PREGRADO

UNIDAD 2 | SOFTWARE DESIGN & PATTERNS

DESIGN VERIFICATION & VALIDATION

SI720 | Diseño y Patrones de Software

Al finalizar la unidad, el estudiante reconoce los principios y procesos para segurar la calidad del software

AGENDA

VERIFICATION AND VALIDATION

TDD

Software Testing

- Proceso de ejecutar un programa o sistema con la intención de encontrar errores (Myers)
- Proceso de encontrar diferencias entre el comportamiento esperado (requerido) y el comportamiento actual del sistema (existente).

Software Testing

- Testing es el proceso de establecer confianza en que el programa o sistema hace lo que se supone que debería hacer.

Hetzel, 1973

- Testing es el proceso de ejecutar un programa o sistema con la intención de encontrar errores.

Myers, 1979

La Prueba del Software

- Objetivo:
- Descubrir errores en el software
- Es necesario crear buenos casos de prueba (aquél que tiene una alta probabilidad de mostrar errores aún no descubiertos)
- Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Pruebas de Software para un tester

- Verificar el programa contra las especificaciones.
- Encontrar errores en el programa.
- Determinar el grado de aceptabilidad para el usuario.
- Asegurarse de que un sistema está listo para usarse.
- Ganar confidencia de que el programa funciona.
- Mostrar que un programa funciona correctamente.
- Demostrar que los errores no están presentes.
- Entender los límites del rendimiento.
- Aprender lo que el sistema no puede hacer.
- Evaluar las capacidades de un sistema.
- Verificar la documentación.
- Convencerse a uno mismo de que el trabajo ya está terminado.

La Importancia de la Definición de Testing

- La mayoría de la gente tiene una visión incorrecta de lo que es testing, y esa es la principal causa de una mala prueba del software.

- Si la meta es demostrar que el programa no tiene errores, entonces subconscientemente estamos jalados hacia esa meta y tendemos a seleccionar datos de prueba que tengan una baja probabilidad de hacer que el programa falle.

Principios de la prueba

- A todas las pruebas se les debería hacer un seguimiento hasta los requisitos del cliente.
- Las pruebas deberían planificarse mucho antes de que empiecen.
- El principio de Pareto es aplicable a la prueba de software. (El 80% de los errores descubiertos con las pruebas surgen de hacerle seguimiento sólo al 20% de todos los módulos del programa).
- Las pruebas empiezan por lo pequeño y progresan hasta lo grande.
- No son posibles las pruebas exhaustivas.
- Para ser más efectivas, las pruebas deberían ser conducidas por un equipo independiente.

Enfoques de Prueba

1. PRUEBAS DE CAJA NEGRA

Conociendo la función para la que fue diseñado, se hacen pruebas que demuestren que cada función es operativa y al mismo tiempo se buscan errores en cada una.

Pruebas sobre la interfaz del software

2. PRUEBAS DE CAJA BLANCA

Conociendo el funcionamiento del producto, se desarrollan pruebas que aseguren "que todas las piezas encajan", que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado adecuadamente.

Examen minucioso de detalles procedimentales

Pruebas de Caja Blanca

Casos de prueba que:

- Garanticen que se ejecuten por lo menos una vez TODOS los caminos, independientemente de cada módulo.
- Ejecuten todas las decisiones lógicas por sus vertientes Cierto y Falso.
- Ejecuten todos los ciclos en sus límites y límites operacionales.
- Ejecuten las estructuras internas de datos para asegurar su validez.

Pruebas de Caja Negra

- Permiten obtener conjuntos de condiciones de entrada que ejecuten todos los requisitos funcionales de un programa.

- Las pruebas de caja negra NO son una alternativa a las técnicas de prueba de caja blanca. Es un enfoque complementario.

- Las pruebas de caja negra intentan hallar errores tales como:

    - Funciones incorrectas o ausentes.

    - Errores de interfaz.

    - Errores en estructuras de datos o en accesos a BD externas.

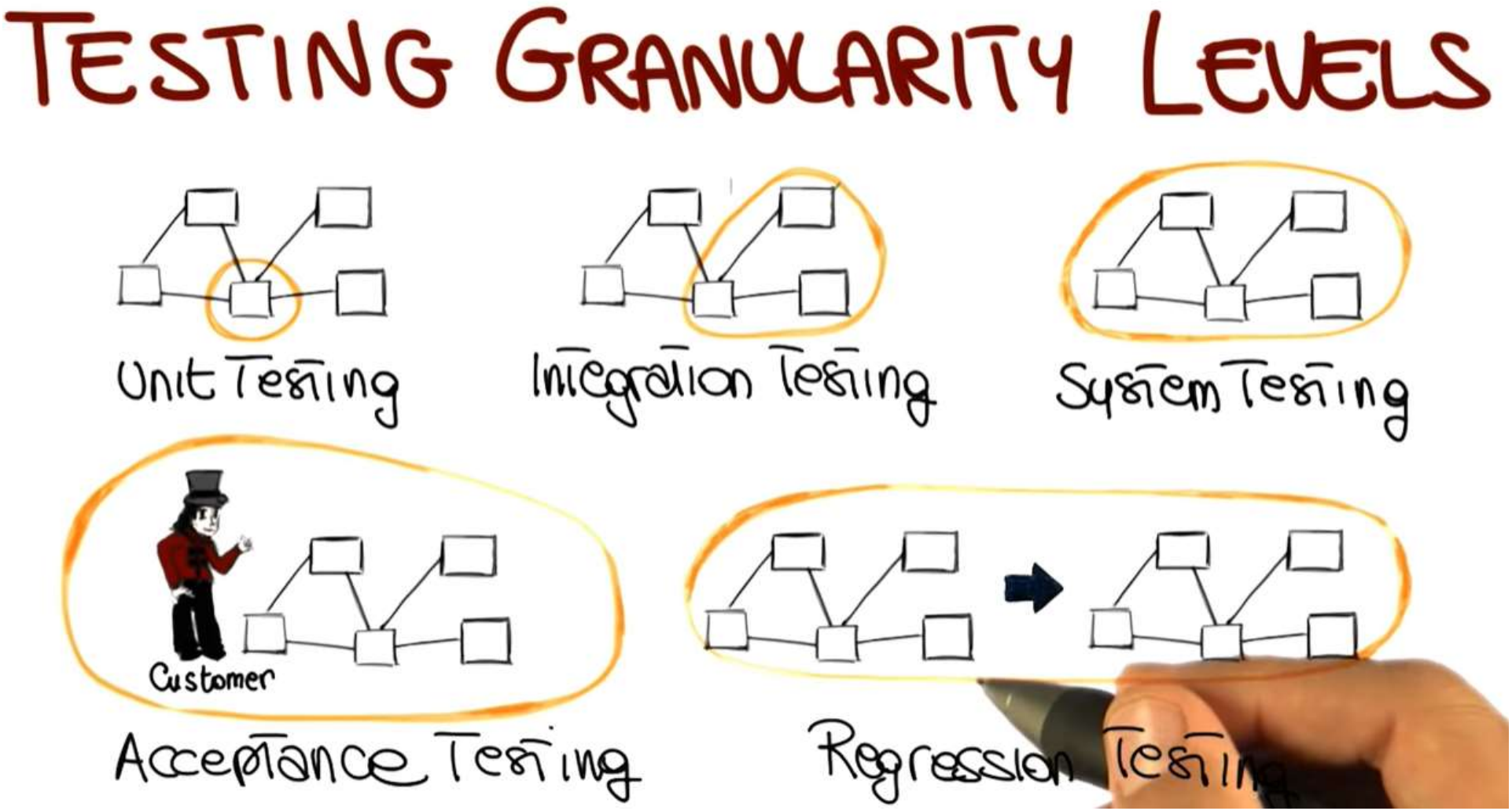
    - Errores de rendimiento.

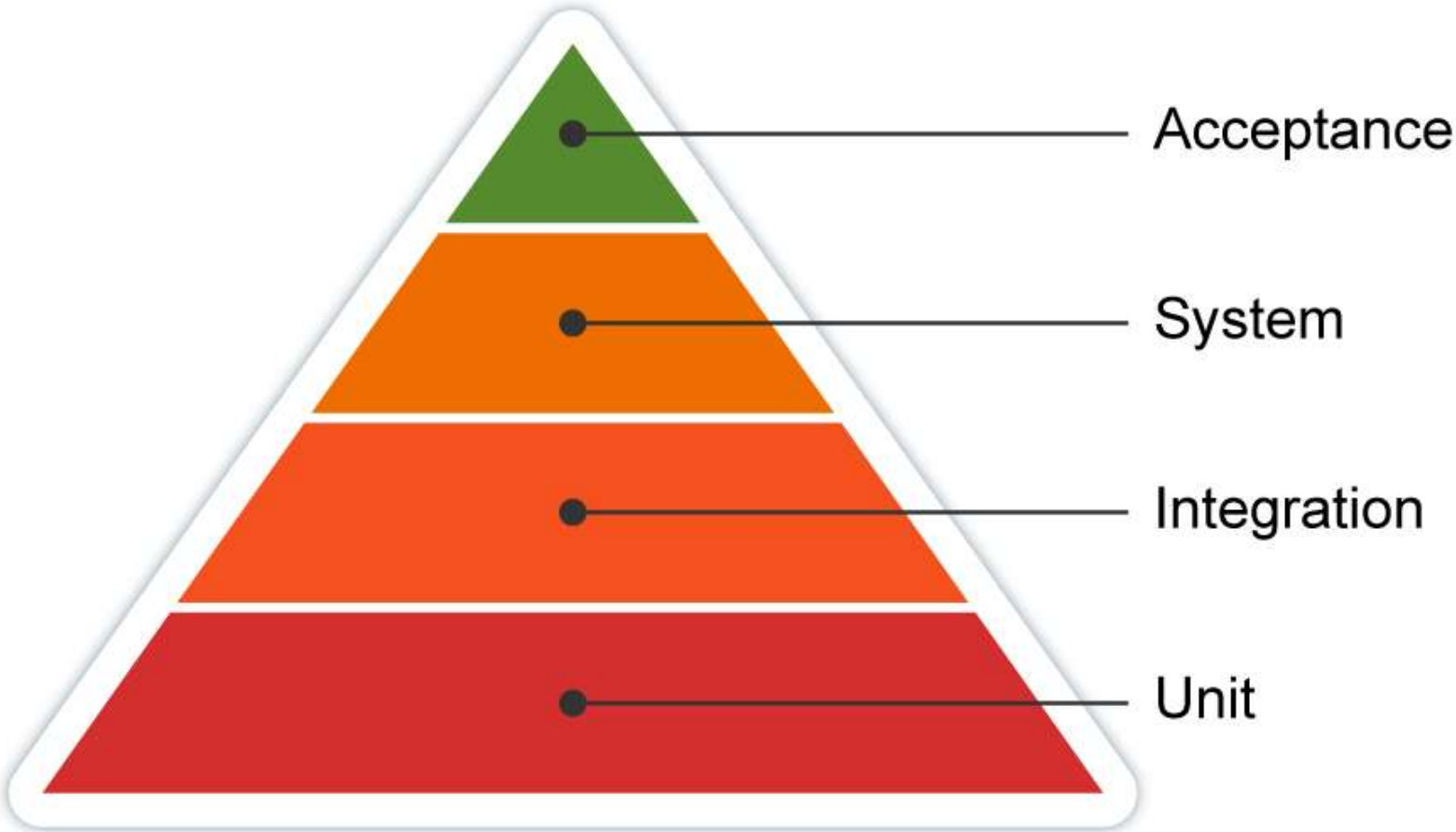
    - Errores de inicialización y de terminación.

Niveles de Testing

- Pruebas Unitarias
  - Pruebas de programas individuales conformen se van escribiendo.
- Pruebas de Sistema
  - Pruebas de grupos de programas integrados.
- Pruebas de Aceptación
  - Pruebas para verificar que el programa esta listo para usarse.

Granularidad de las Pruebas




Acceptance
System
Integration
Unit

Unit Tests

- Evalúan el correcto funcionamiento de un módulo (función) de código).
- Características
  - Automatizable
  - Completas
  - Reutilizables
  - Independientes

Integration Tests

- Técnica sistemática para construir la estructura del programa mientras que al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción.
- Se realizan dentro del ámbito de desarrollo de software y posterior a las pruebas unitarias.
- Se pruebas varios elementos unitarios que participan en un proceso.

Integration Types

Big Bang

Combina todos los módulos por anticipado, se prueba todo el producto.

Incremental

Se desarrollan módulos pequeños y funcionales, donde los errores son más fáciles de aislar y corregir.

System Testing

Buscan probar a la aplicación (o sistema) como un todo.

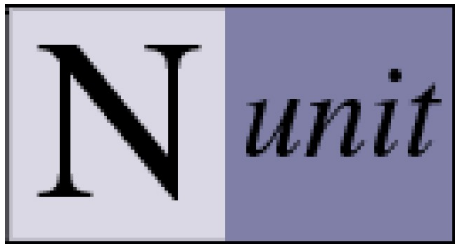
Están basadas en functional & non-functional requirements.

Se realizaran Testing environment.

Acceptance Testing

Validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista funcional y no funcional.

Toolset



MockServer

Definiciones Básicas

- De acuerdo al estándar IEEE/ANSI, 1990 [Std 610.12-1990]
  - Mistake: Una acción humana que produce un resultado incorrecto. (Equivocación?)
  - Fault: Un paso, proceso, o definición de datos incorrecto en un programa. El resultado de un “mistake” (Potencialmente puede llevar a un “failure”). (Falta?)
  - Failure: Un resultado incorrecto. El resultado (o manifestación de la falta). (Falla?)
  - Error: La cantidad por la cuál el resultado es incorrecto.

Verificación y Validación

- Verificación, como se define por la IEEE/ANSI, es el proceso de evaluar un sistema o componente para determinar si los productos de una fase de desarrollo dada, satisfacen las condiciones impuestas al inicio de esa fase.

- Validación, como se define por la IEEE/ANSI, es el proceso de evaluar un sistema o componente durante o al final del proceso de desarrollo para determinar si se satisfacen los requerimientos especificados.

Testing = Verificación + Validación

Planeación de la Verificación

- Para cada tipo de verificación (requerimientos, diseño funcional, diseño interno, código) se debe establecer:
  - La actividad de verificación que será realizada.
  - Los métodos utilizados (Walk-through, inspecciones, etc.)
  - Las áreas específicas del producto de trabajo que serán y que no serán verificadas.
  - Los riesgos asociados con cualquier área que no será verificada.
  - Asignar prioridades a las áreas del producto de trabajo que serán verificadas.
  - Recursos, calendarizaciones, instalaciones, herramientas y responsabilidades.

Planeación de la Validación

- Para cada actividad de validación se debe establecer:
  - Los métodos de prueba.
  - Las pruebas que serán automatizadas.
  - Las herramientas de pruebas.
  - El software de apoyo.
  - La administración de la configuración.
  - Los riesgos.
  - Recursos, calendarizaciones, instalaciones, y responsabilidades.

Pruebas por Revisiones

- **¿Qué es una revisión?**
  - Es cualquiera de una variedad de actividades que involucran:
    - Evaluaciones de cuestiones técnicas.
    - Evaluación de rendimiento por compañeros de trabajo.
- **Objetivos de una Revisión**
  - El objetivo de cualquier revisión es obtener información confiable acerca del estado o calidad del trabajo.

Tipos de Revisiones

- Requerimientos
- Especificaciones
- Diseño
- Codificación
- Procedimientos
- Documentación
- Instalación
- Implementación
- Diseño de Pruebas
- Procedimientos de Pruebas
- Planes de Prueba

Revisiones Formales vs Informales

- Las revisiones informales son cuando preguntamos a nuestros compañeros por su opinión.
- Las revisiones formales son cuando un grupo de personas es responsable de una evaluación exacta y de producir un reporte por escrito de los resultados.
- Las revisiones formales son las que se pueden utilizar como una técnica de pruebas.

Revisiones de Productos de Pruebas

- Plan de Pruebas
- Especificaciones del Diseño de Pruebas
- Procedimientos de Pruebas
- Casos de Pruebas
- Reportes de Pruebas

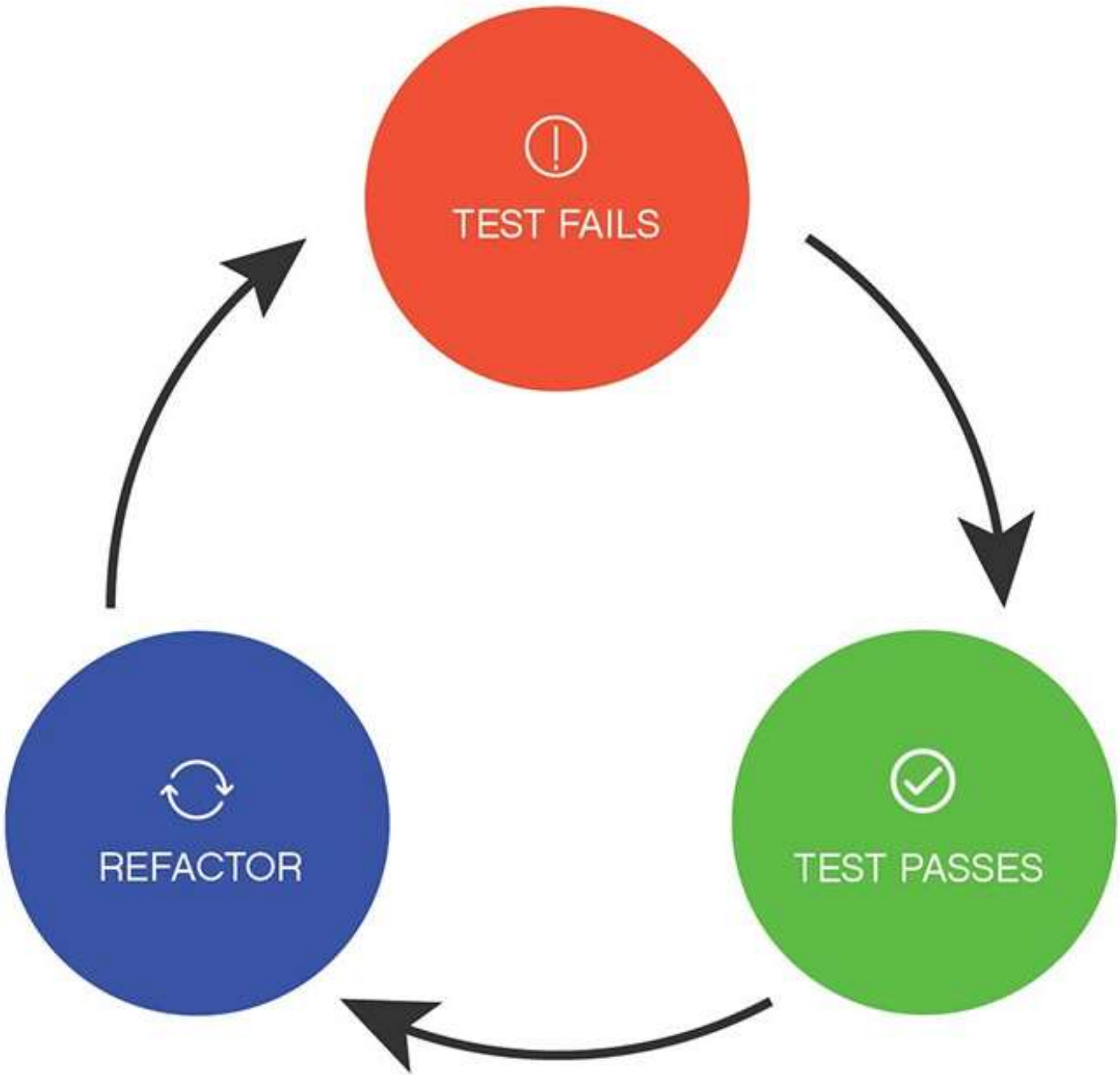
TDD
Test-Driven Development

Test-Driven Development

Defines a process in which the developer writes tests before writing code. Follows a cycle of Red > Green > Refactor.

Red: Write a test and see that it fails.
Green: Write the code to make the test pass.
Refactor: Review the code and tests and make changes to simplify them without breaking any of the working tests.

Repeat this cycle, completing all requirements.
Generally a developer-only practice.

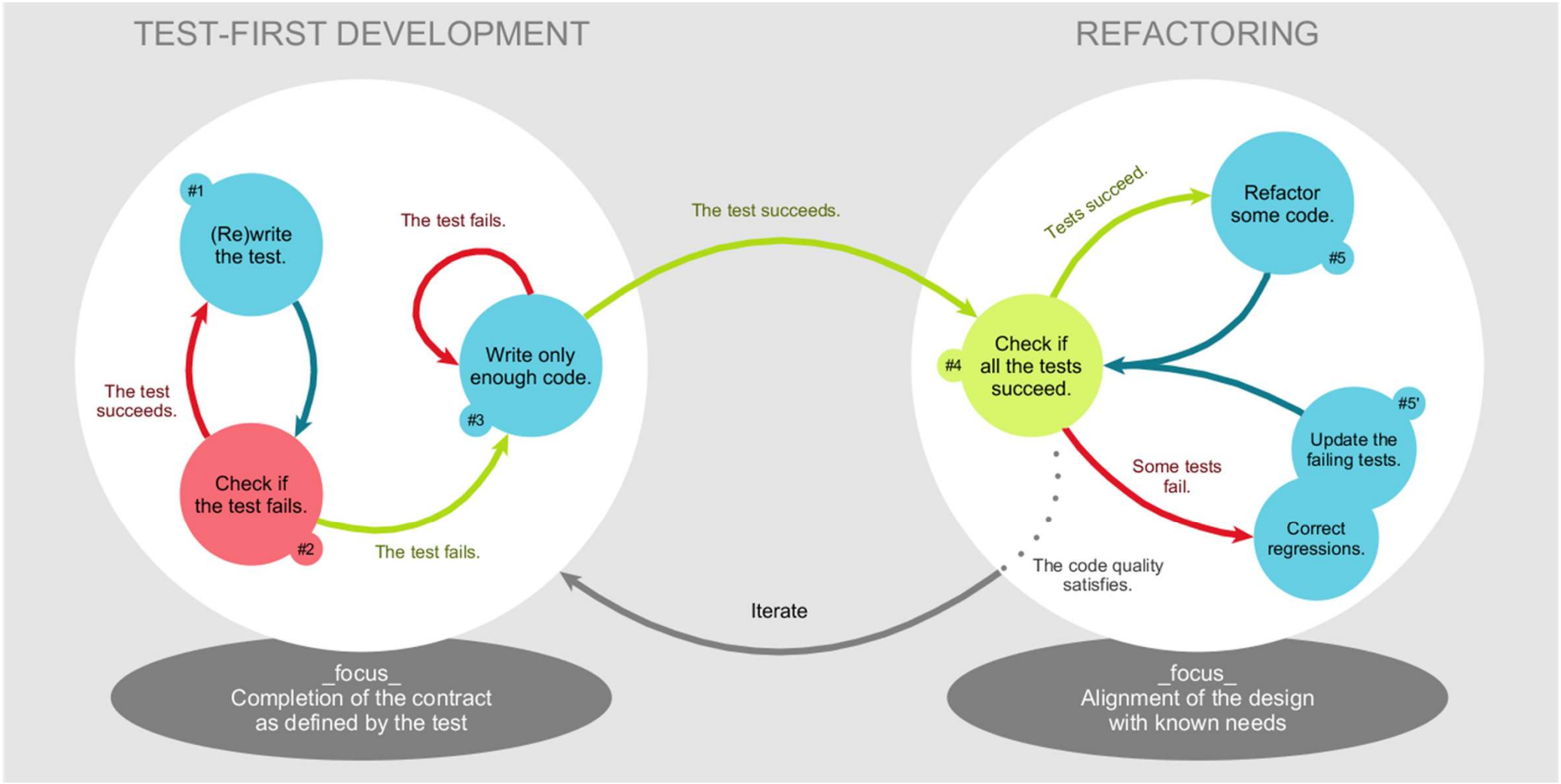

TEST FAILS
TEST PASSES
REFACTOR

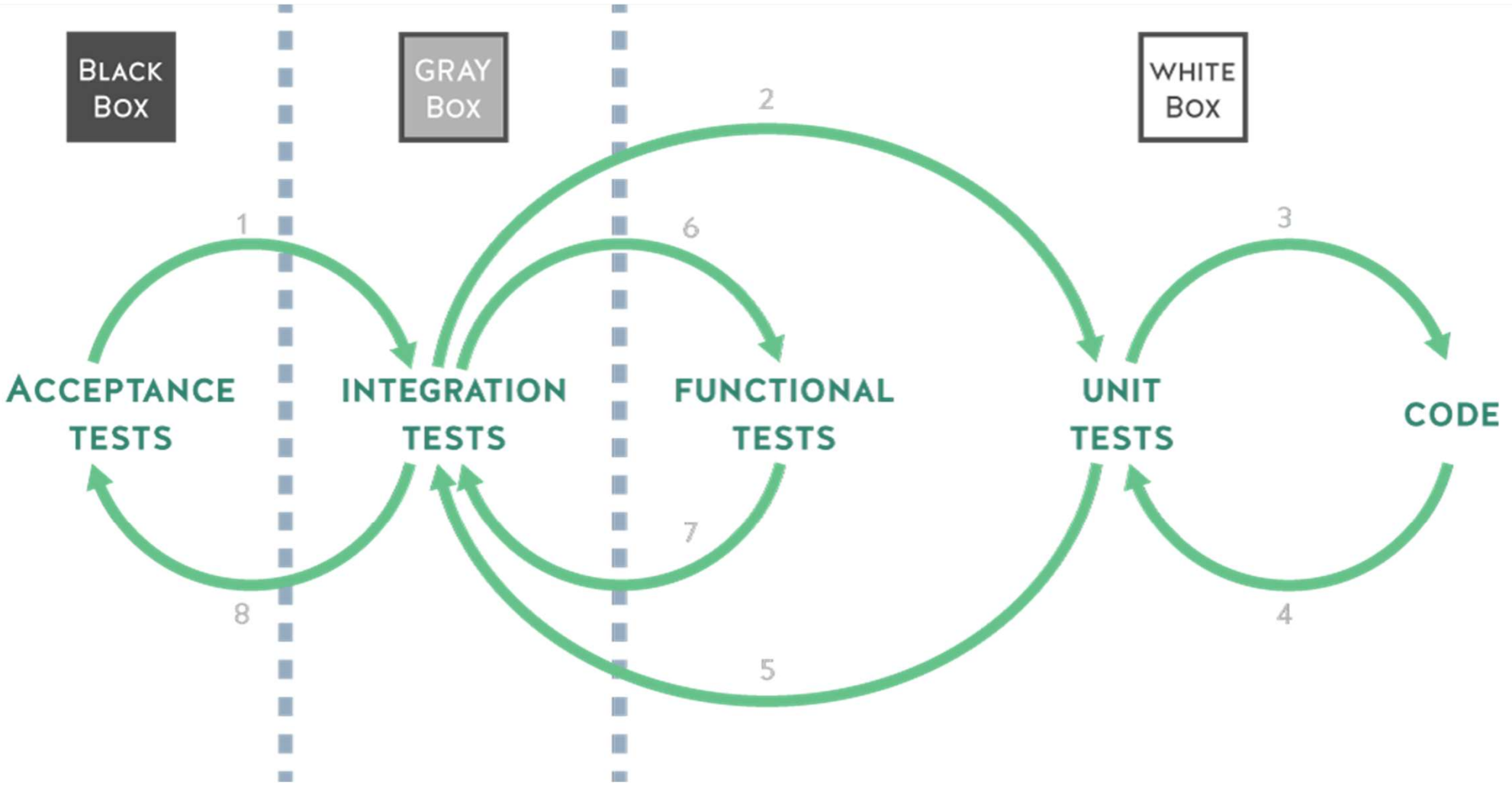
Test-Driven Development

- Desarrollo dirigido por pruebas:
  - NO es un método de testing, sino de desarrollo
  - NO reemplaza a las pruebas de performance, rendimiento, ni usabilidad
  - El objetivo es: “Código limpio que funciona”
  - Escribir los tests antes que el código, y refactorizar incrementalmente

Rojo, Verde, Refactorizar

- Escribimos la prueba, y la dejamos **fallar**.
- Código pecaminoso para **pasar** (¡progreso!).
- Implementación completa, **pasando** la prueba.
- **Refactorizamos**.
- Las pruebas son nuestro cinturón de seguridad.


TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
The test succeeds.
Write only enough code.
#3
The test succeeds.
Check if the test fails.
#2
The test fails.
Tests succeed.
Refactor some code.
#5
Check if all the tests succeed.
#4
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
focus
Completion of the contract as defined by the test
focus
Alignment of the design with known needs


BLACK BOX
GRAY BOX
WHITE BOX
ACCEPTANCE TESTS
INTEGRATION TESTS
FUNCTIONAL TESTS
UNIT TESTS
CODE
1
2
3
4
5
6
7
8

Beneficios de TDD

- No hay código sin pruebas asociadas
- El código se origina y permanece sólido
- Las pruebas perduran
- Las pruebas son documentación
- Efecto psicológico

RESUMEN

Recordemos

Design Verification & Validation

TDD

PREGRADO

Ingeniería de Software

Escuela de Ingeniería de Sistemas y Computación | Facultad de Ingeniería

UPC
Universidad Peruana
de Ciencias Aplicadas

Prolongación Primavera 2390,
Monterrico, Santiago de Surco
Lima 33 - Perú
T 511 313 3333
https://www.upc.edu.pe

exígete, innova