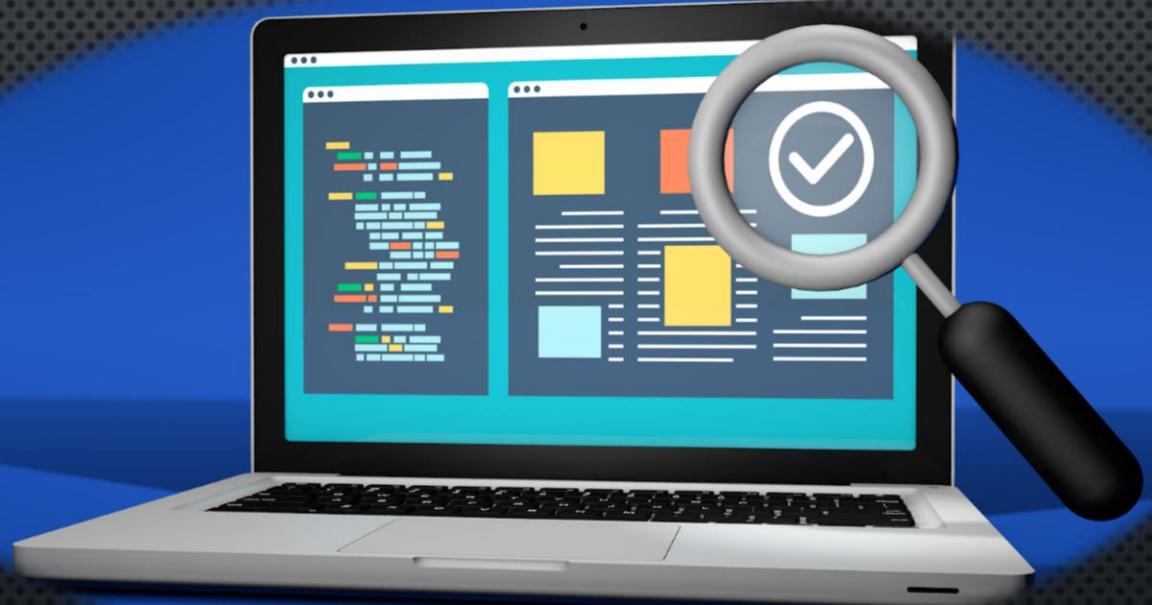


# METODOLOGÍAS DE PRUEBAS DE SISTEMAS



**Materia:** Metodología de Pruebas de Software

**Profesor:** Ing. Pablo Andrés Pérez

# TEMARIO DE LA CLASE

- *PROCESO DEL CICLO DE VIDA DE DESARROLLO DE SOFTWARE*
- *REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES*
- *MODELO V*
- *PROCESO DE PRUEBAS*
- *VERIFICACIÓN Y VALIDACIÓN*
- *DEFINICIONES IMPORTANTES*
- *INTRODUCCIÓN AL TRABAJO PRACTICO DE LA MATERIA*

# CICLO DE VIDA DEL DESARROLLO DE SOFTWARE

- EL CICLO DE VIDA DEL DESARROLLO DE SOFTWARE (SDLC) ES UN PROCESO RENTABLE Y EFICIENTE EN TÉRMINOS DE TIEMPO EMPLEADO POR LOS EQUIPOS DE DESARROLLO PARA DISEÑAR Y CREAR SOFTWARE DE ALTA CALIDAD.
- EL OBJETIVO ES *MINIMIZAR LOS RIESGOS DEL PROYECTO POR MEDIO DE UNA PLANIFICACIÓN ANTICIPADA QUE PERMITA QUE EL SOFTWARE CUMPLA LAS EXPECTATIVAS DEL CLIENTE* DURANTE LA FASE DE PRODUCCIÓN Y POSTERIORMENTE.
- ESTA METODOLOGÍA ESTABLECE UNA SERIE DE PASOS QUE DIVIDEN EL PROCESO DE DESARROLLO DE SOFTWARE EN TAREAS QUE SE PUEDEN ASIGNAR, COMPLETAR Y MEDIR.

# CICLO DE VIDA DEL DESARROLLO DE SOFTWARE

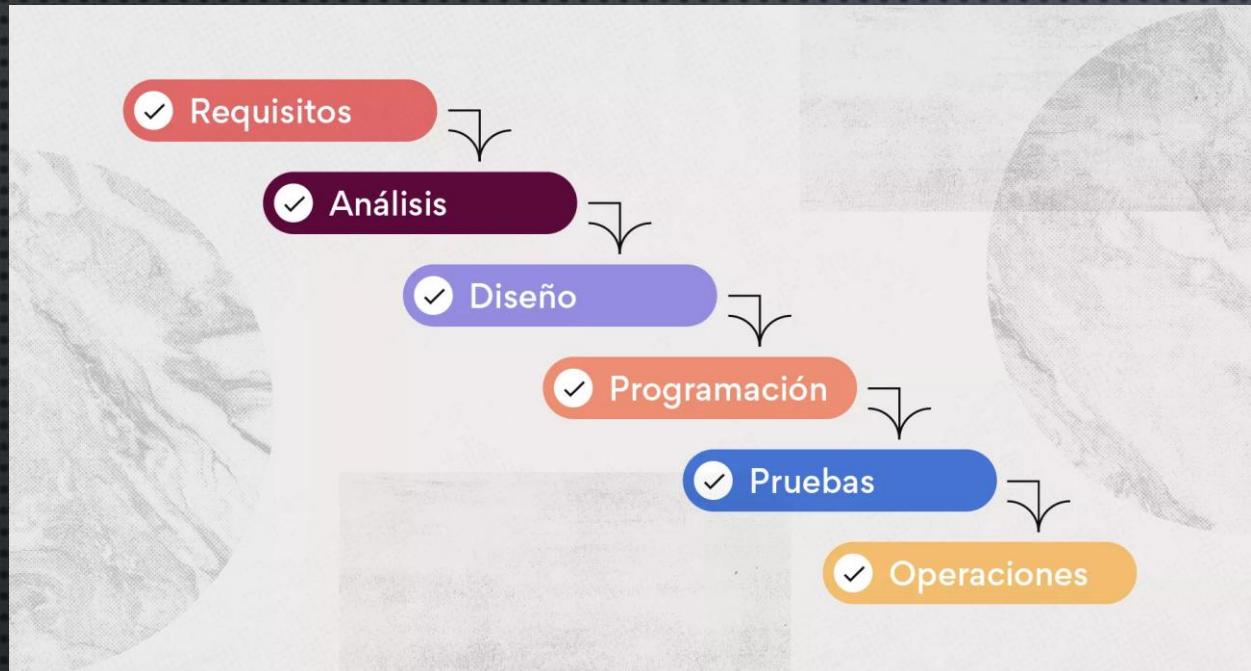
- UN MODELO DE CICLO DE VIDA DEL DESARROLLO DE SOFTWARE PRESENTA DE MANERA CONCEPTUAL Y ORGANIZADA UN MODELO PARA PERMITIR QUE LAS ORGANIZACIONES LO IMPLEMENTEN.
- DIFERENTES MODELOS DISPONEN LAS FASES EN DIVERSOS ÓRDENES CRONOLÓGICOS PARA OPTIMIZAR EL CICLO DE DESARROLLO.
- ALGUNOS MODELOS CONOCIDOS SON:
  - *MODELO EN CASCADA*
  - *MODELO EN ESPIRAL*
  - *MODELO ITERATIVO*
  - *MODELO AGIL*

# MODELO ESTANDAR

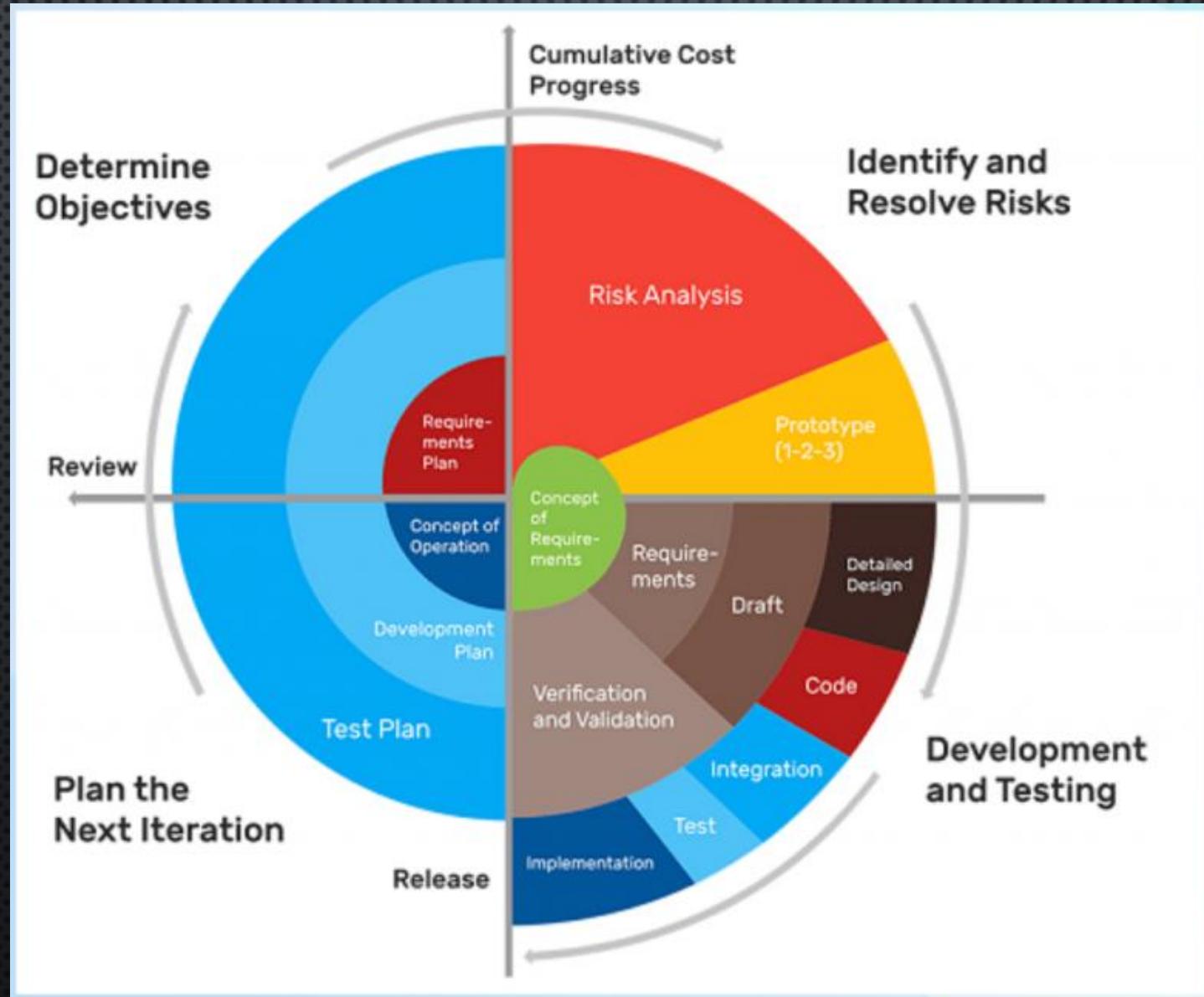
## Las 7 fases del SDCL



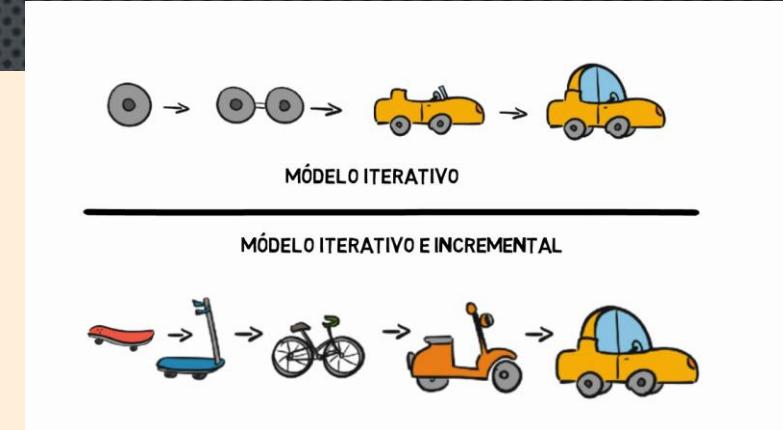
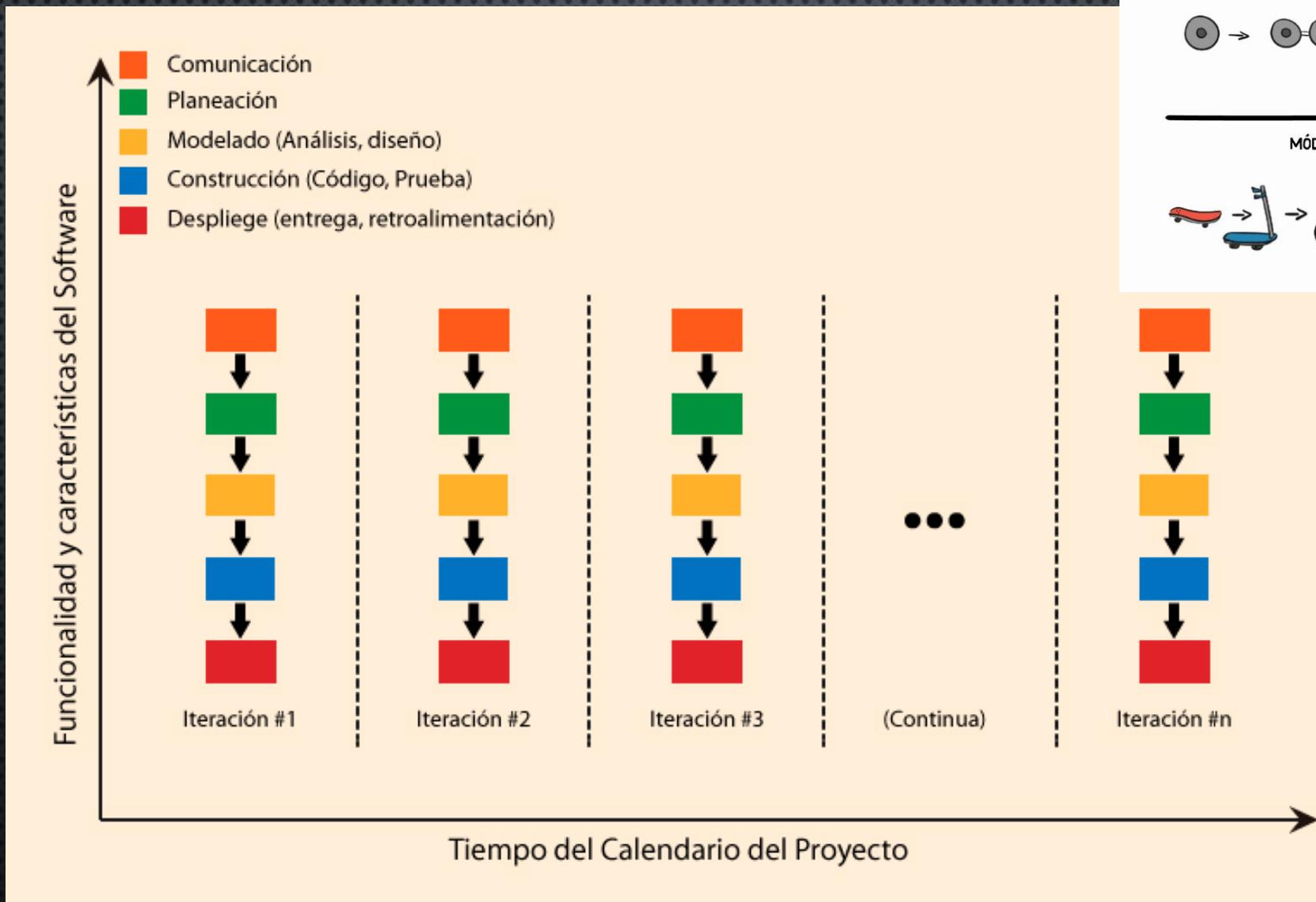
# MODELO CASCADA



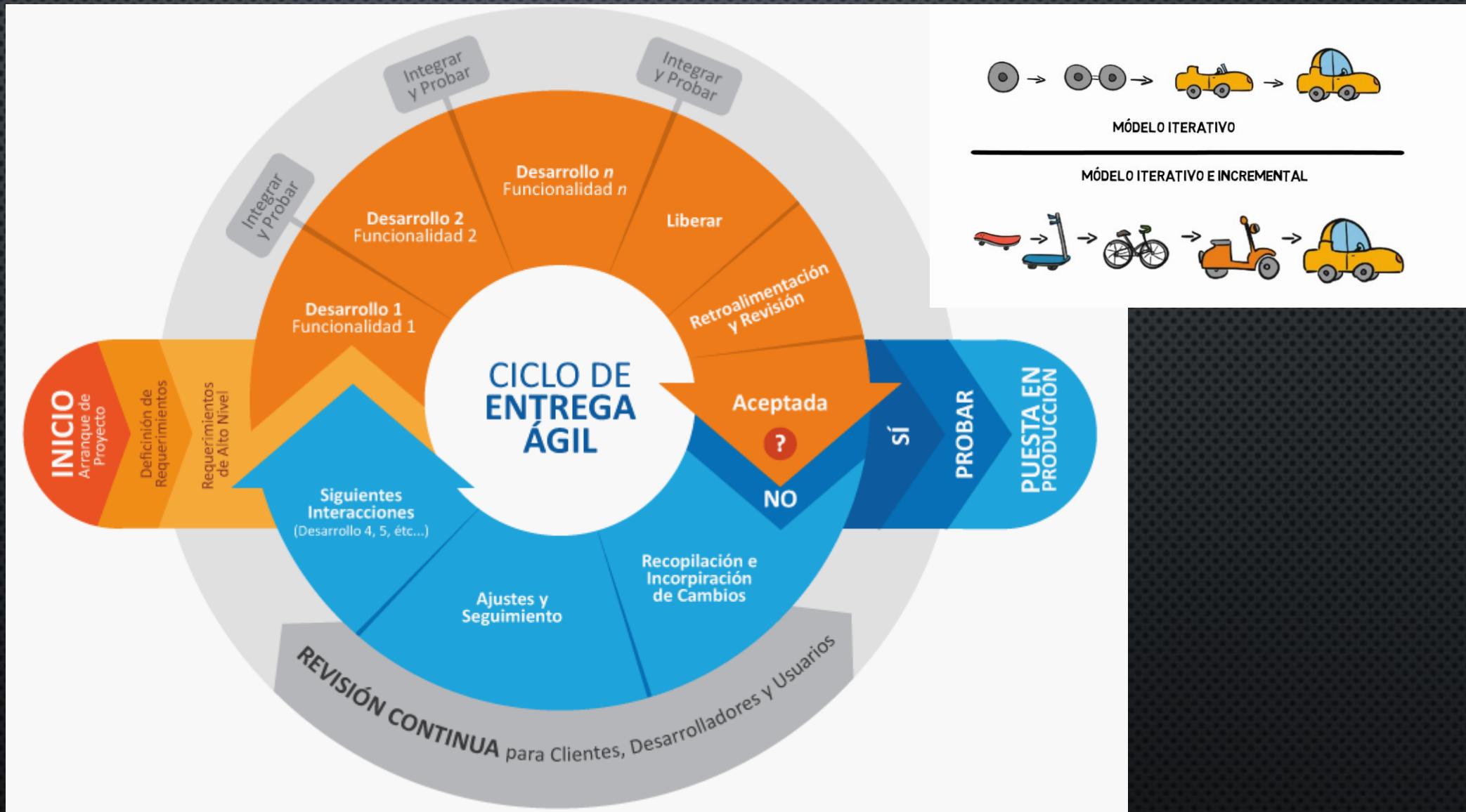
# MODELO EN ESPIRAL



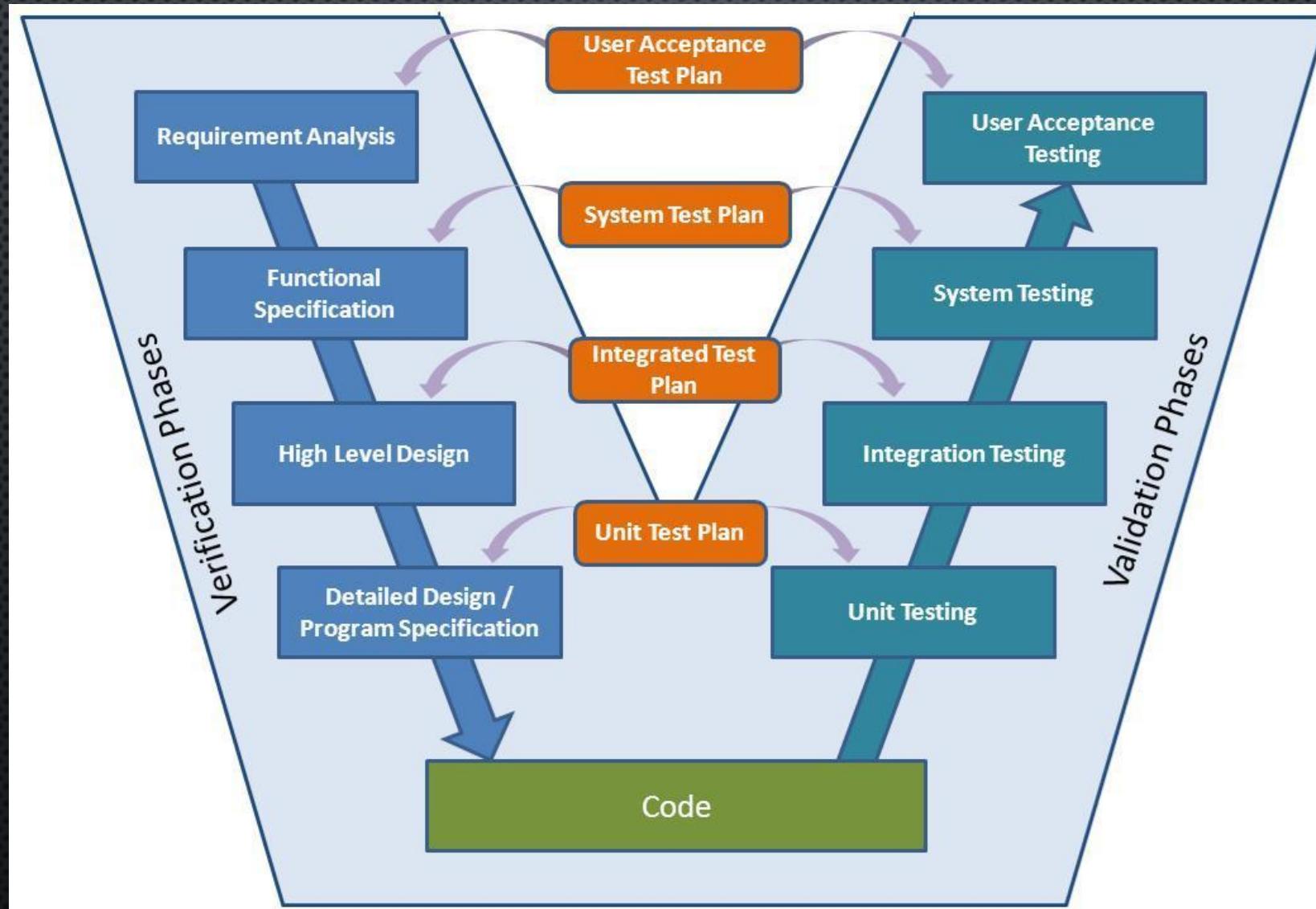
# MODELO ITERATIVO

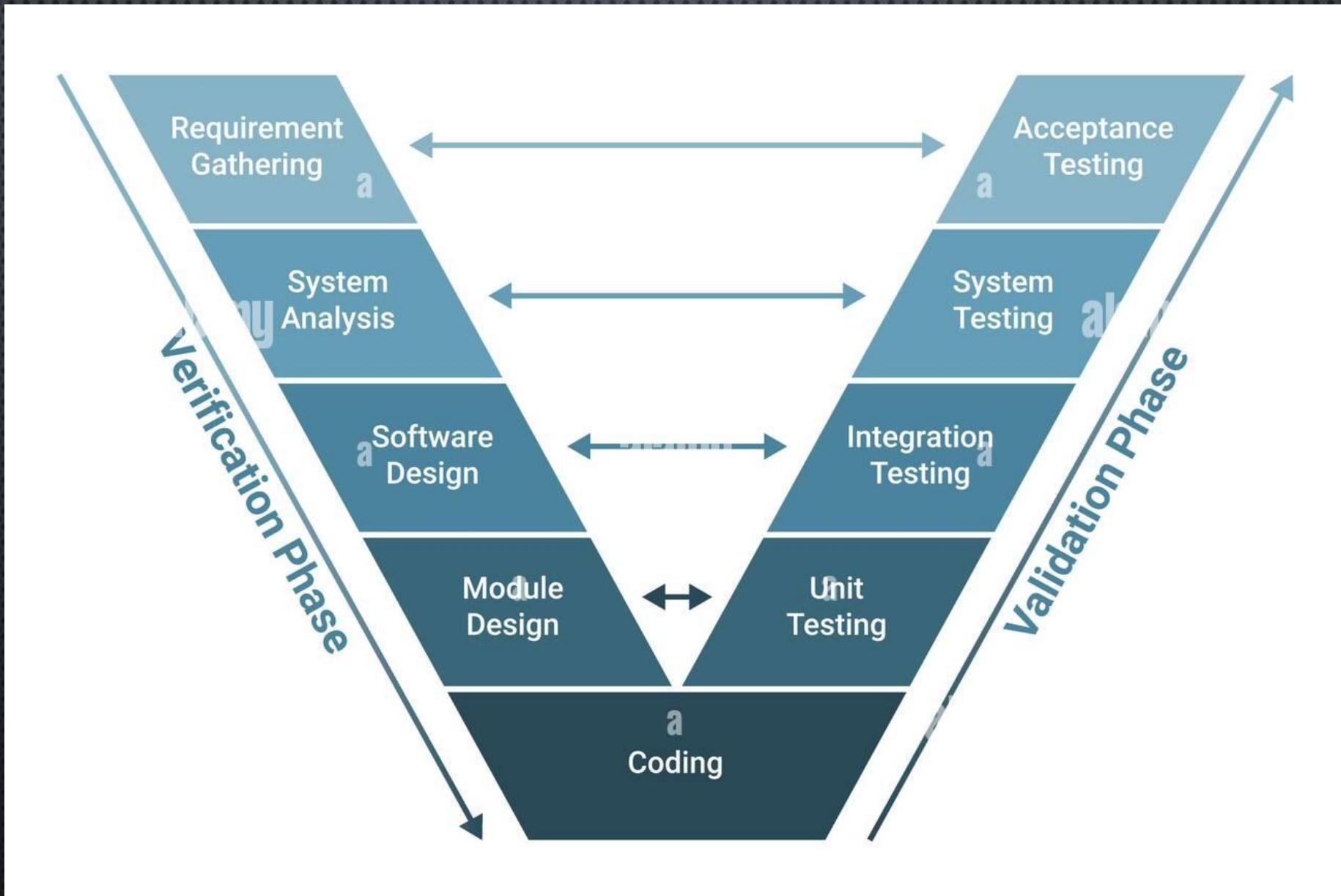


# MODELO ÁGIL

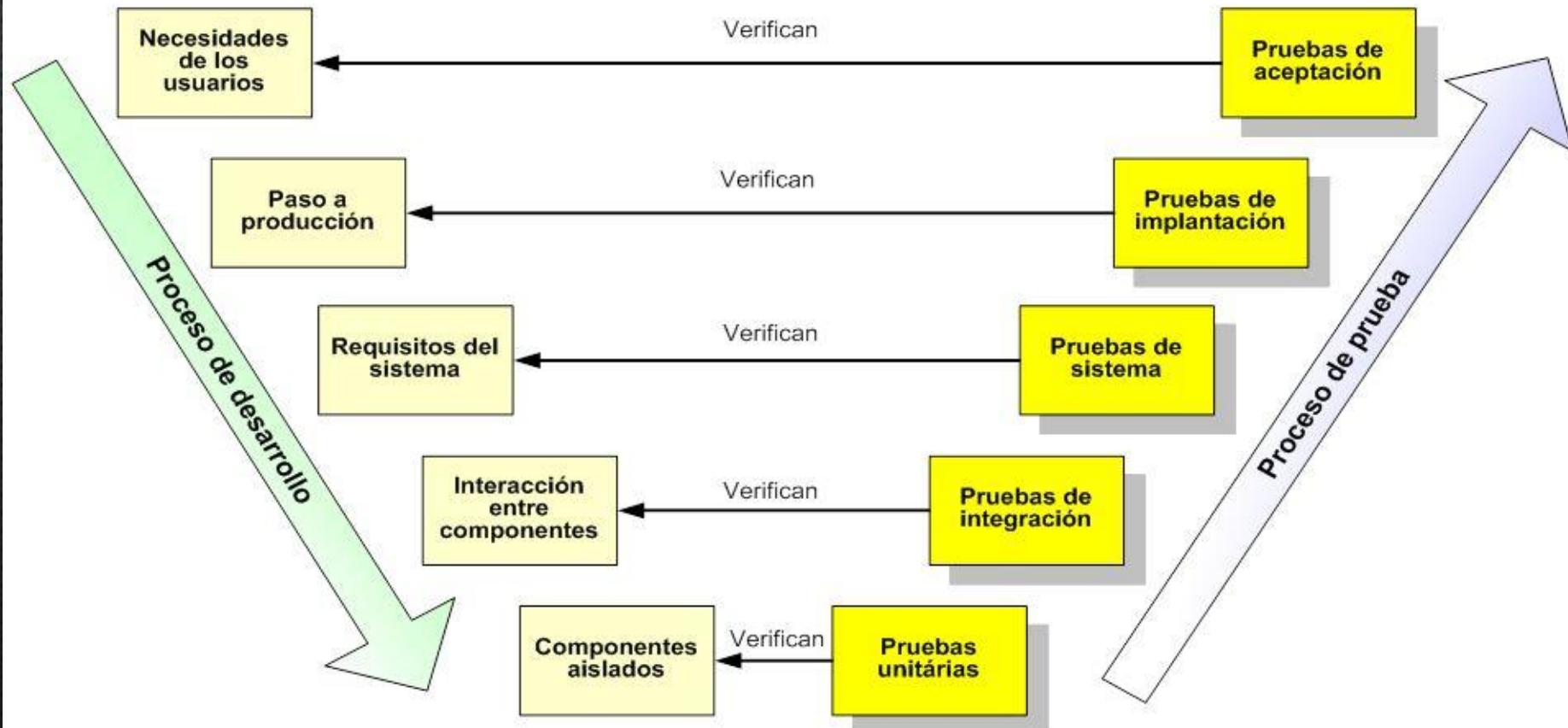


# MODELO EN V



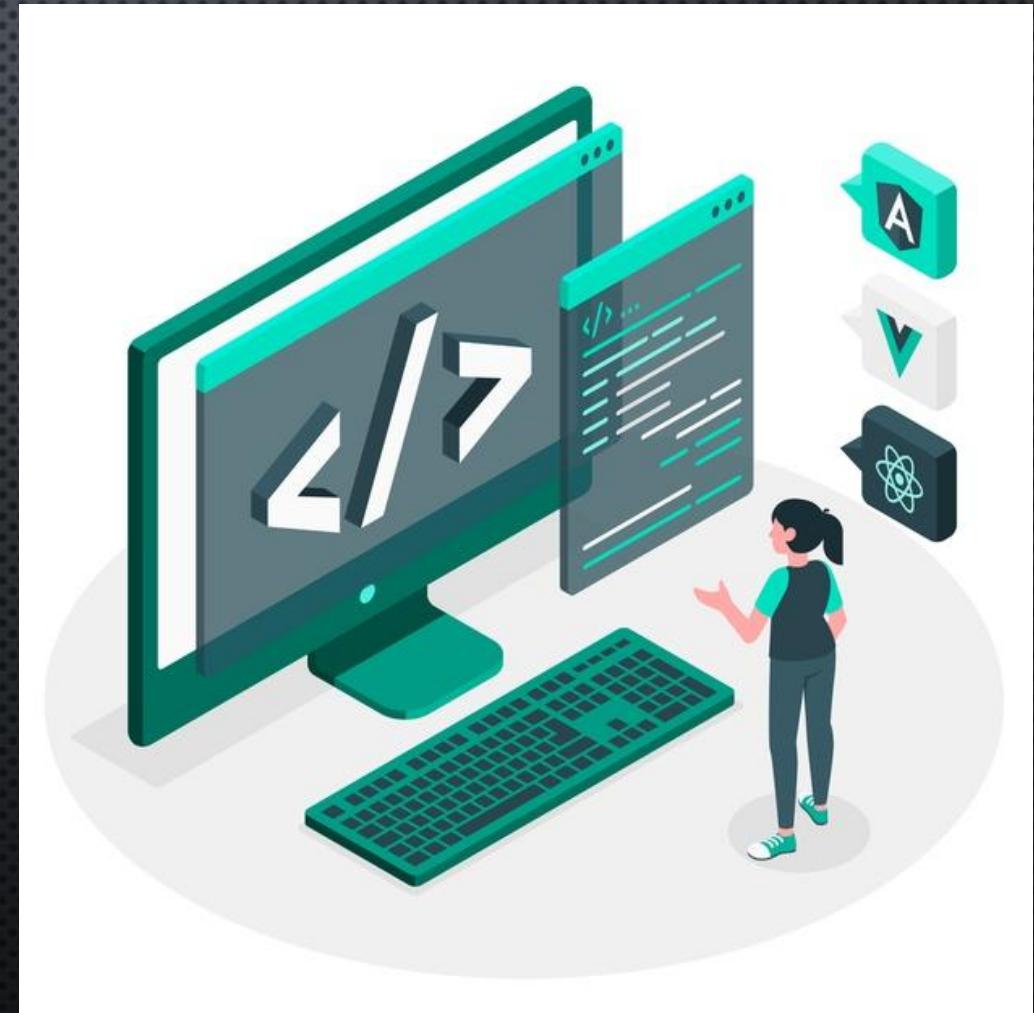


# Niveles de prueba



# REQUERIMIENTOS FUNCIONALES

- LOS REQUERIMIENTOS FUNCIONALES SON DESCRIPCIONES DETALLADAS DE LAS FUNCIONES Y CARACTERÍSTICAS QUE UN SISTEMA DEBE TENER PARA CUMPLIR CON SUS OBJETIVOS Y SATISFACER LAS NECESIDADES DE SUS USUARIOS.
- ESPECIFICAN CÓMO EL SISTEMA DEBE COMPORTARSE EN TÉRMINOS DE ENTRADAS, PROCESAMIENTO Y SALIDAS, ASÍ COMO LAS INTERACCIONES CON LOS USUARIOS Y OTROS SISTEMAS.
- SON ESENCIALES PARA GUIAR EL DISEÑO, DESARROLLO, IMPLEMENTACIÓN Y PRUEBA DEL SISTEMA, ASEGURANDO QUE CUMPLA CON LAS EXPECTATIVAS DE LOS USUARIOS Y LAS NECESIDADES DEL NEGOCIO.
- DEBEN SER CLAROS, DETALLADOS Y VERIFICABLES, LO QUE SIGNIFICA QUE SE PUEDAN MEDIR Y EVALUAR PARA DETERMINAR SI EL SISTEMA CUMPLE CON ELLOS.



# REQUERIMIENTOS NO FUNCIONALES

- LOS REQUERIMIENTOS NO FUNCIONALES SON UN CONJUNTO DE CARACTERÍSTICAS Y ATRIBUTOS QUE DESCRIBEN CÓMO UN SISTEMA DEBE COMPORTARSE Y OPERAR, PERO QUE NO ESTÁN DIRECTAMENTE RELACIONADOS CON LAS FUNCIONES ESPECÍFICAS QUE REALIZA EL SISTEMA.
- A DIFERENCIA DE LOS REQUERIMIENTOS FUNCIONALES, QUE SE CENTRAN EN QUÉ HACE EL SISTEMA, LOS REQUERIMIENTOS NO FUNCIONALES SE CENTRAN EN CÓMO LO HACE Y EN CÓMO DEBE CUMPLIR CIERTOS CRITERIOS DE RENDIMIENTO, CALIDAD, SEGURIDAD Y OTROS ASPECTOS IMPORTANTES.
- ESTOS REQUISITOS SON CRUCIALES PARA GARANTIZAR QUE EL SISTEMA SEA CONFIABLE, SEGURO, EFICIENTE Y USABLE.

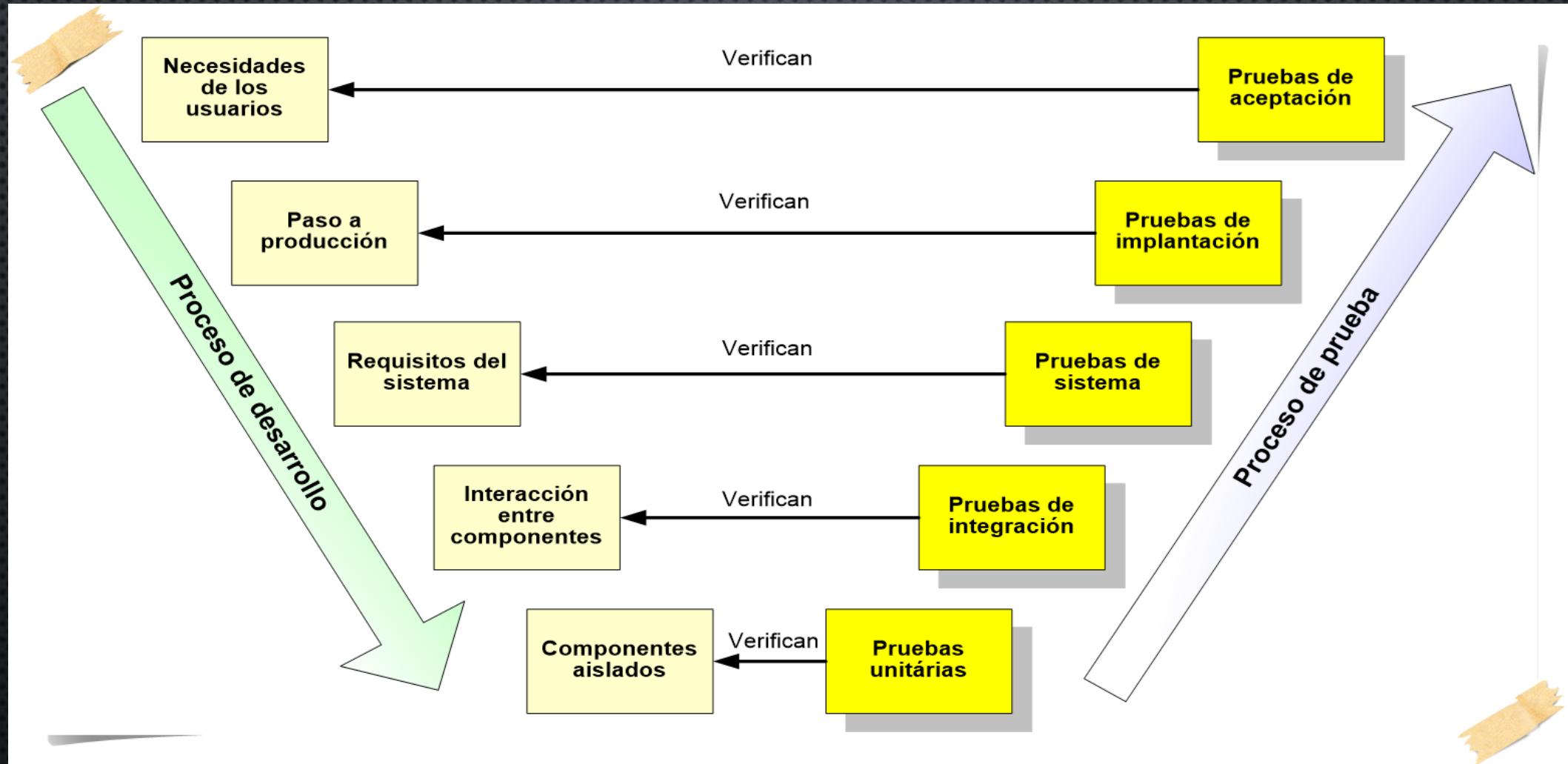


# PRACTIQUEMOS...

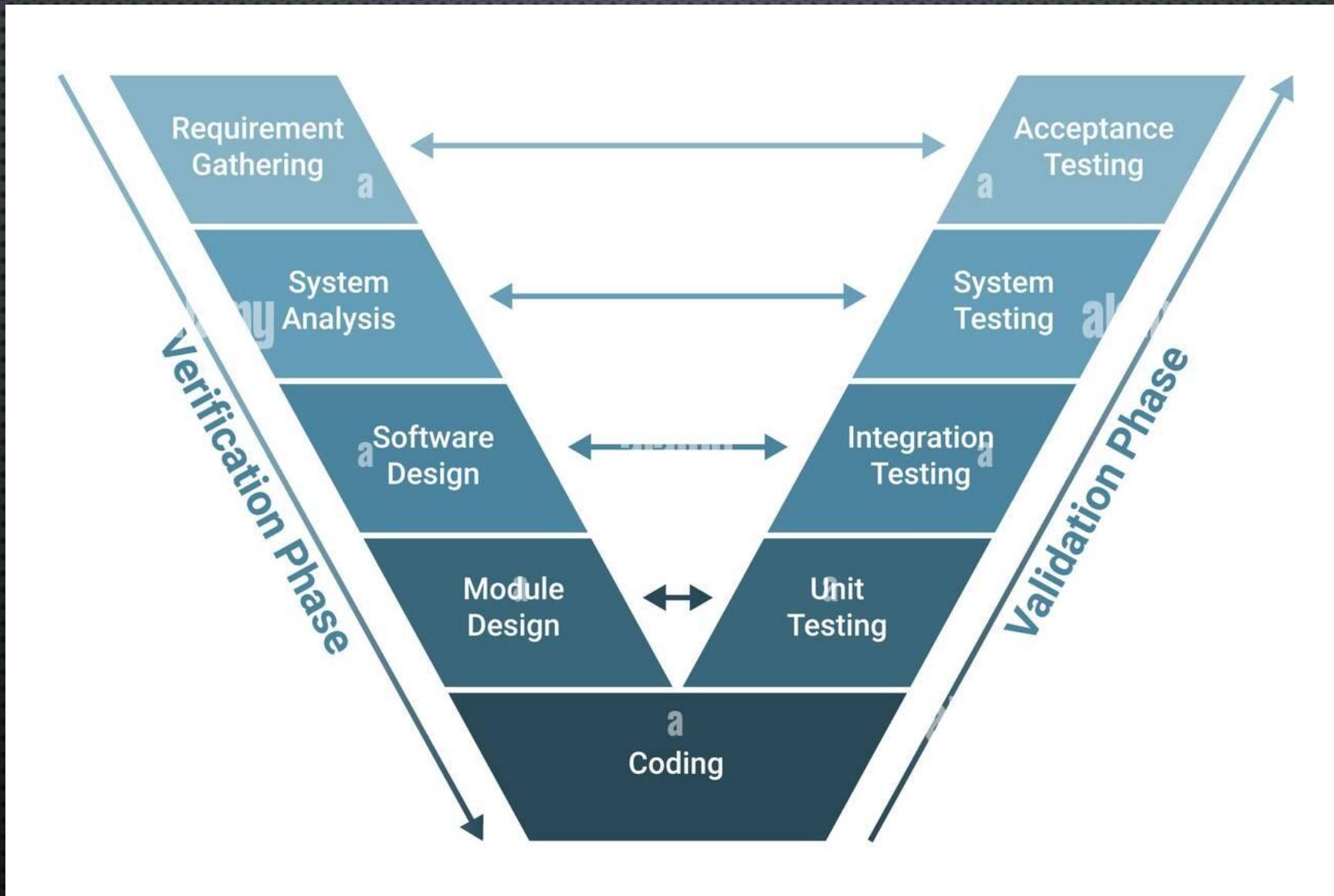
- USTED ES PARTE DEL EQUIPO DE DESARROLLO DEL IFTS18-SOFT Y RECIBE A UN CLIENTE DE UNA EMPRESA DE TRANSPORTE DE LARGA DISTANCIA QUIEN LE SOLICITA UN SOFTWARE PARA PODER GESTIONAR LA VENTA DE PASAJES Y BRINDAR SERVICIOS DE ENCOMIENDAS (SEGUIMIENTO) Y SEGUIMIENTO DE SUS UNIDADES.
- REDACTE UN “REQUERIMIENTO FUNCIONAL” Y UNO “NO FUNCIONAL” PARA EL SISTEMA Y PIENSE PARA CADA UNO COMO PROBARÍA QUE FUNCIONE CORRECTAMENTE.



# MODELO EN V Y NIVELES DE PRUEBA



# DE V&V A LAS PRUEBAS



# PROCESO DE PRUEBAS

- ¿EN QUÉ CONSISTE EL PROCESO DE PRUEBAS?

UN PROCESO DE PRUEBAS ES UNA VERIFICACIÓN DINÁMICA DEL COMPORTAMIENTO DEL SOFTWARE A PARTIR DE UN CONJUNTO FINITO DE CASOS DE PRUEBA, ES DECIR, QUE SE EJECUTA UN PROGRAMA Y MEDIANTE TÉCNICAS EXPERIMENTALES SE TRATA DE DESCUBRIR QUE ERRORES TIENE.

UNA PRUEBA CONSTA, AL MENOS, DE TRES ELEMENTOS: ACCIONES, VALORES DE PRUEBA Y RESULTADO. DEBEMOS TENER EN CUENTA QUE LOS CASOS DE PRUEBA SON FINITOS (Y CUANTOS MENOS, MEJOR), ES DECIR, TIENEN UN LÍMITE.

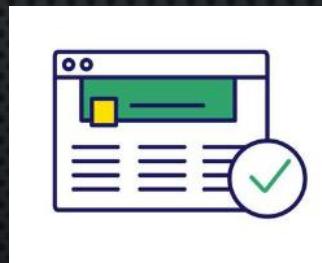
DEBEMOS TENER EN CUENTA QUE LAS PRUEBAS SÓLO PUEDEN ENCONTRAR LOS ERRORES QUE BUSCAN. POR ESTO ES TAN IMPORTANTE UN BUEN DISEÑO DE PRUEBAS.

# FUNDAMENTOS DE TESTING

- **VERIFICACIÓN:** “¿ESTOY CONSTRUYENDO CORRECTAMENTE EL PRODUCTO?”  
**EL SOFTWARE DEBE AJUSTARSE A SU ESPECIFICACIÓN.**  
**DESARROLLO LA FUNCIONALIDAD DE ACUERDO AL REQUERIMIENTO.**



- **VALIDACIÓN:** “¿ESTOY CONSTRUYENDO EL PRODUCTO CORRECTO?”  
**EL SOFTWARE DEBE HACER LO QUE EL CLIENTE QUIERE QUE HAGA.**  
**LO QUE DESARROLLE HACE LO QUE EL CLIENTE ESPERABA**



# FUNDAMENTOS DE TESTING

Pressman:

- **Verificación** se refiere al conjunto de tareas que garantizan que el software implementa correctamente una función específica.
- **Validación** es un conjunto diferente de tareas que aseguran que el software que se construye sigue los requerimientos del cliente.

■ Boehm:

**Verificación:** ¿Estamos construyendo el producto correctamente?

**Validación:** ¿Estamos construyendo el producto correcto?

# FUNDAMENTOS DE TESTING

---

Sommerville

## Verificación

- Busca comprobar que el sistema cumple con los requerimientos especificados (funcionales y no funcionales)
- ¿El software está de acuerdo con su especificación?

## Validación

- Busca comprobar que el software hace lo que el usuario espera.
- ¿El software cumple las expectativas del cliente?

# DE V&V A LAS PRUEBAS

- LA PRUEBA DEL SOFTWARE ES UN ELEMENTO DE UN CONCEPTO MÁS AMPLIO QUE SUELE DENOMINARSE VERIFICACIÓN Y VALIDACIÓN (V&V).
- VERIFICACIÓN ES EL CONJUNTO DE ACTIVIDADES QUE ASEGURAN QUE EL SOFTWARE IMPLEMENTE CORRECTAMENTE UNA FUNCIÓN ESPECÍFICA.
- VALIDACIÓN ES UN CONJUNTO DIFERENTE DE ACTIVIDADES QUE ASEGURAN QUE EL SOFTWARE CONSTRUIDO RESPETA LOS REQUISITOS DEL CLIENTE (REQUERIMIENTOS).

# DE V&V A LAS PRUEBAS

- **BOEHM LO ESTABLECE DE OTRA MANERA:**

VERIFICACIÓN: "¿ESTAMOS CONSTRUYENDO EL PRODUCTO CORRECTAMENTE?"

VALIDACIÓN: "¿ESTAMOS CONSTRUYENDO EL PRODUCTO CORRECTO?"

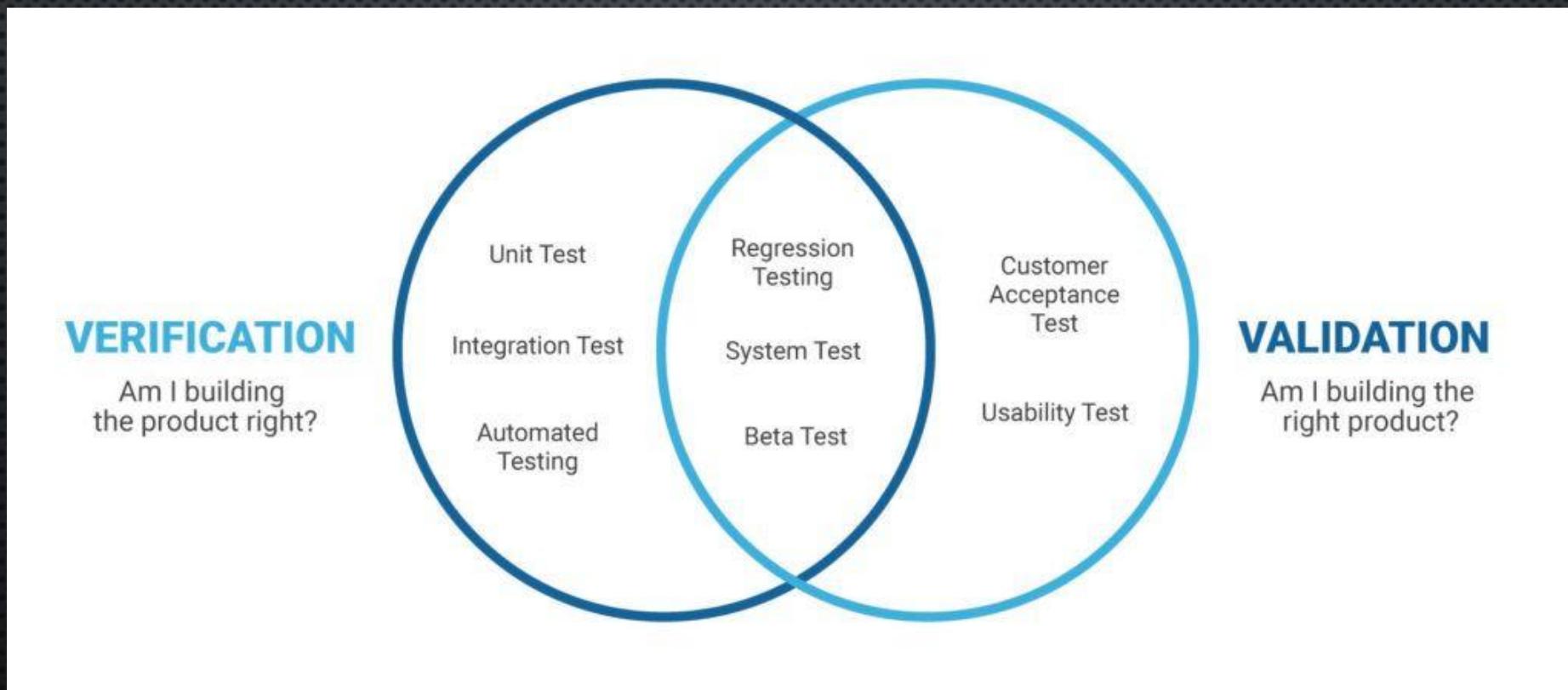
- LA VERIFICACIÓN Y LA VALIDACIÓN ABARCAN UNA AMPLIA LISTA DE ACTIVIDADES DE ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE [PRE10]: REVISIONES TÉCNICAS FORMALES, AUDITORÍAS DE CALIDAD Y DE CONFIGURACIÓN, MONITOREO DEL DESEMPEÑO, SIMULACIÓN, FACTIBILIDAD, REVISIÓN DE LA DOCUMENTACIÓN Y LA BASE DE DATOS, ANÁLISIS DE ALGORITMOS, PRUEBAS DE DESARROLLO, DE FACILIDAD DE USO, DE CALIFICACIÓN Y DE INSTALACIÓN. AUNQUE LAS ACTIVIDADES DE PRUEBA TIENEN UN PAPEL DEMASIADO IMPORTANTE EN V&V, TAMBIÉN SE NECESITAN MUCHAS OTRAS ACTIVIDADES.

# DE V&V A LAS PRUEBAS

- LAS PRUEBAS SON EL ÚLTIMO BASTIÓN PARA LA EVALUACIÓN DE LA CALIDAD Y, DE MANERA MÁS PRAGMÁTICA, EL DESCUBRIMIENTO DE ERRORES.
- PERO LAS PRUEBAS NO DEBEN CONSIDERARSE COMO UNA “RED DE SEGURIDAD”. COMO SUELE DECIRSE: “NO ES POSIBLE PROBAR LA CALIDAD. SI NO ESTÁ AHÍ ANTES DE QUE EMPIECE LA PRUEBA, NO ESTARÁ CUÁNDΟ SE TERMINE”. LA CALIDAD SE INCORPORA AL SOFTWARE EN TODO EL PROCESO DE INGENIERÍA.
- LA APLICACIÓN CORRECTA DE MÉTODOS Y HERRAMIENTAS, LAS REVISIONES TÉCNICAS FORMALES Y EFECTIVAS JUNTO CON UNA ADMINISTRACIÓN Y UNA MEDICIÓN SÓLIDAS APORTRAN LA CALIDAD, QUE SE CONFIRMA DURANTE LAS PRUEBAS.

# DE V&V A LAS PRUEBAS

- LA **PRUEBA DE SOFTWARE** ES UN ELEMENTO DE UN TEMA MÁS AMPLIO QUE USUALMENTE SE CONOCE COMO **VERIFICACIÓN Y VALIDACIÓN**



# CONCEPTOS A TENER EN CUENTA

- **BUGS:** SON DEFECTOS DEL PROGRAMA QUE SE ENCUENTRAN OPERANDO EL MISMO, YA SEA DURANTE LA PRUEBA O DURANTE SU USO.
- LOS BUGS PROVIENEN DE LOS DEFECTOS, PERO NO TODOS LOS DEFECTOS CAUSAN BUGS (ALGUNAS ESTÁN LATENTES PERO NUNCA SE ENCUENTRAN).
- LOS DEFECTOS PUEDEN ENCONTRARSE MUCHAS VECES, DANDO COMO RESULTADO MÚLTIPLES BUGS POR DEFECTO.



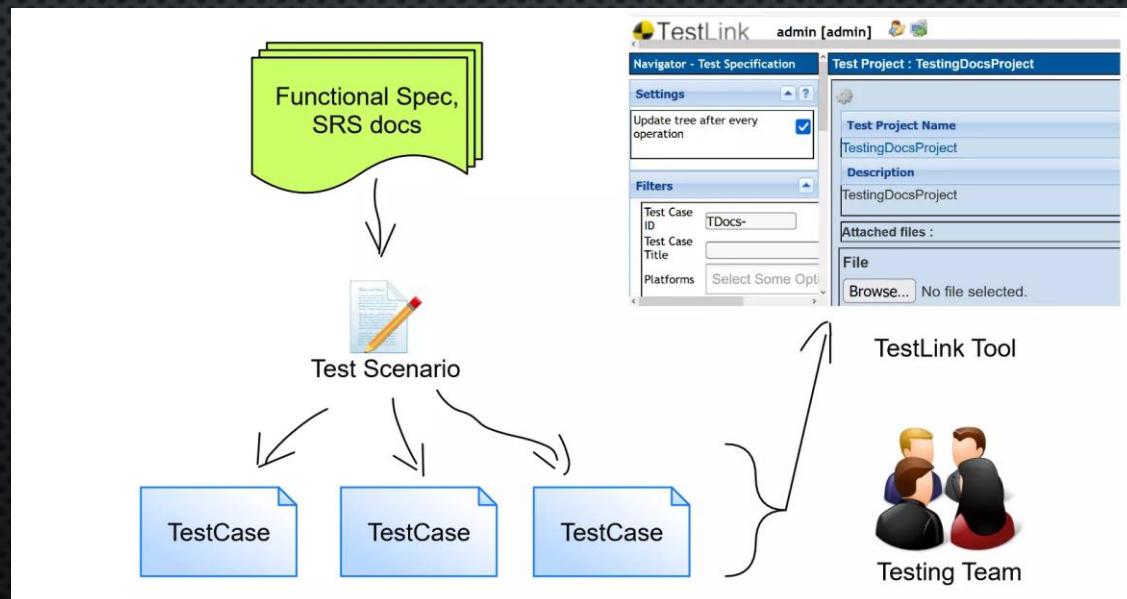
# CONCEPTOS A TENER EN CUENTA

- **DEBUGGIN:** ES EL PROCESO DE DIAGNOSTICAR LA CAUSA DE UN BUG Y CORREGIRLO. ES UNA ACTIVIDAD DE CORRECCIÓN Y NO DE PRUEBA



# QUE ES UN CASO DE PRUEBA

En ingeniería del software, un caso de prueba (en inglés, test case) es un **conjunto de condiciones o variables** bajo las cuales se determinará si una aplicación, un sistema de software o una característica o comportamiento de estos resulta o no aceptable.



# TEST CASE

Formato de caso de Prueba	
<b>Objetivo del caso de prueba</b>	Validar que en la creación de un nuevo perfil de una cuenta activa de netflix, la opción para el perfil infantil funcione correctamente y muestre la franja de películas infantiles.
<b>Identificador</b>	TC_profiles_0001
<b>Nombre Del Caso</b>	Perfil Infantil Creación perfil
<b>Precondiciones</b>	<ul style="list-style-type: none"> <li>• Una cuenta de usuario activa</li> <li>• Ejecutar el TC_login_0001 autenticación de usuario.</li> </ul>
<b>Paso</b>	<b>Resultado Esperado</b>
1) Ir a la opción “Administrar Perfiles” y hacer clic	Se debe mostrar la interfaz de administración de perfiles
2) Elegir la opción “Añadir Perfil”	
3) Ingresar un nombre para el perfil	
4) Activar el cuadro “Infantil”	Se debe mostrar una alerta de la edad asociada a la opción.
5) Clic en el botón continuar	
6) Clic en el botón listo	
7) Elegir el nuevo perfil que se acabó de crear en el paso 3	Se debería mostrar la franja de películas infantiles disponibles.

# ENFOQUES DE PRUEBAS

- **PRUEBAS ESTÁTICAS:** SON EL TIPO DE PRUEBAS QUE **SE REALIZAN SIN EJECUTAR EL CÓDIGO DE LA APLICACIÓN.** PUEDE REFERIRSE A LA REVISIÓN DE DOCUMENTOS, YA QUE NO SE HACE UNA EJECUCIÓN DE CÓDIGO. ESTO SE DEBE A QUE SE PUEDEN REALIZAR "PRUEBAS DE ESCRITORIO" CON EL OBJETIVO DE SEGUIR LOS FLUJOS DE LA APLICACIÓN.



- **PRUEBAS DINÁMICAS:** APUNTA A EJECUTAR LAS PRUEBAS EN TODO O PARTE DEL SOFTWARE **PARA DETERMINAR SI FUNCIONA SEGÚN LO ESPERADO (CÓDIGO EN EJECUCIÓN).** LAS PRUEBAS DINÁMICAS **PERMITEN EL USO DE TÉCNICAS DE CAJA NEGRA Y CAJA BLANCA CON MAYOR AMPLITUD.** DEBIDO A LA NATURALEZA DINÁMICA DE LA EJECUCIÓN DE PRUEBAS **ES POSIBLE MEDIR CON MAYOR PRECISIÓN EL COMPORTAMIENTO DE LA APLICACIÓN DESARROLLADA.**



# Enfoques de Pruebas

- CUAL ES LA PRINCIPAL DIFERENCIA ENTRE LAS PRUEBAS ESTÁTICAS Y PRUEBAS DINÁMICAS???



# Enfoques de Pruebas



## Pruebas de ciclo de desarrollo

### Pruebas dinámicas

Planificación      Diseño      Ejecución

Pruebas funcionales

Pruebas de integración de sistemas

Pruebas de regresión

### Pruebas estáticas

Control de productos  
Quality Assurance QA

Revisiones documentales

Validación entradas-salidas

Análisis de código estático

Control de procesos  
Process Assurance PA

# Pruebas dinámicas

## En ejecución

Las pruebas se realizan mientras el código está en ejecución.

## Detección

Estas pruebas se enfocan en la detección de defectos.

## Pruebas tardías

Estas pruebas se realizan cuando el código es desplegado a un ambiente de pruebas.

## Técnicas

Se utilizan tipos de pruebas:

- Funcionales
- No funcionales



## Pruebas dinámicas

## Pruebas estáticas

# Pruebas estáticas

## No ejecución

Las pruebas se realizan a productos de trabajo como documentos de requerimientos, casos de prueba, planes de prueba y código.

## Prevención

Estas pruebas se enfocan en la prevención de defectos.

## Pruebas tempranas

Estas pruebas pueden ser realizadas desde las primeras etapas del ciclo de vida del software.

## Técnicas

Se utilizan técnicas como:

- Revisión técnica
- Inspección
- Revisión de código

# Clasificación de las pruebas

Según el conocimiento del código

Caja Negra

Caja Blanca

Según la etapa de desarrollo

Unitario

Integración

Sistema

Aceptación

Regresión

Según el aspecto a evaluar

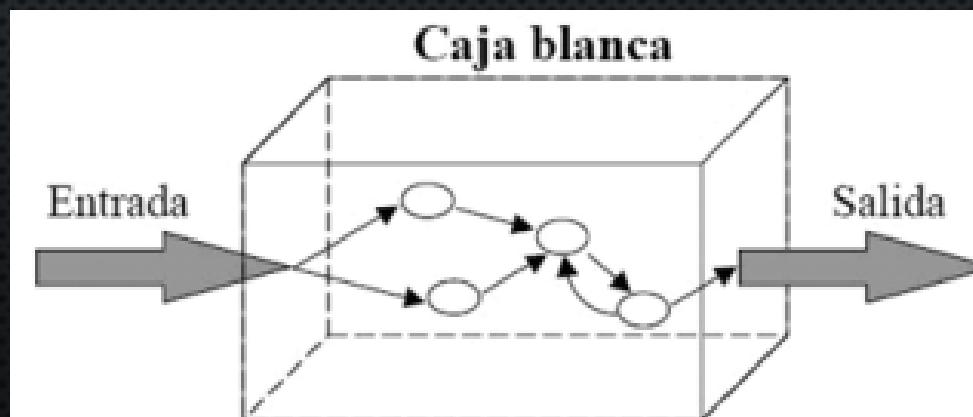
Funcional

Para-funcional

# PRUEBAS DE CAJAS

- **PRUEBAS DE CAJA BLANCA:**

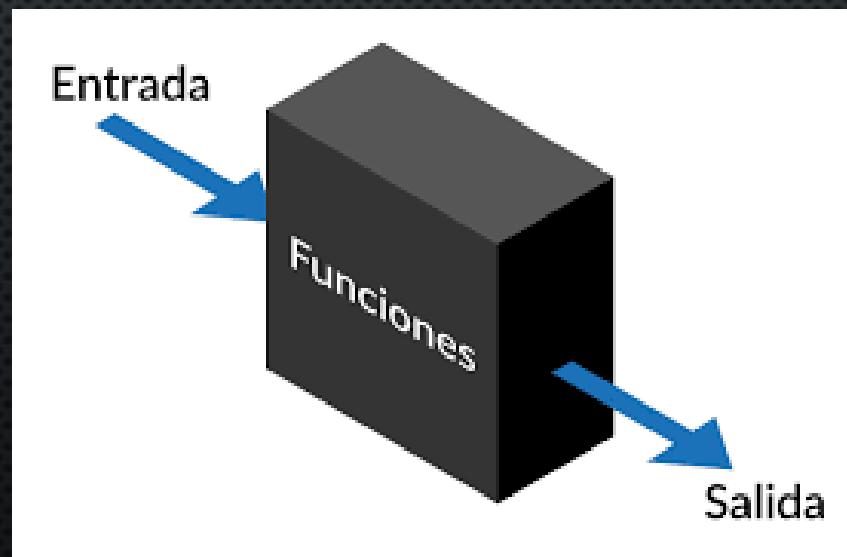
- EN LAS PRUEBAS DE CAJA BLANCA SE PIENSAN Y DISEÑAN CASOS DE PRUEBA **CONOCIENDO EL CÓDIGO DEL SOFTWARE.**
- ES REALIZADO POR EL DESARROLLADOR QUE ESCRIBIÓ EL CÓDIGO O POR OTRO DESARROLLADOR, YA QUE **LAS PRUEBAS ESTÁN FUERTEMENTE LIGADAS AL CÓDIGO.**
- ES IMPORTANTE DESTACAR QUE LAS PRUEBAS DE CAJA BLANCA **ESTÁN ENFOCADAS A IDENTIFICAR DEBILIDADES EN EL DISEÑO DEL CÓDIGO** Y NO A DETECTAR DISCREPANCIAS ENTRE LOS REQUERIMIENTOS Y SU IMPLEMENTACIÓN.
- EN LAS PRUEBAS DE CAJA BLANCA LOS PROGRAMAS SON VISTOS EN TÉRMINOS DE UNIDADES DE CÓDIGO QUE INTERACTÚAN ENTRE SÍ.



# PRUEBAS DE CAJAS

- **PRUEBAS DE CAJA NEGRA:**

- SE TRATA AL SISTEMA COMO UNA CAJA NEGRA, DE LA QUE NO SE CONOCE NI SU CONTENIDO NI SU ESTRUCTURA INTERNA.
- PARA LA EJECUCIÓN DE PRUEBAS DE CAJA NEGRA, **SE SUMINISTRAN DATOS DE ENTRADA AL SISTEMA, SE OBSERVA LA SALIDA OBTENIDA Y SE COMPARA CON LA SALIDA ESPERADA.**
- SI BIEN **NO CONOCEMOS** LA ESTRUCTURA INTERNA DEL SISTEMA, SABEMOS CÓMO SE ESPERA QUE SE COMPORTE.



# PRUEBAS UNITARIAS

- ESTE TIPO DE PRUEBAS, SE PRUEBAN DE FORMA AISLADA LAS UNIDADES O CONJUNTO DE UNIDADES RELACIONADAS DE UN SOFTWARE.
- PODEMOS CONSIDERAR **UNA UNIDAD COMO UNA CLASE, UN MÉTODO, UN COMPONENTE O UN SUBSISTEMA.**
- LAS PRUEBAS UNITARIAS INTENTAN DETECTAR DISCREPANCIAS ENTRE LOS REQUERIMIENTOS Y EL COMPORTAMIENTO REAL DE LA UNIDAD.
- SE PONE ÉNFASIS EN VERIFICAR LA ESPECIFICACIÓN DE LA INTERFAZ DE LAS UNIDADES.
- ESTAS PRUEBAS PUEDEN SER **MANUALES O AUTOMATIZADAS**, PUEDEN SER EJECUTADAS POR QUIEN DESARROLLÓ EL SOFTWARE O POR UN TERCERO.

# PRUEBAS UNITARIAS

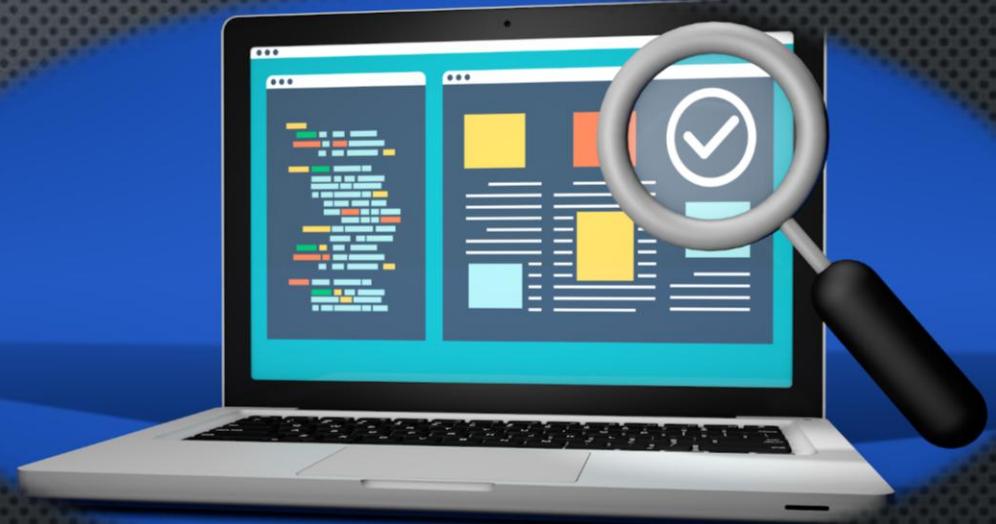
- PASOS A SEGUIR:
  - SEGÚN EL LENGUAJE DE PROGRAMACIÓN QUE PROBEMOS VAMOS A TENER DIFERENTES HERRAMIENTAS PARA REALIZAR LAS PRUEBAS UNITARIAS.
  - EN EL CASO DE PYTHON CONTAMOS CON LIBRERÍA UNITTEST
  - SE CREA UN ARCHIVO CON UNA CLASE DONDE SE CARGAN LOS TEST.
    - LOS TEST DEBEN CONTENER EL RESULTADO ESPERADO DE LA PRUEBA.
    - LOS VALORES QUE SE DEBEN PROBAR
    - LA LÓGICA QUE DEVUELVA EL RESULTADO OBTENIDO.

```
11  
12 class TestOperaciones(unittest.TestCase):  
13     def setUp(self):  
14         # Aquí, opcionalmente, ejecuta lo que deberías ejecutar antes  
15         # de comenzar cada test.  
16         pass  
17  
18     def test_suma(self):  
19         esperado = 3  
20         actual = suma(1, 2)  
21         # Pásalo en el orden: actual, esperado  
22         self.assertEqual(actual, esperado)  
23  
24     def test_resta(self):  
25         esperado = 5  
26         actual = resta(10, 5)  
27         # Pásalo en el orden: actual, esperado  
28         self.assertEqual(actual, esperado)  
29  
30     def test_multiplicacion(self):  
31         esperado = 50  
32         actual = multiplicacion(10, 5)
```

# MATERIAL DE CONSULTA

- [HTTPS://PROGRAMACIONYMAS.COM/BLOG/TIPOS-DE-TESTING-EN-DESARROLLO-DE-SOFTWARE](https://programacionymas.com/blog/tipos-de-testing-en-desarrollo-de-software)
- [HTTPS://WWW.ATLASSIAN.COM/ES/CONTINUOUS-DELIVERY/SOFTWARE-TESTING/TYPES-OF-SOFTWARE-TESTING](https://www.atlassian.com/es/continuous-delivery/software-testing/types-of-software-testing)
- [HTTPS://ES.WIKIPEDIA.ORG/WIKI/PRUEBAS\\_DE\\_SOFTWARE](https://es.wikipedia.org/wiki/Pruebas_de_software)
- [HTTPS://SIЛО.TIPS/DOWNLOAD/CRITERIOS-DE-CLASIFICACION](https://siло.tips/download/criterios-de-clasificacion)
- [HTTPS://PARZIBYTE.ME/BLOG/2019/06/11/UNIT-TESTING-PYTHON-REALIZAR-PRUEBAS-CODIGO/#DOCUMENTACION\\_DE\\_UNITTEST\\_EN\\_PYTHON](https://parzibyte.me/blog/2019/06/11/unit-testing-python-realizar-pruebas-codigo/#documentacion_de_unittest_en_python)

# METODOLOGÍA DE PRUEBAS DE SOFTWARE



**Materia:** Metodología de Pruebas de Software  
**Profesor:** Ing. Pablo Andrés Pérez