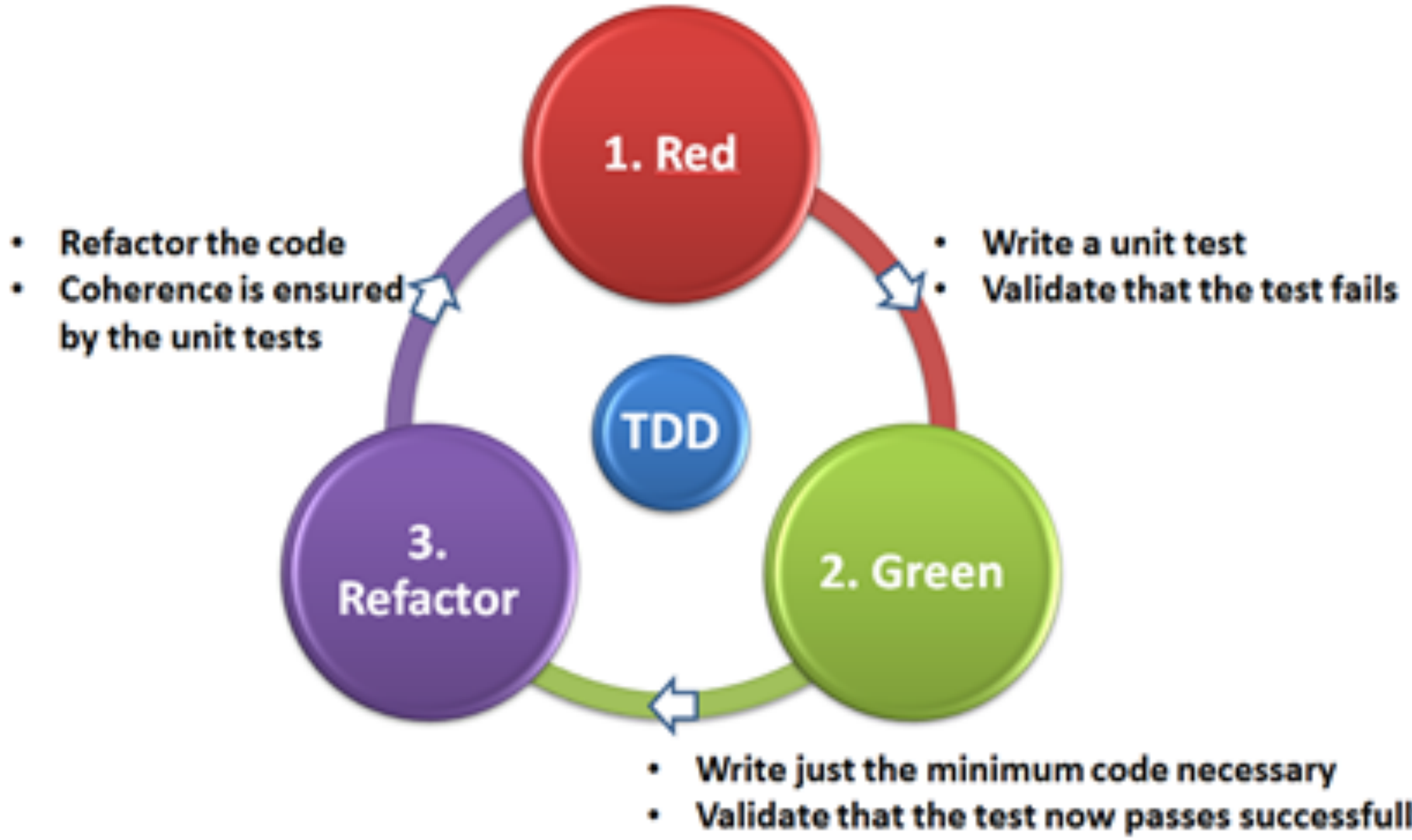
# Behavior Driven Development (BDD)

SWE 261P

# Test Driven Development (TDD)

TDD=test driven development

- Define the test first
- Implement the functionality
- Run the test
- Iterate


1. Red
• Write a unit test
• Validate that the test fails
2. Green
• Write just the minimum code necessary
• Validate that the test now passes successfully
3. Refactor
• Refactor the code
• Coherence is ensured by the unit tests
TDD

# Read Dan North article: "Introducing BDD"

- http://dannorth.net/introducing-bdd/

# Behavior Driven Development (BDD)

- Motivations: Programmers needed help with TDD
  - Knowing where to start
  - What to test
  - What not to test
  - How much to test
  - What to call their test cases
  - How to understand why a test fails

# BDD (from North)

- Test method names should be sentences
  - An expressive test name is helpful when a test fails
  - A simple sentence template keeps test methods focused.

# BDD (from North)

- BDD provides a “ubiquitous language” for analysis
  - Requirements
  - Design
  - Business Analysis
  - Testing

# BDD (from North article)

- Behavior is a more useful word than test.
  - Emphasize behavior over testing.
  - Method:
    - ~~Write (initially failing) Test~~
    - Determine (or define) the next most important behavior.
    - Implement Functionality for ~~Test~~ Behavior
    - Test Functionality against Behavior (typically using automated testing frameworks)
    - Iterate

# BDD vs. TDD

- BDD often considered more high-level than TDD (which is generally thought of at a smaller unit-test level)
  - Generally true…
  - but, not necessarily so: BDD can be used at the unit-testing level, and TDD can be used at the system level
- Difference in focus/mindset

# BDD

- Imposes a structure to ***test specifications*** or in the new parlance ***"behavior specifications"***
- States the purpose and situation of each test
- Based on Agile Process concepts of "user stories"

# BDD Forms

User Story

As a [X]
I want [Y]
so that [Z]

Scenario

**Given** some initial context (the givens),
**When** an event occurs,
**then** ensure some outcomes.

# Example

+**Title**: Customer withdraws cash+
**As a** customer,
**I want to** withdraw cash from an ATM,
**so that** I don't have to wait in line at the bank.

+**Scenario** 1: Account is in credit+
**Given** the account is in credit
**And** the card is valid
**And** the dispenser contains cash
**When** the customer requests cash
**Then** ensure the account is debited
**And** ensure cash is dispensed
**And** ensure the card is returned

+**Scenario** 2: Account is overdrawn past the overdraft limit+
**Given** the account is overdrawn
**And** the card is valid
**When** the customer requests cash
**Then** ensure a rejection message is displayed
**And** ensure cash is not dispensed
**And** ensure the card is returned

# Non-User Run BDD

- Such framing as stories and scenarios lead to automation, which enables scaling

- But, does not accomplish the more-fuzzy "validation" aspects of UAT

- Tools such as JBehave, Cucumber, RSpec, CBehave, Jasmine (for JavaScript), Spock