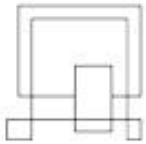
VIA University College
Software Engineering



# Test cases

## [SWE1]

# From Requirements To Code

**ANALYSIS** — **DESIGN**

*Problem Focus, WHAT? For Customers & Programmers*

*Solution Focus, HOW! For Programmers*

Problem Domain

Description

Background Description

Generalization

Defines

Writes

Non-Functional Requirements

User Stories

Combine into Use Cases

Use Case Diagram

Description

Activity Diagram

Use Case Descriptions

Scenario Clarification

System Sequence Diagram

Scenario Clarification

Detailed Behaviour

Scenario Verification

Clarification

Test Cases

State Machine Diagram

All diagrams must have a textual description

Domain Model Description

Description

Domain Model

Leads To

Leads To

Conceptual ER-Diagram

Normalisation & mapping

Global ER/Relation Diagram

Design Class Diagram

Extracted from

Extracted from

Created interactively

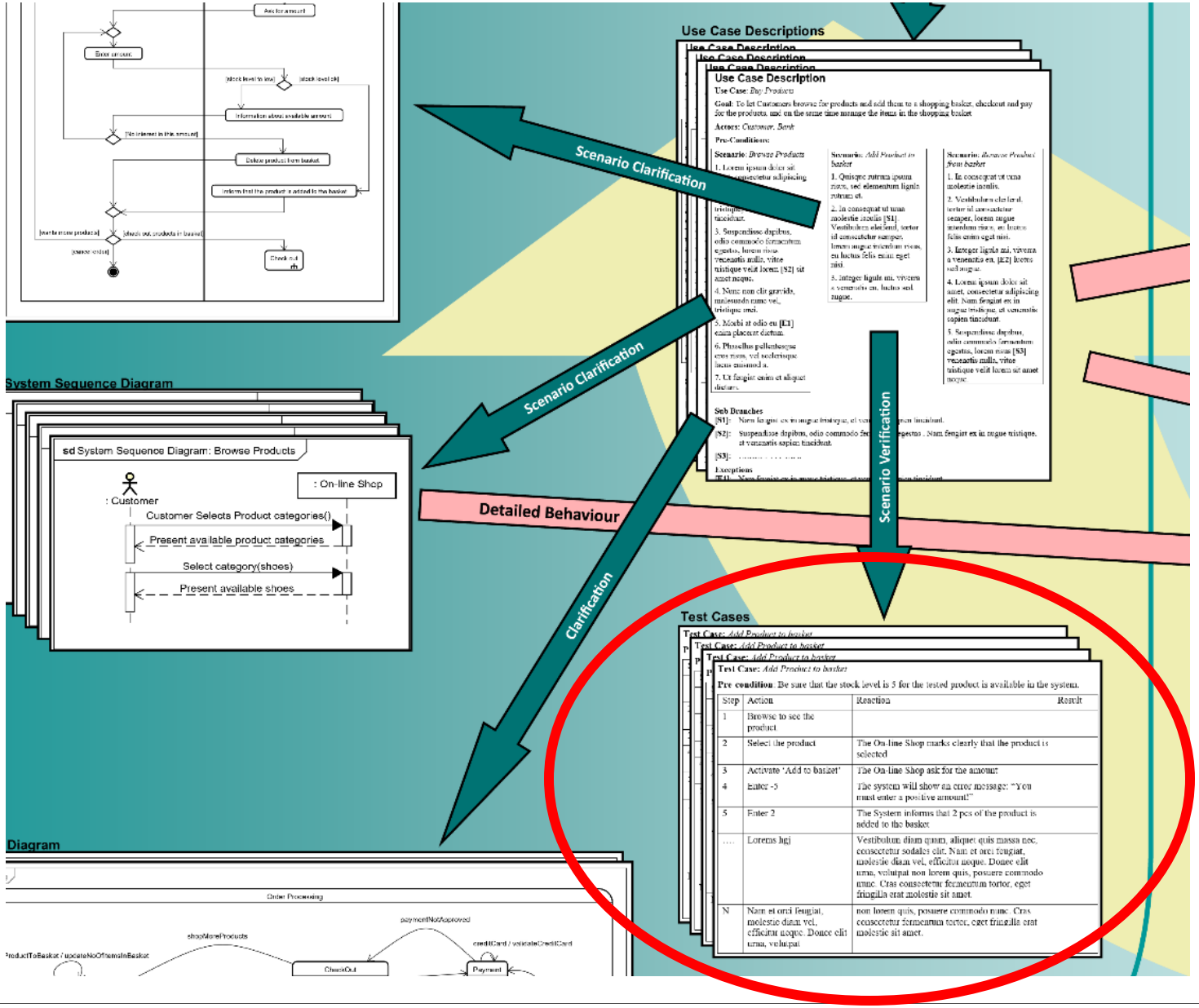
Sequence Diagram

Deployment Diagram

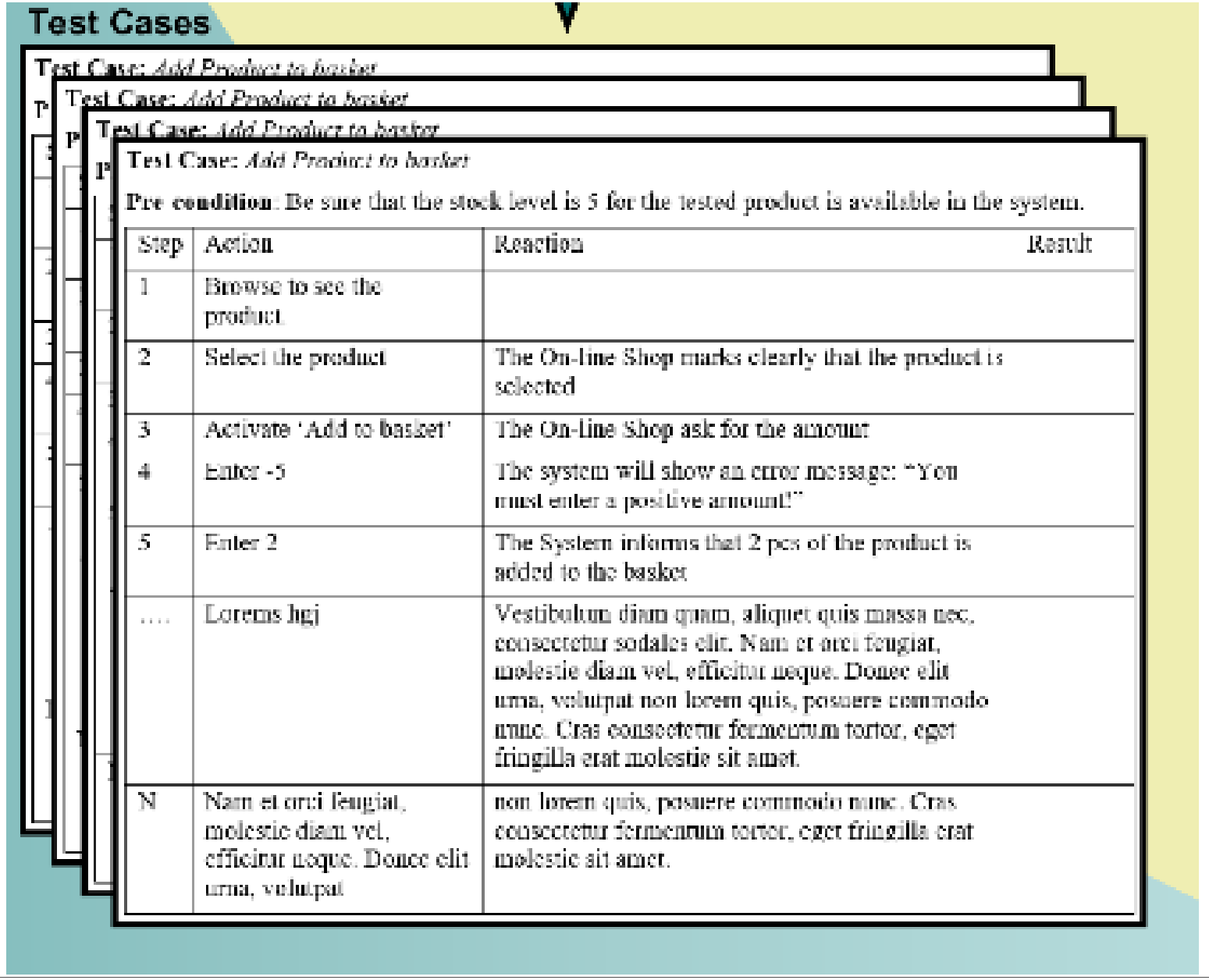
Code

*Problem Domain Vocabulary!*

# Test cases


Use Case Descriptions
Use Case Description
Use Case: Buy Products
Scenario Clarification
Scenario Clarification
Scenario Verification
Detailed Behaviour
Clarification
System Sequence Diagram
sd System Sequence Diagram: Browse Products
: Customer
: On-line Shop
Customer Selects Product categories()
Present available product categories
Select category(shoes)
Present available shoes
Test Cases
Test Case: Add Product to basket
Pre condition: Be sure that the stock level is 5 for the tested product is available in the system.
Step
Action
Reaction
Result
1
Browse to see the product
2
Select the product
The On-line Shop marks clearly that the product is selected
3
Activate 'Add to basket'
The On-line Shop ask for the amount
4
Enter -5
The system will show an error message: "You must enter a positive amount!"
5
Enter 2
The System informs that 2 pcs of the product is added to the basket
....
Lorems hgj
Vestibulum diam quam, aliquet quis massa nec, consectetur sodales elit. Nam et orci feugiat, molestie diam vel, efficitur neque. Donec elit urna, volutpat non lorem quis, posuere commodo nunc. Cras consectetur fermentum tortor, eget fringilla erat molestie sit amet.
N
Nam et orci feugiat, molestie diam vel, efficitur neque. Donec elit urna, volutpat
non lorem quis, posuere commodo nunc. Cras consectetur fermentum tortor, eget fringilla erat molestie sit amet.
Diagram
Order Processing
shopMoreProducts
paymentNotApproved
CheckOut
Payment

# Test cases


Test Cases
Test Case: Add Product to basket
Test Case: Add Product to basket
Test Case: Add Product to basket
Test Case: Add Product to basket
Pre condition: Be sure that the stock level is 5 for the tested product is available in the system.
Step
Action
Reaction
Result
1
Browse to see the product
2
Select the product
The On-line Shop marks clearly that the product is selected
3
Activate 'Add to basket'
The On-line Shop ask for the amount
4
Enter -5
The system will show an error message: "You must enter a positive amount!"
5
Enter 2
The System informs that 2 pcs of the product is added to the basket
....
Loremsbgj
Vestibulum diam quam, aliquet quis massa nec, consectetur sodales elit. Nam et arci feugiat, molestie diam vel, efficitur neque. Donec elit urna, volutpat non lorem quis, posuere commodo nunc. Cras consectetur fermentum tortor, eget fringilla erat molestie sit amet.
N
Nam et arci feugiat, molestie diam vel, efficitur neque. Donec elit urna, volutpat
non lorem quis, posuere commodo nunc. Cras consectetur fermentum tortor, eget fringilla erat molestie sit amet.

# Testing

- Testing is about measuring compliance to expected behaviour
  - You cannot test without knowing the expected behaviour
  - The purpose of a test is *never* to show that the program is correct
  - The purpose of the test is to find bugs
- Alpha and Beta testing
- Black box vs White box testing
- Level of testing
  - User acceptance testing
  - Integration testing
  - Unit testing
- The V-model

# Alpha and beta testing

- Alpha and beta testing are unstructured
  - Alpha test is internal
  - Beta test is external
- They ask the tester (an end user) to use the program in real-life scenarios
- Both usability and bug testing
- It's only a test if you get feedback and bug reports

# Black box testing

- In black box testing, we test the interface of a thing without knowing (or caring about) the internal structure
- The interface could be
  - The user interface
  - The public methods of a view model
  - A communication protocol (RMI/sockets)
  - The public methods of any model classes or other classes

# White box testing

- Testing with full access to the implementation
- Purpose of the test is to make sure tricky parts are tested
- Special focus is on *code coverage*
  - Did all lines of code run during the test? If not, how many?
  - Where all if-statements tested with both outcomes?
  - Where all loops run 0, 1, and many times?
- More important than the actual percentage is *which* lines of code didn't run

# Unit testing

- A unit is a low level piece of the system
  - Typically a method or a class
- Unit testing exhaustively tests that the unit performs as expected
  - Expected meaning: As designed
  - Sometimes the unit tests are the documented behaviour of the unit
  - Also test situations that are not likely to happen in the full system
- Can be both a white and black box test, but usually a black box test
  - Focus on code coverage
  - Often uses Test-driven development for highest code quality
- JUnit is obviously used for this

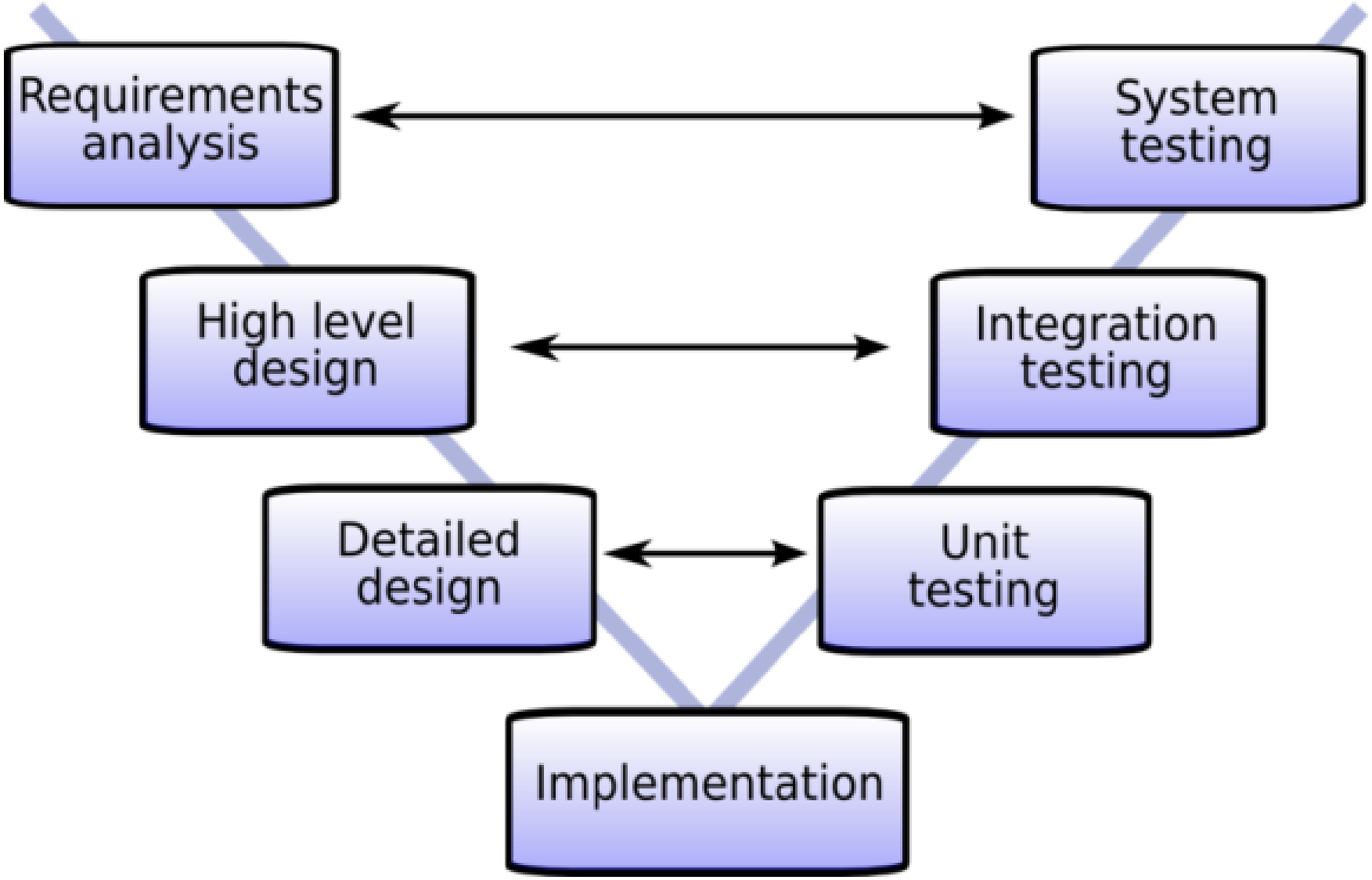
# Integration tests

- Used to test that the units work together correctly
- Often used to test the architecture
  - In particular, how calling a method in the ViewModel manages to call all the way to the database
- Not exhaustive testing
  - Only key scenarios
- Requires some knowledge of the architecture, but little to no knowledge of the code
  - Sometimes called *grey box testing*
- JUnit is good to use here

# Acceptance testing / System testing

- System (or functional) testing is almost the same
  - Acceptance testing is external
  - System/ functional testing is internal
- Testing compliance to *requirements*
  - Requirements are typically in the form of use cases
  - This is about testing the (supposed) final program
- Always a black box test

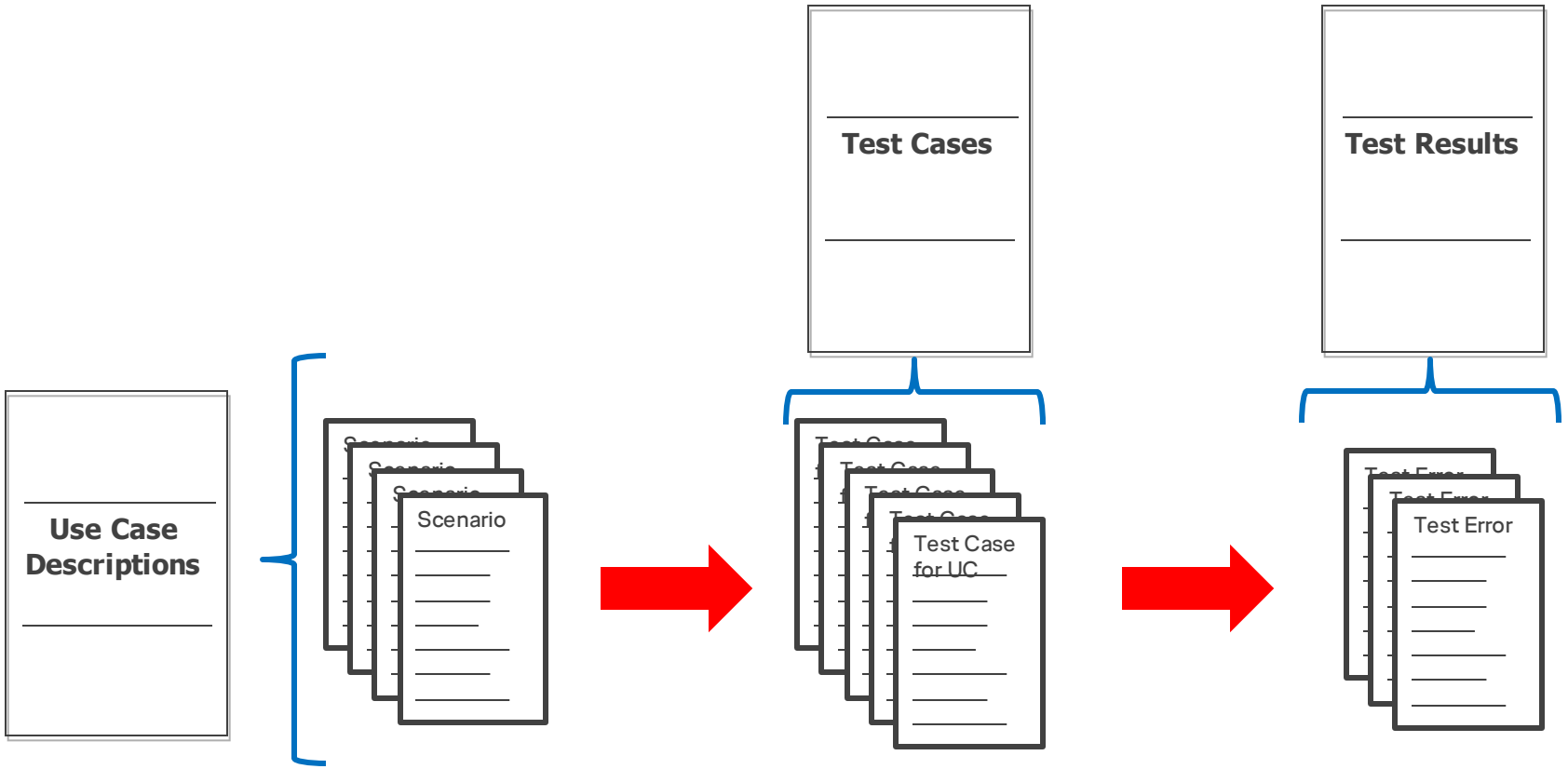
# V-model


Requirements analysis
System testing
High level design
Integration testing
Detailed design
Unit testing
Implementation

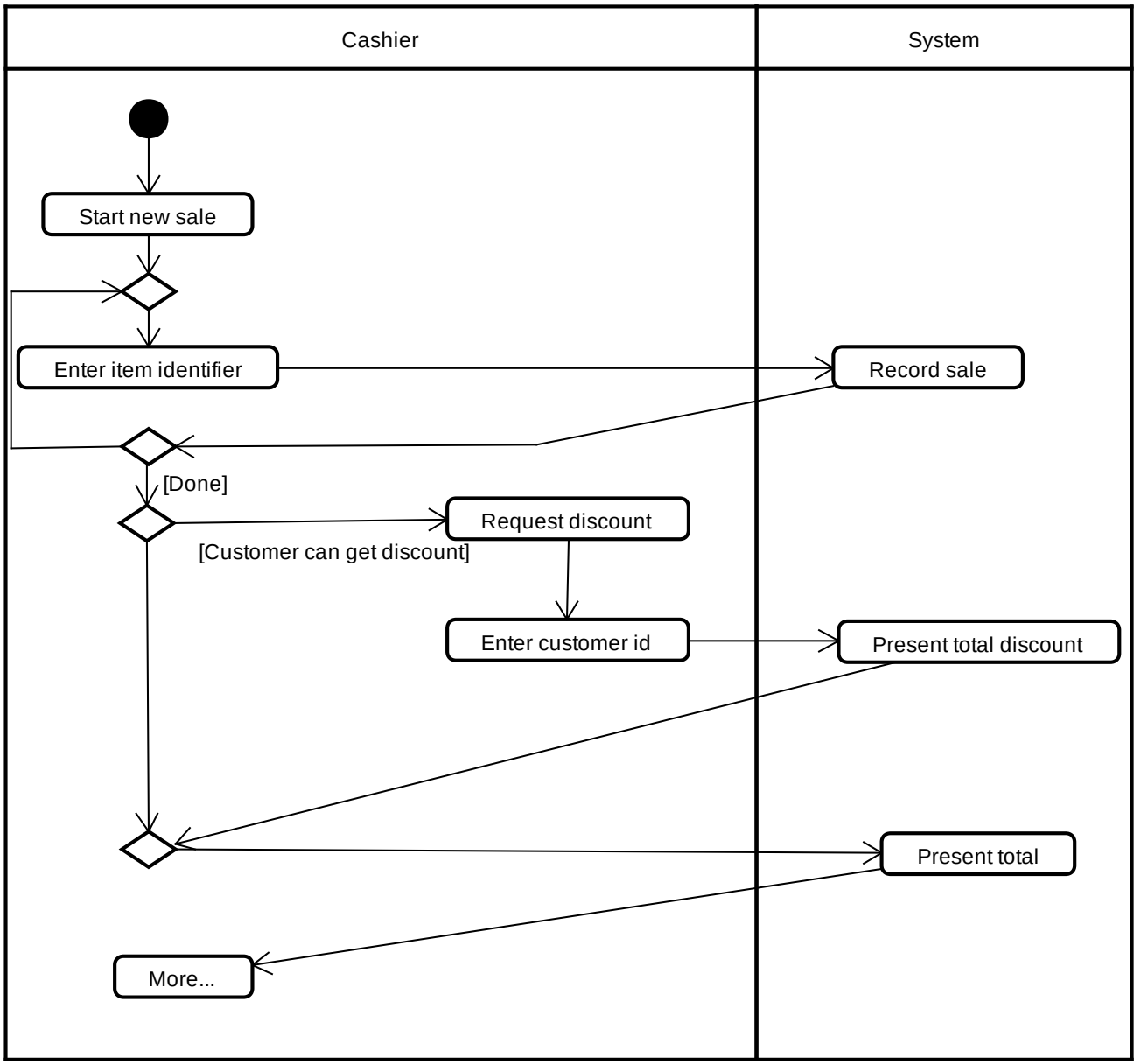
# Use case scenarios

- Use case scenarios are both the Base sequence and the Alternate sequences
- For test purposes we also have to look at:
  - Looping scenarios
    - Zero/one/more (ZOM) rule
    - Sometimes zero/more is enough
    - Sometimes zero is supposed to be impossible – *that needs to be tested.*
  - Relevant combinations of alternate scenarios
- This is a lot of scenarios, but should they really go untested?

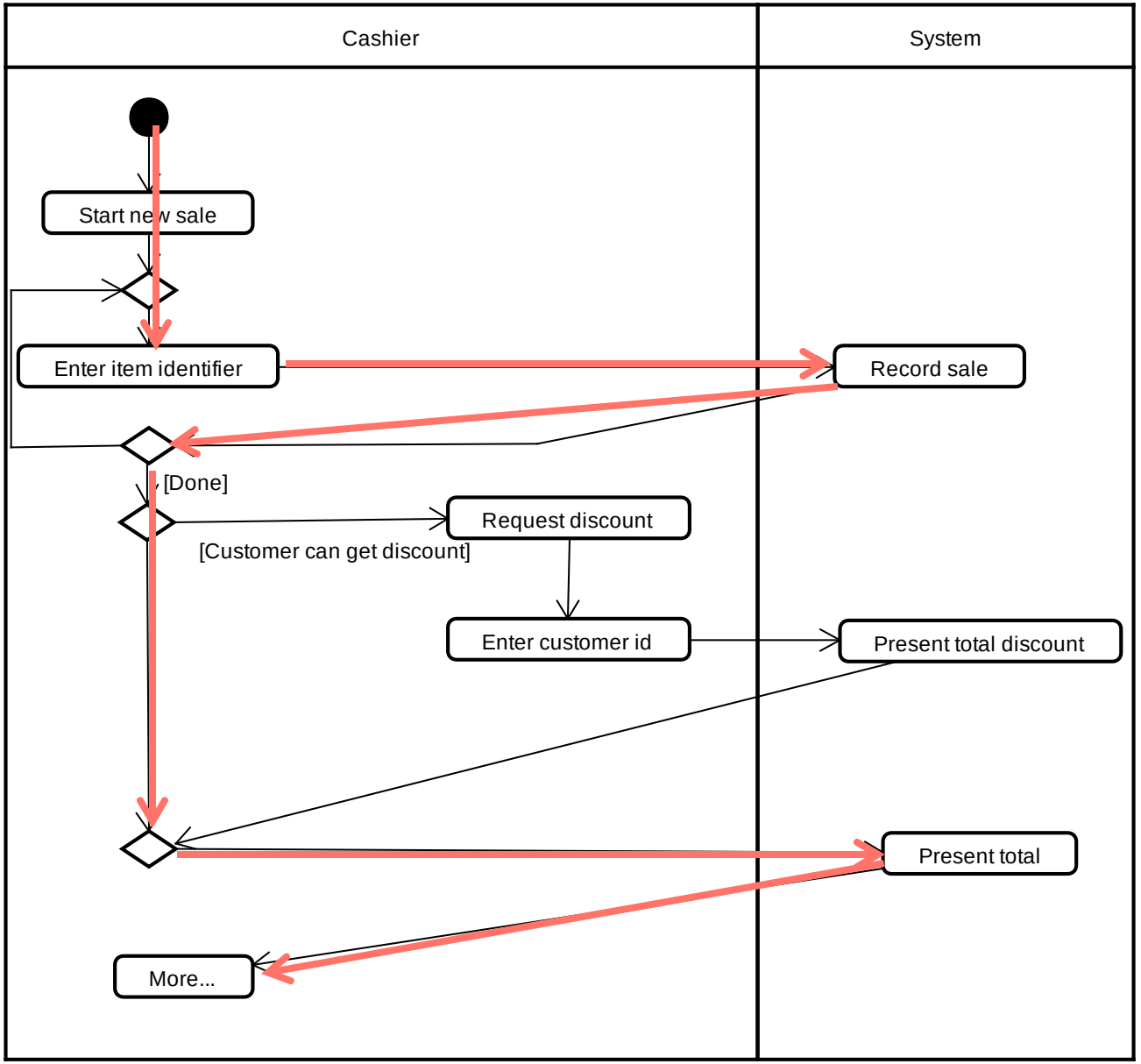
# Use Case and Test Cases



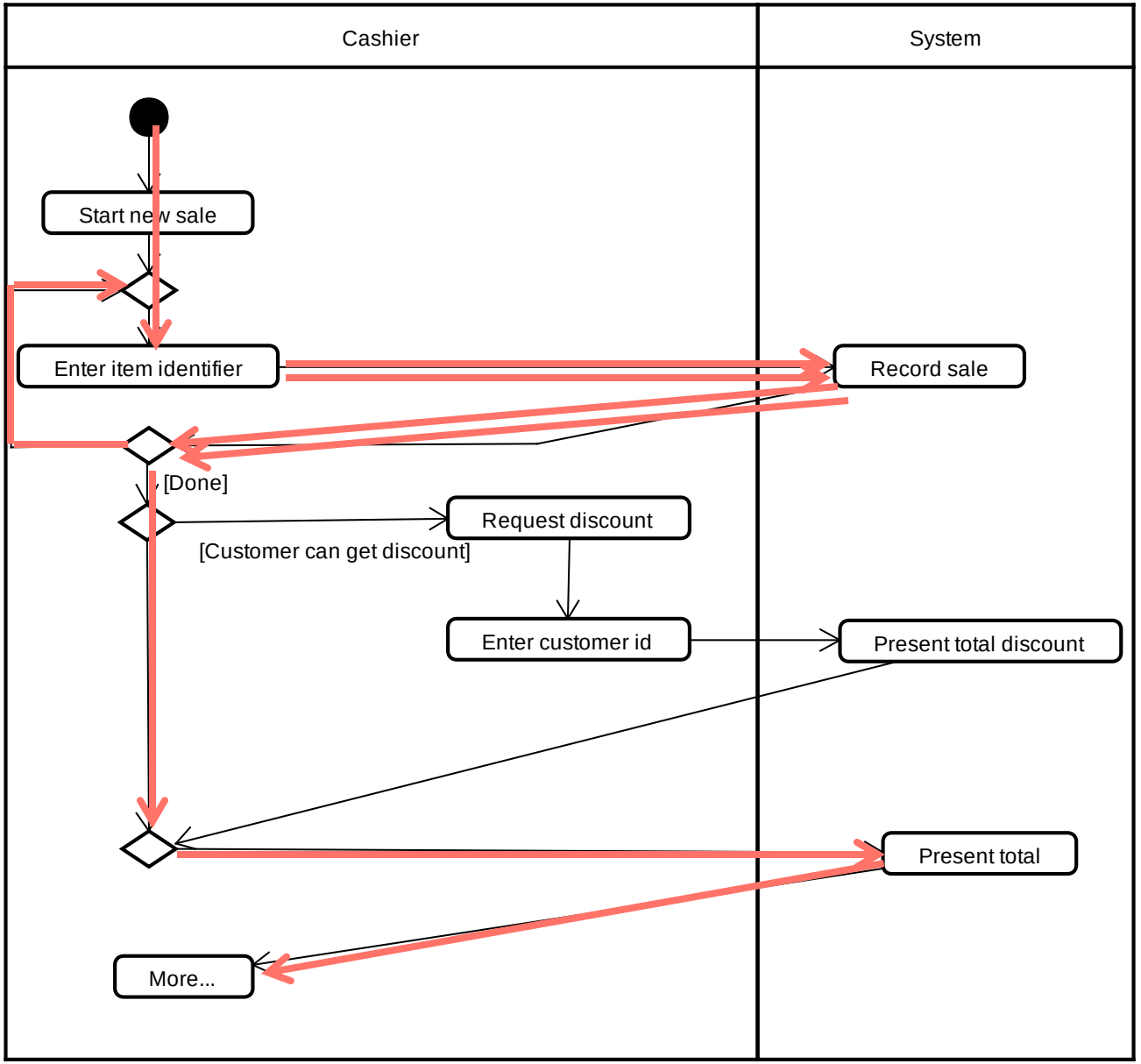
# Activity diagram



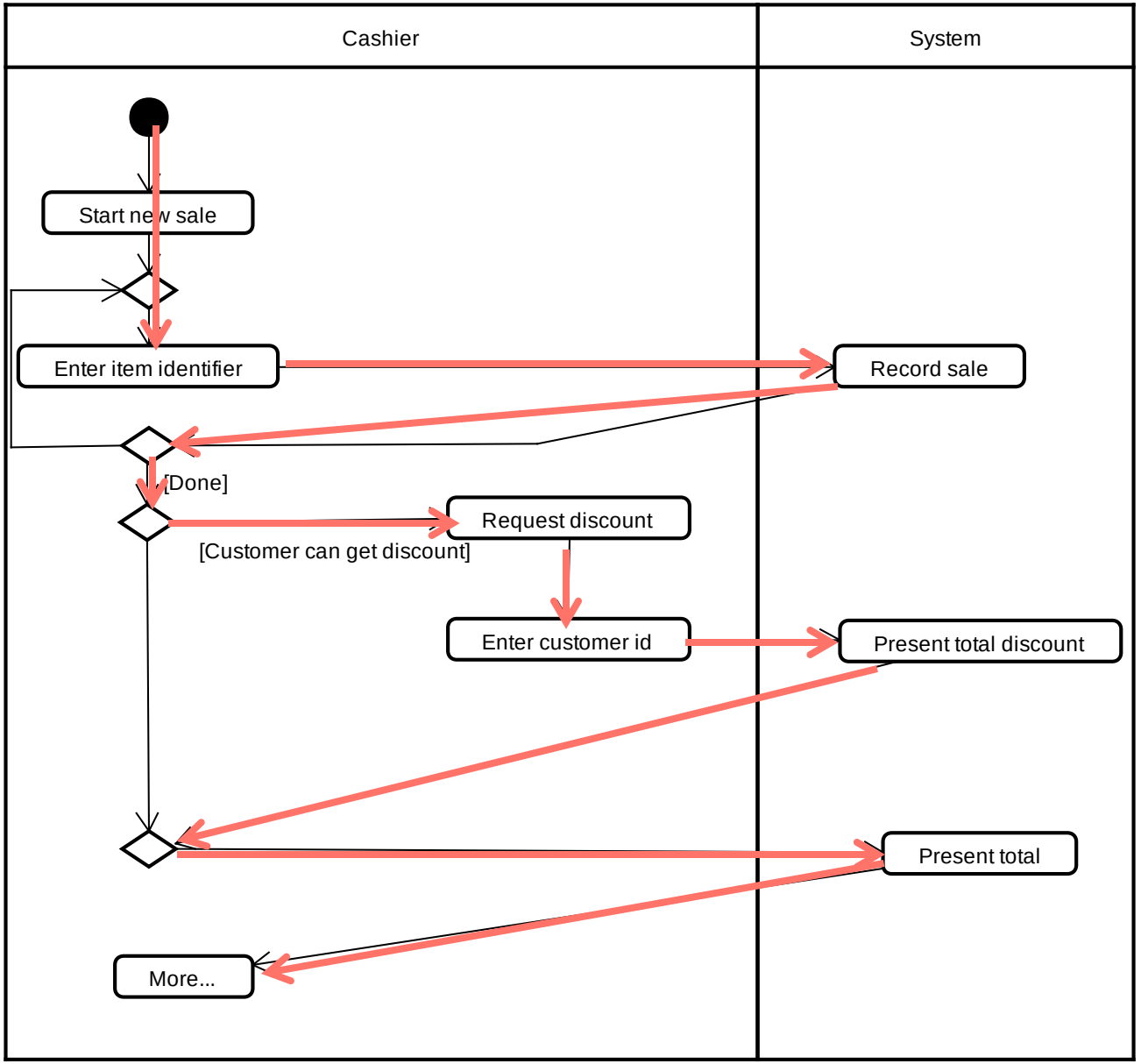
# Main scenario with 1 item



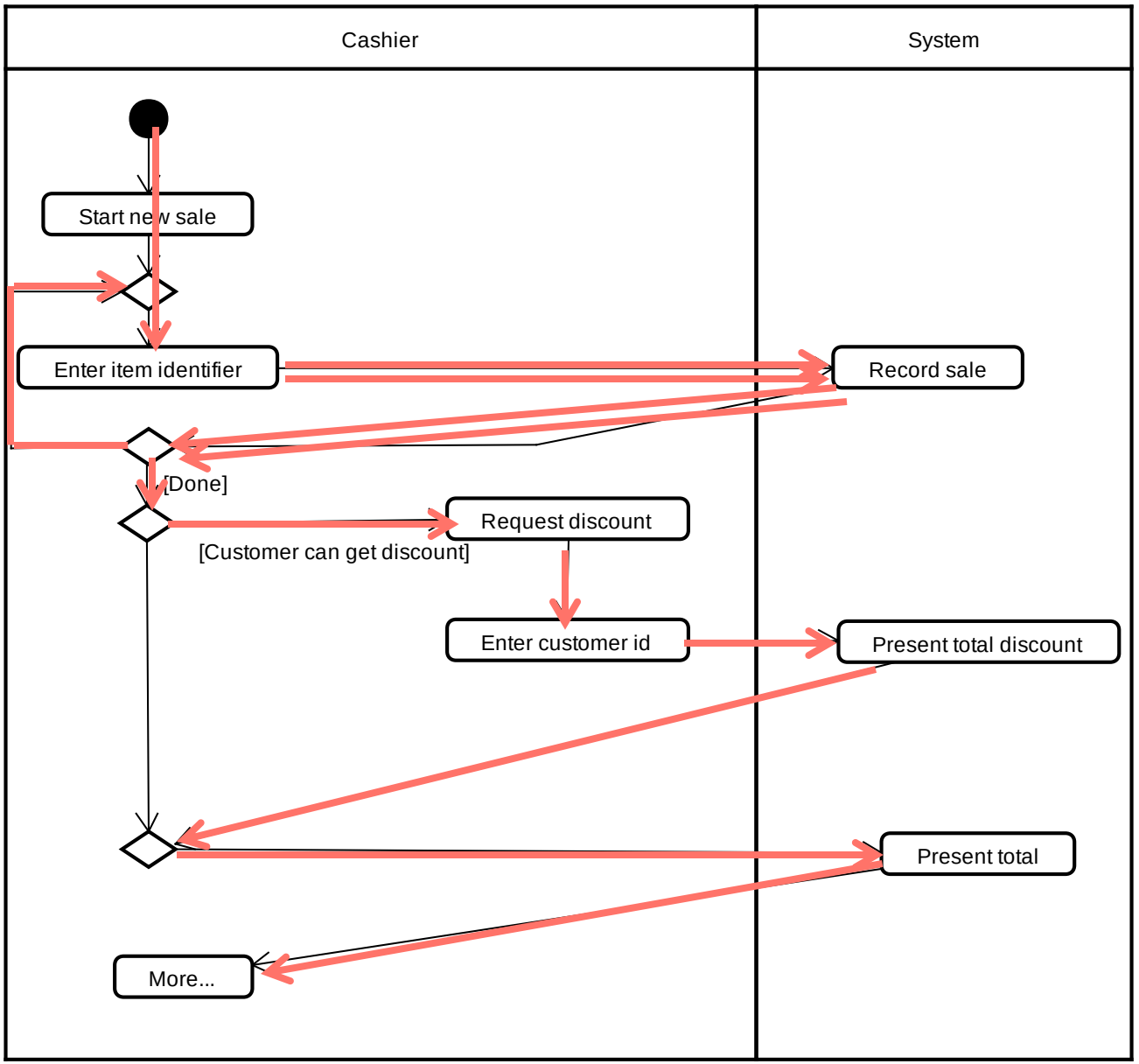
# Main scenario with more items



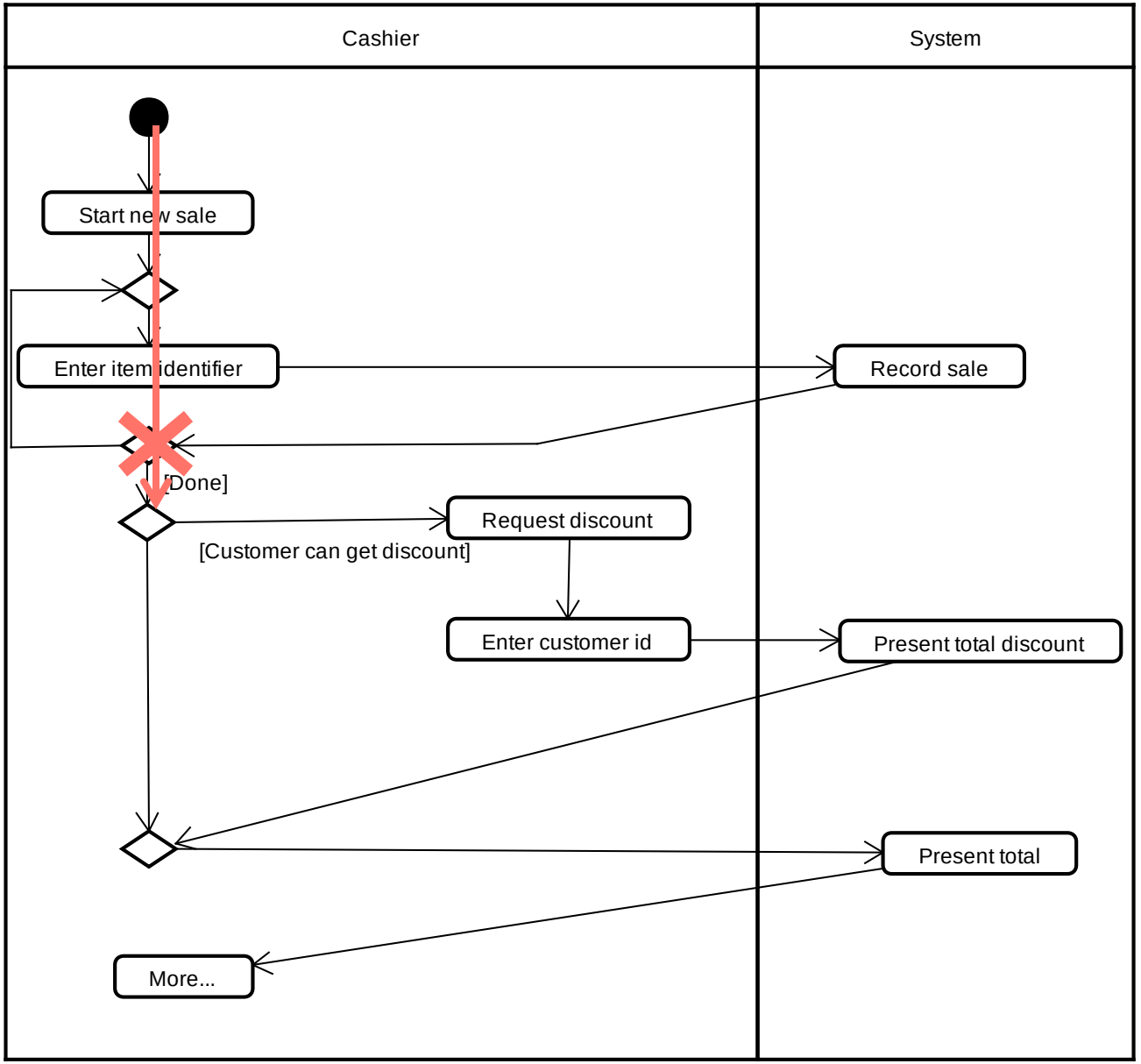
# Alternate scenario with 1 item



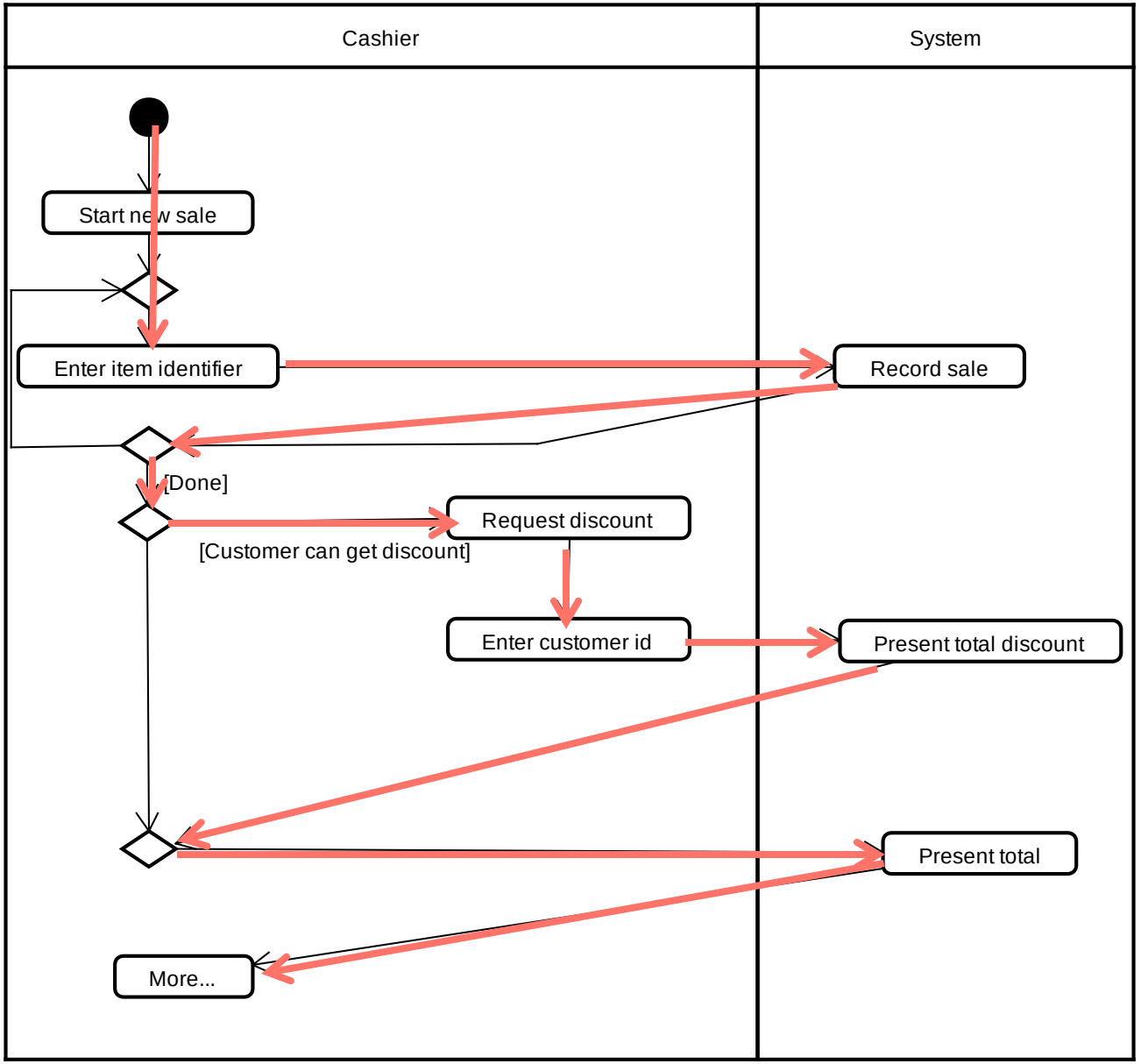
# Alternate scenario with more items



# 0 items impossible



# Example scenario



# Find relevant values

- Regular values
  - "Hello" is a perfectly fine string for testing
  - For numbers, the 0, 1, more rule (ZOM) applies as well
  - Lists with various number of items – 0, 1, more (ZOM)
- Blank values
  - Empty strings, lists, null strings and other objects
  - 0
- Boundary values
  - Minimal allowed value
  - Maximal allowed value
- Invalid values
  - One less than minimum
  - One greater than maximum

# Example scenario values

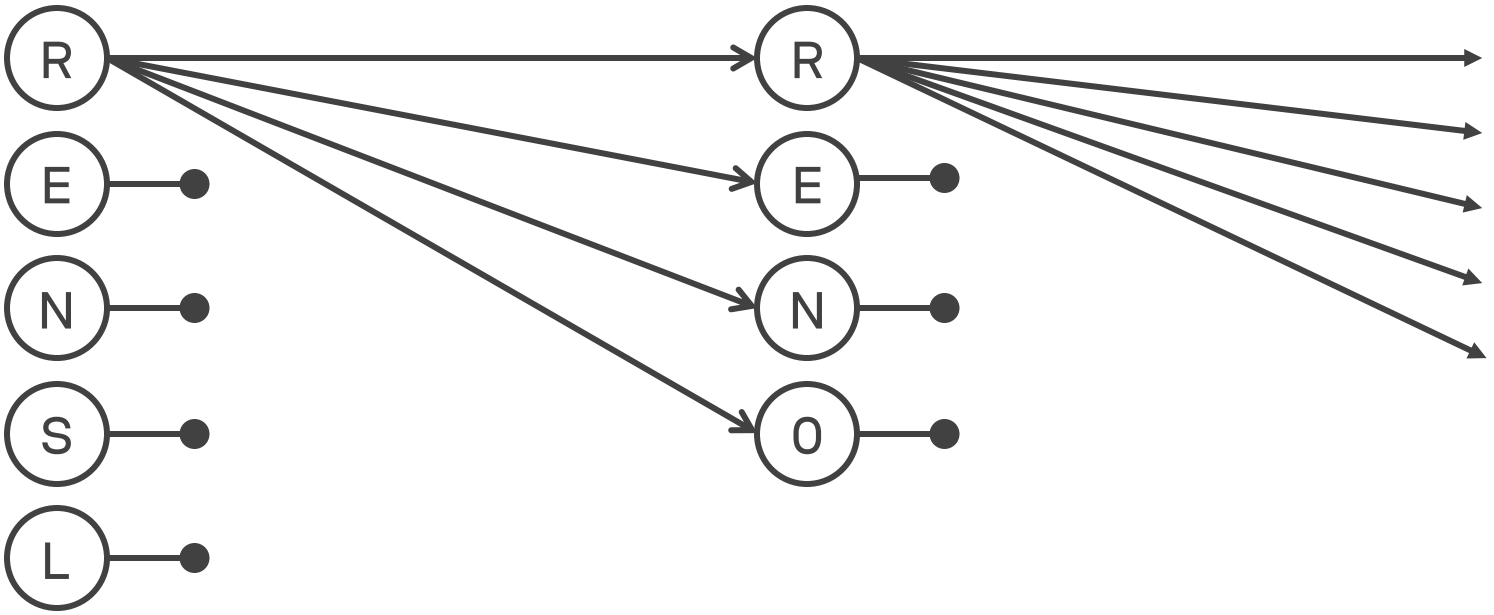
## Item ID

- Regular
- Empty
- Negative
- ID too short
- ID too long
- *Note: Not in database is another scenario*

## Customer ID

- Regular
- Empty
- Negative
- Zero
- *Note: Not in database is another scenario*

# Paths



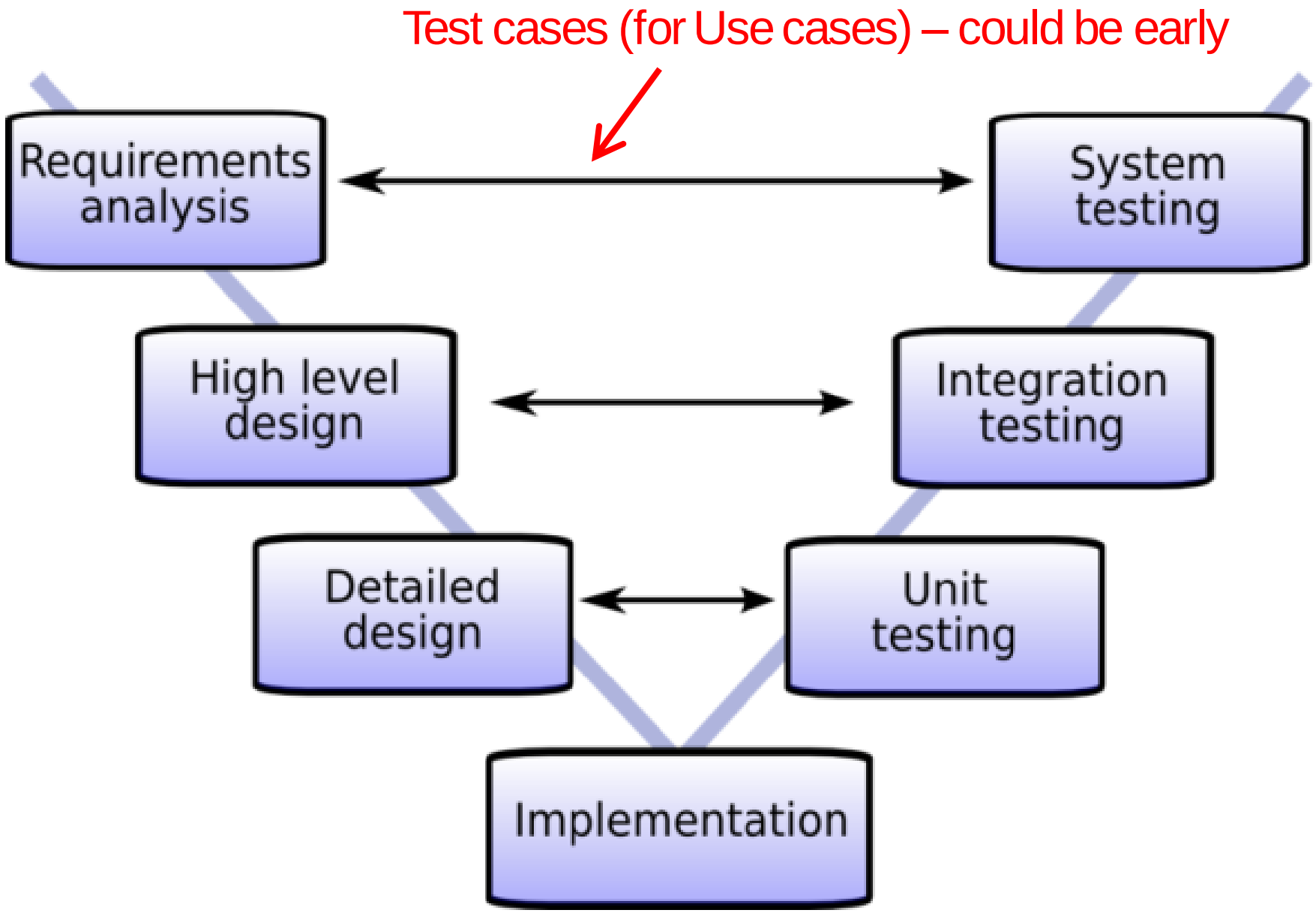
# Test cases

| Step | TC1 | TC2 | TC3 | TC4 | TC5 | TC6 | TC7 | TC8 |
|---|---|---|---|---|---|---|---|---|
| 2 | New sale | New sale | New sale | New sale | New sale | New sale | New sale | New sale |
| 3 | R | R | R | R | E | N | S | L |
| 4 | Done | Done | Done | Done | | | | |
| 5b (1) | Request discount | Request discount | Request discount | Request discount | | | | |
| 5b (2) | R | E | N | 0 | | | | |
| *(and so on)* | | | | | | | | |

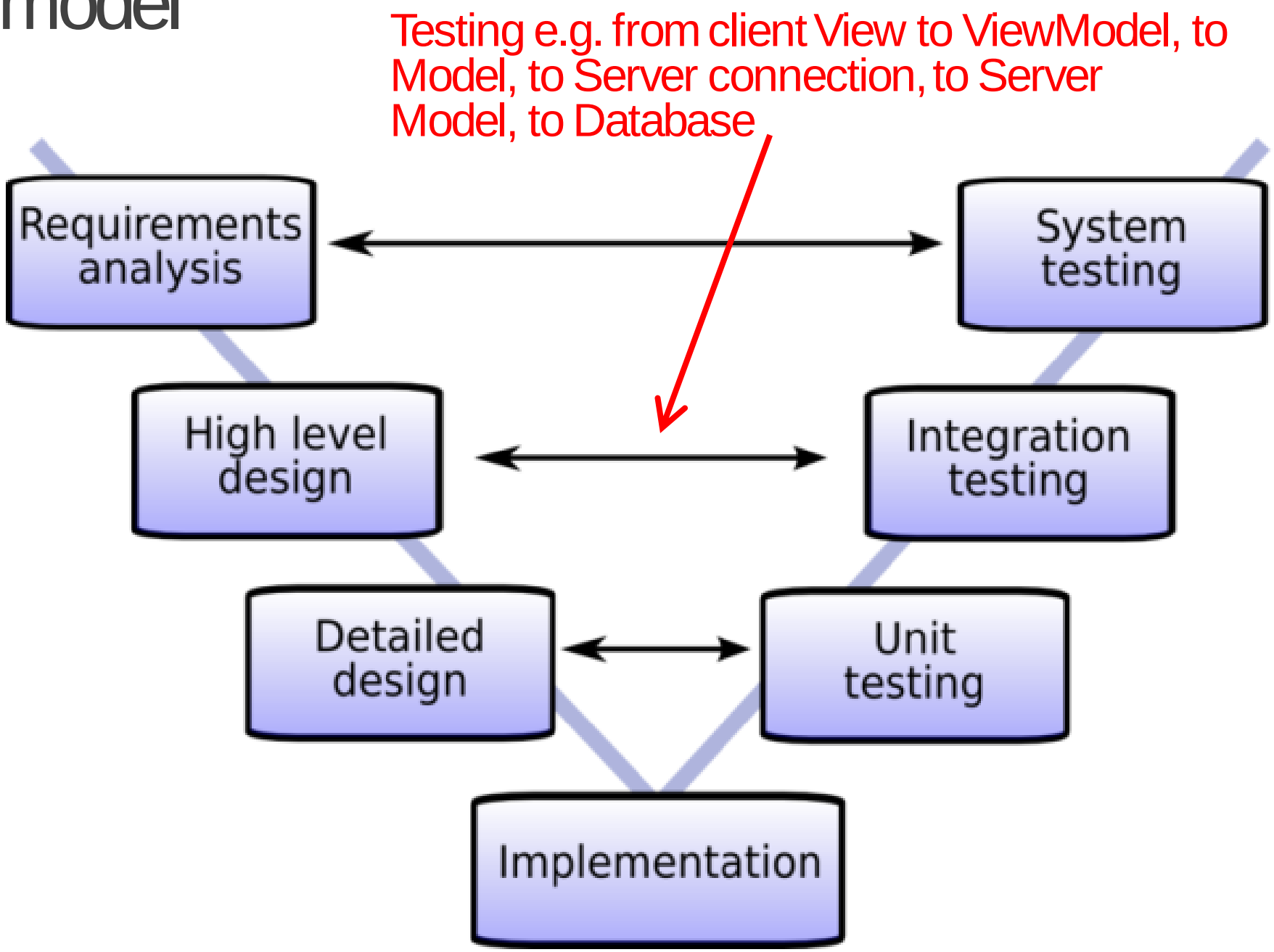
# Test case 1

| Step | Input field | Value | Expected result | Actual result |
|---|---|---|---|---|
| 2 – Start new sale | | | Requests Item ID | |
| 3 – Enter Item ID | Item ID | 549872316 | Accepted | |
| 4 | | | Total: $2.98 | |
| 5 – Indicate Done | | | Total: $2.98 | |
| 5b (1) – Request discount | | | Requests Customer Id | |
| 5b (2) – Enter Customer Id | Customer Id | 21519 | Accepted | |
| 5b (3) | | | Discount: $0.06 | |
| 5 | | | Total: $2.92 | |
| *(and so on)* | | | | |

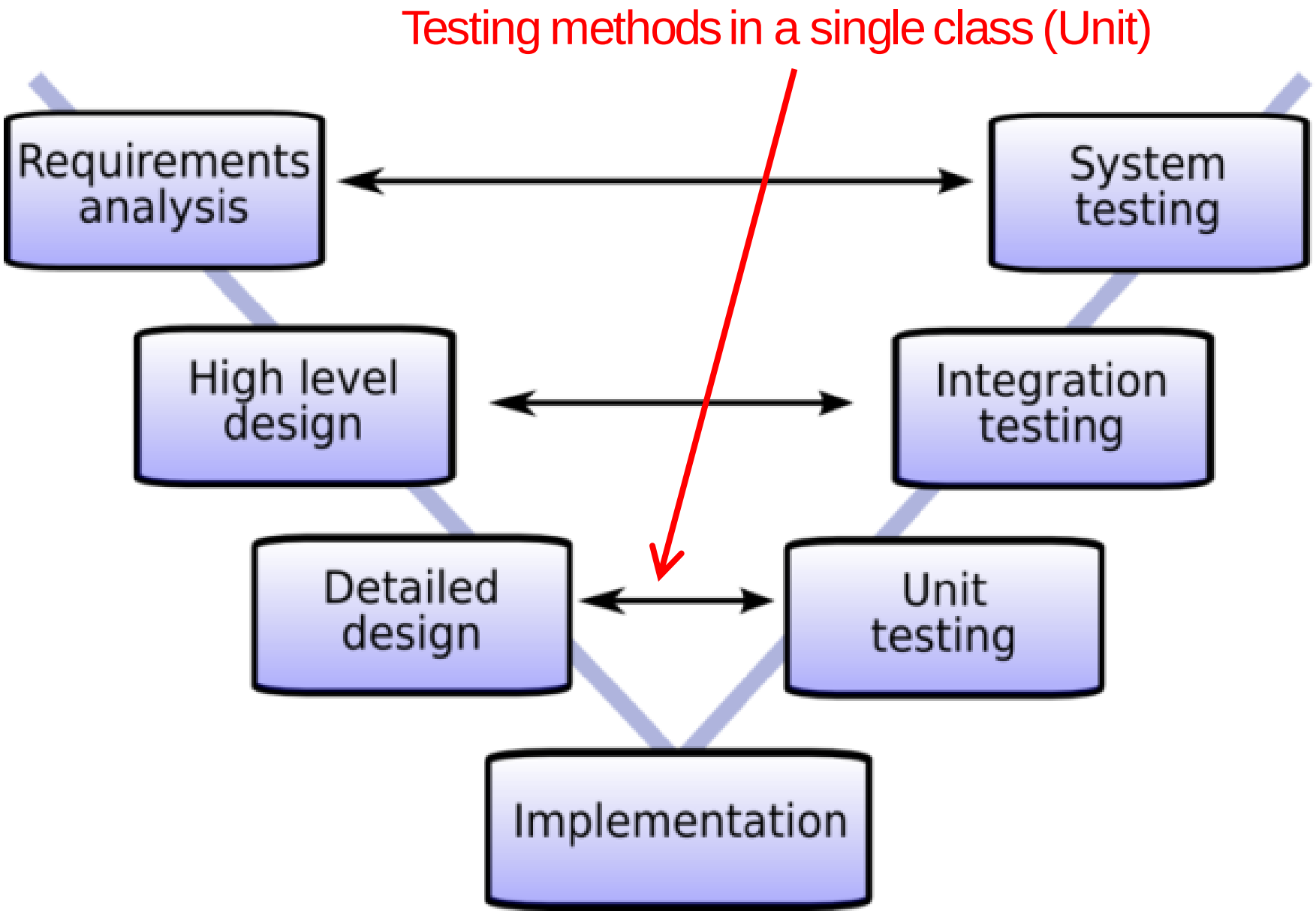
# V-model


Test cases (for Use cases) – could be early
Requirements analysis
System testing
High level design
Integration testing
Detailed design
Unit testing
Implementation

# V-model


Testing e.g. from client View to ViewModel, to Model, to Server connection, to Server Model, to Database
Requirements analysis
System testing
High level design
Integration testing
Detailed design
Unit testing
Implementation

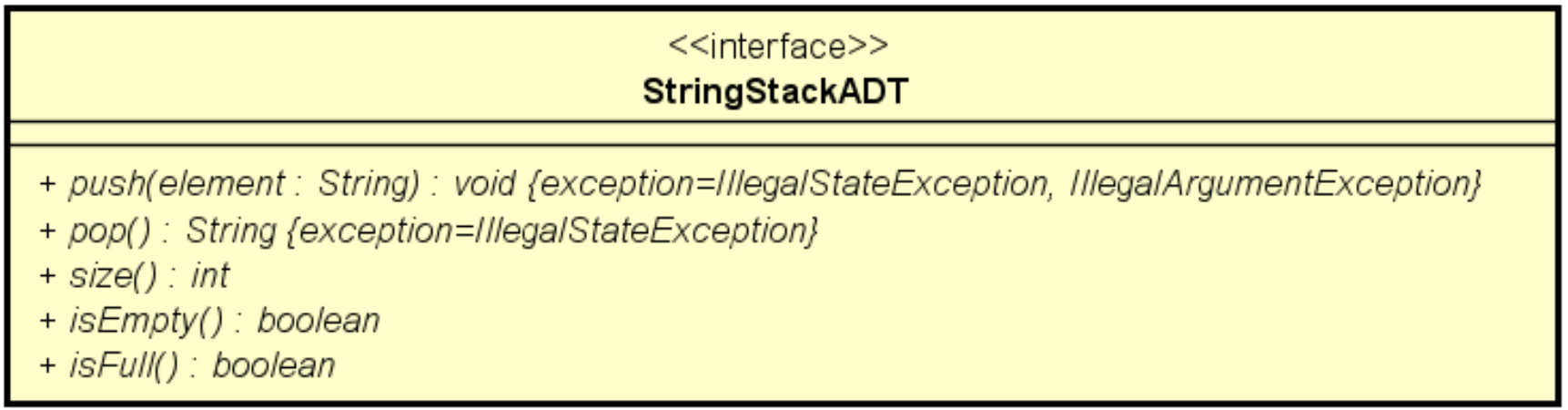
# V-model


Testing methods in a single class (Unit)
Requirements analysis
System testing
High level design
Integration testing
Detailed design
Unit testing
Implementation

# ZOMBIES

- Zero
  - Simple post-conditions of a just created object/module
- One
  - Tests dealing with single items
- Many
  - Tests dealing with more items or more complex scenarios
- Boundary
  - Lower bounds (left and right), upper bounds (left and right)
- Interface definition
  - Writing tests defines the needed interfaces for our modules
- Exceptions
  - Tests with exceptions being thrown
- Simple
  - Test simple scenarios one by one. Do not test many things in one test case
  - Implement the simplest solutions to pass the tests

# Unit test example: A Stack of Strings




- The stack could be implemented as bounded or unbounded (a max size or never full)
  - A max size less than 1 should not be accepted
- An element is pushed at the top
  - Pushing a `null` element throws an IllegalArgumentException
  - Pushing an element on a full stack throws an IllegalStateException
- An element is popped out from the top (LIFO)
  - Popping from an empty stack throws an IllegalStateException
- Method `size` returns the number of elements stored

# Unit test example: A Stack of Strings

`Constructor (int)`

- Z: 0 elements, should throw an exception
- O: 1 element, one element can be created
- M: 5 elements can be created
- B: {0, 1} (already tested in Z and O)
- E: -6 elements throws an exception

`push(String element)`

- Z: push a null string in a 5 element stack throws an exception
- O: one push in a 1 element stack can be done
- O: one push in a 5 element stack can be done
- M: 3 push in a 5 element stack can be done
- B: 5 element stack, number of push {5, 6}. The right boundary throws an exception the left don't.
- E: (already tested in Z and B)

# Unit test example: A Stack of Strings

`pop()`

- Z: (not relevant)
- O: one pop after no push in a 5 element stack throws an exception
- O: one pop after one push in a 1 element stack
- O: one pop after one push in a 5 element stack
- O: one pop after 3 push in a 5 element stack return the last pushed
- M: 3 pop after 4 push in a 5 element stack
- B: After 3 push in a 5 element stack, number of pop {3, 4}
- E: (already tested in O1)

`size()`

- Z: 5 element stack with no push has size 0
- O: one push in a 5 element stack has size 1
- O: one push and one pop in a 5 element stack has size 0
- M: {push, push, pop, push} in a 5 element stack has size 2
- B: 5 push in a 5 element stack has size 5
- E: (no exceptions thrown)

# Unit test example: A Stack of Strings

`isEmpty()`

- Z: 5 element stack with no push, isEmpty=true
- O: one push in a 5 element stack: isEmpty=false
- O: one push and one pop in a 5 element stack, isEmpty=true
- M: {push, push, pop, push} in a 5 element stack: isEmpty=false
- B: 5 push in a 5 element stack, isEmpty=false
- E: (no exceptions thrown)

`isFull()`

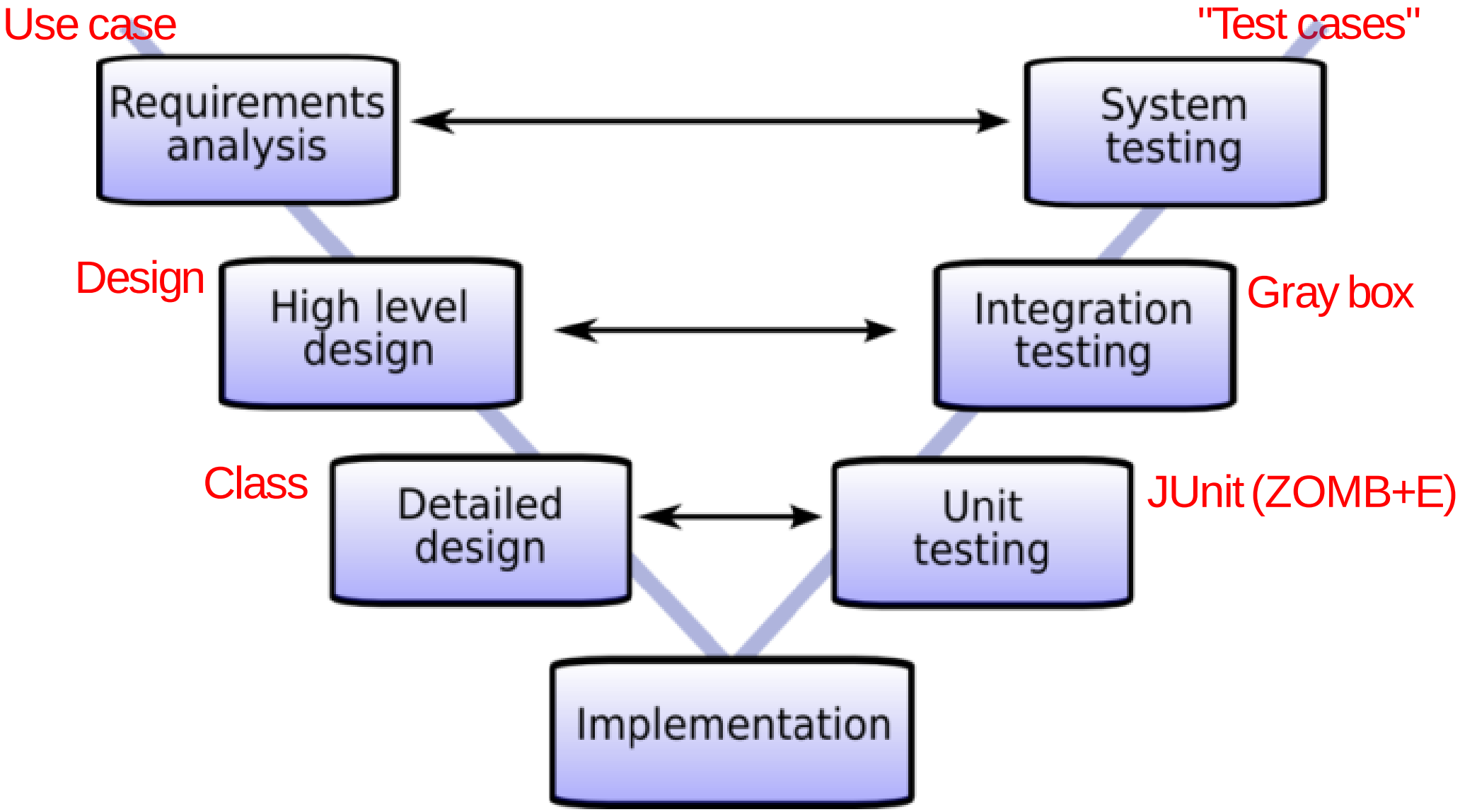
- Z: No push in a 5 element stack, isFull=false
- O: one push in a 1 element stack: isFull=true
- O: one push in a 5 element stack: isFull=false
- M: {push, push, pop, push} in a 5 element stack: isFull=false
- B: 5 push in a 5 element stack, isFull=true
- E: (no exceptions thrown)

# When to make tests?

- System testing (Test cases / Use case testing)
  - Test cases can be created as soon as you have defined fully dressed Use case descriptions
  - Test for one use case can be performed as soon as this use case has been implemented (can be sooner than other use cases are done)

- Unit testing (ZOMB+E)
  - Tests can be created as soon as the expected behavior for a class or method is known (could be before it is implemented)
    - When interfaces / method are defined and behavior agreed on
    - In Test driven development, implementing source class simultaneously with running test methods

# V-model


Use case
"Test cases"
Requirements analysis
System testing
Design
High level design
Integration testing
Gray box
Class
Detailed design
Unit testing
JUnit (ZOMB+E)
Implementation