Tema 13: Pruebas Software

Introducción a la Ingeniería del Software y los Sistemas de Información I

Ingeniería Informática – Tecnologías Informáticas

Departamento de Lenguajes y Sistemas Informáticos

Índice

Introducción

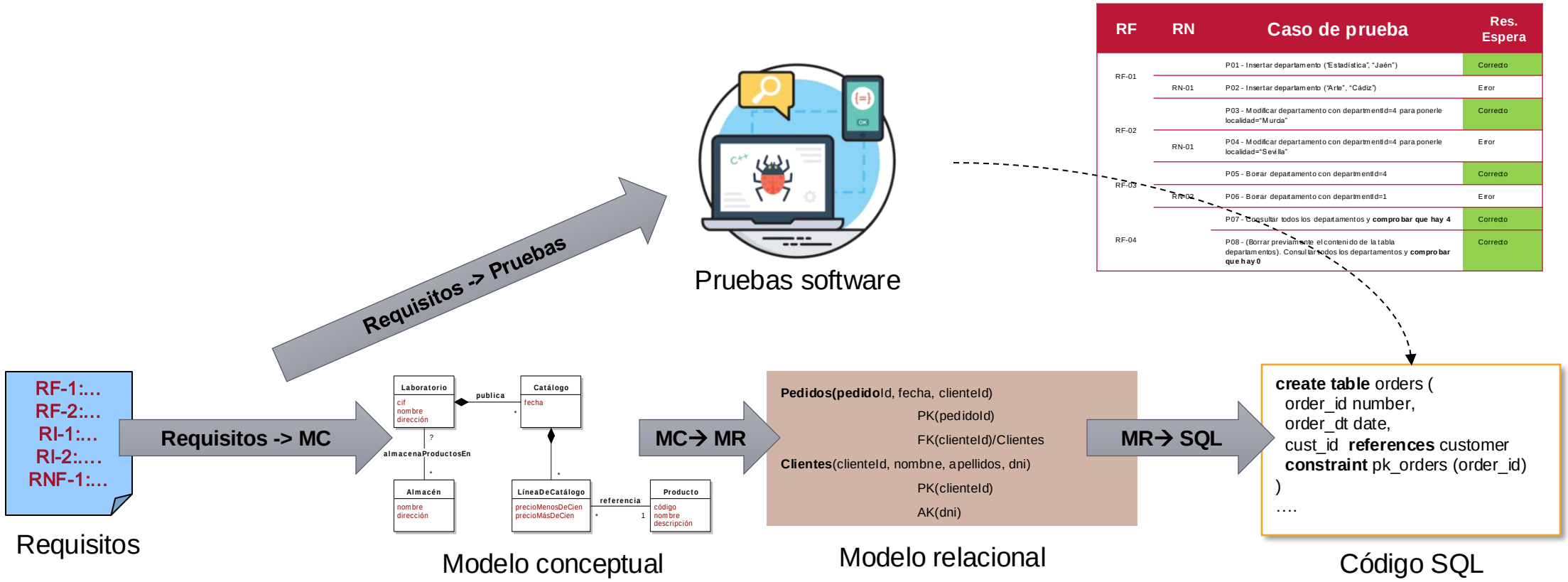
Estándar ANSI/IEEE 1059:

Las pruebas en sistemas software (software testing) son un proceso para:

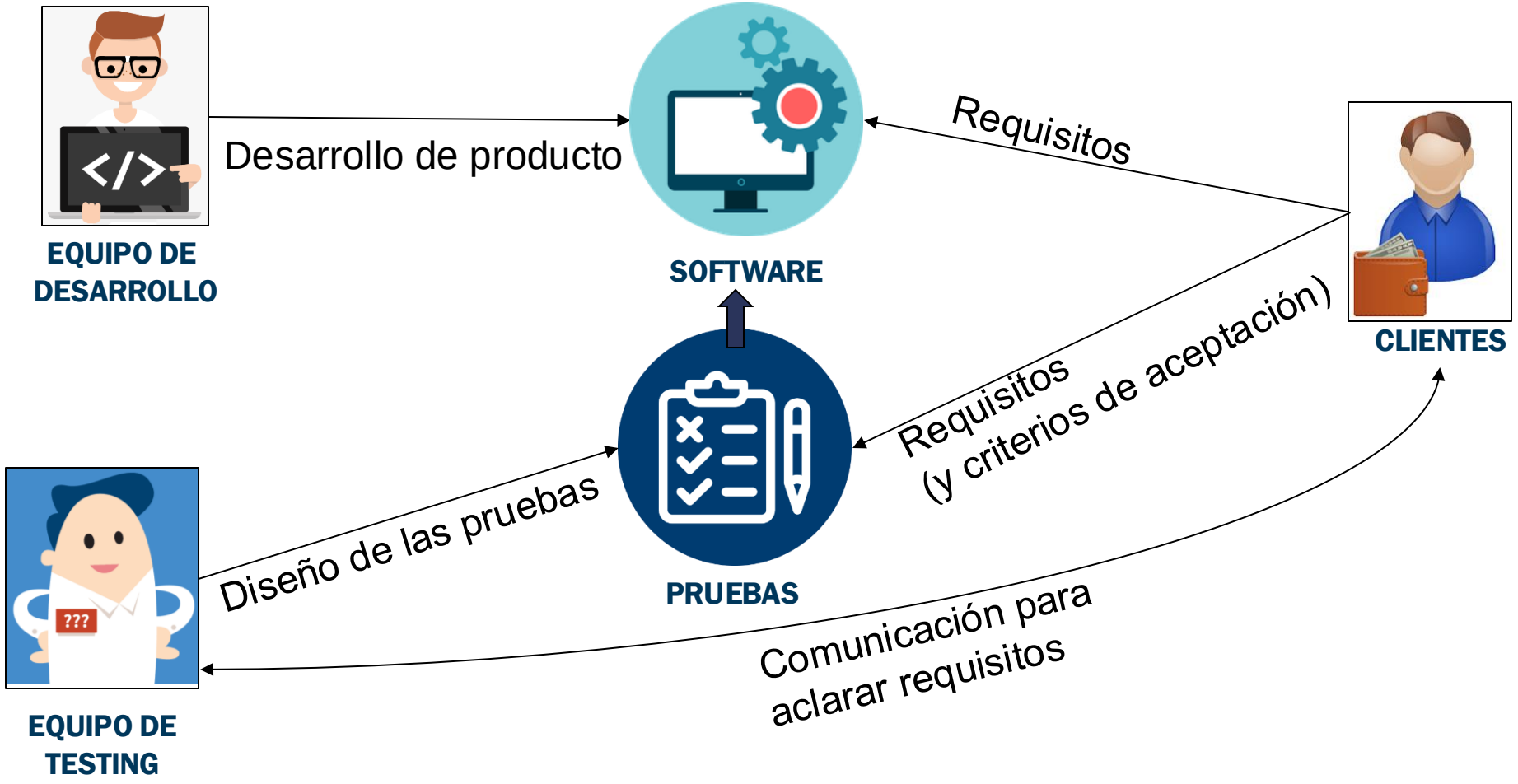
- Evaluar la funcionalidad de una aplicación software.
- Averiguar si el software desarrollado satisface o no los requisitos.
- Encontrar posibles defectos.
- Maximizar la calidad del producto.

Introducción

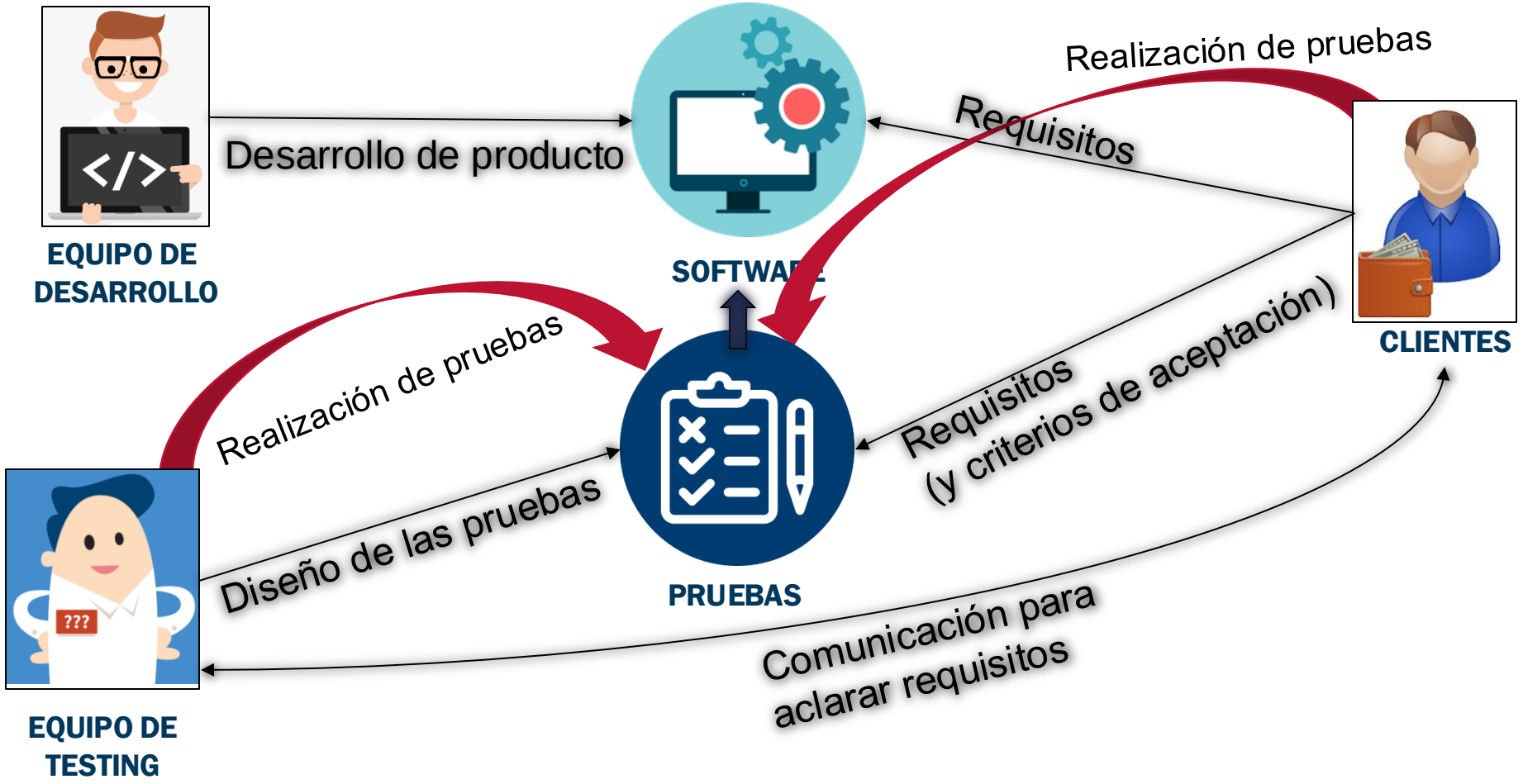
Trazabilidad



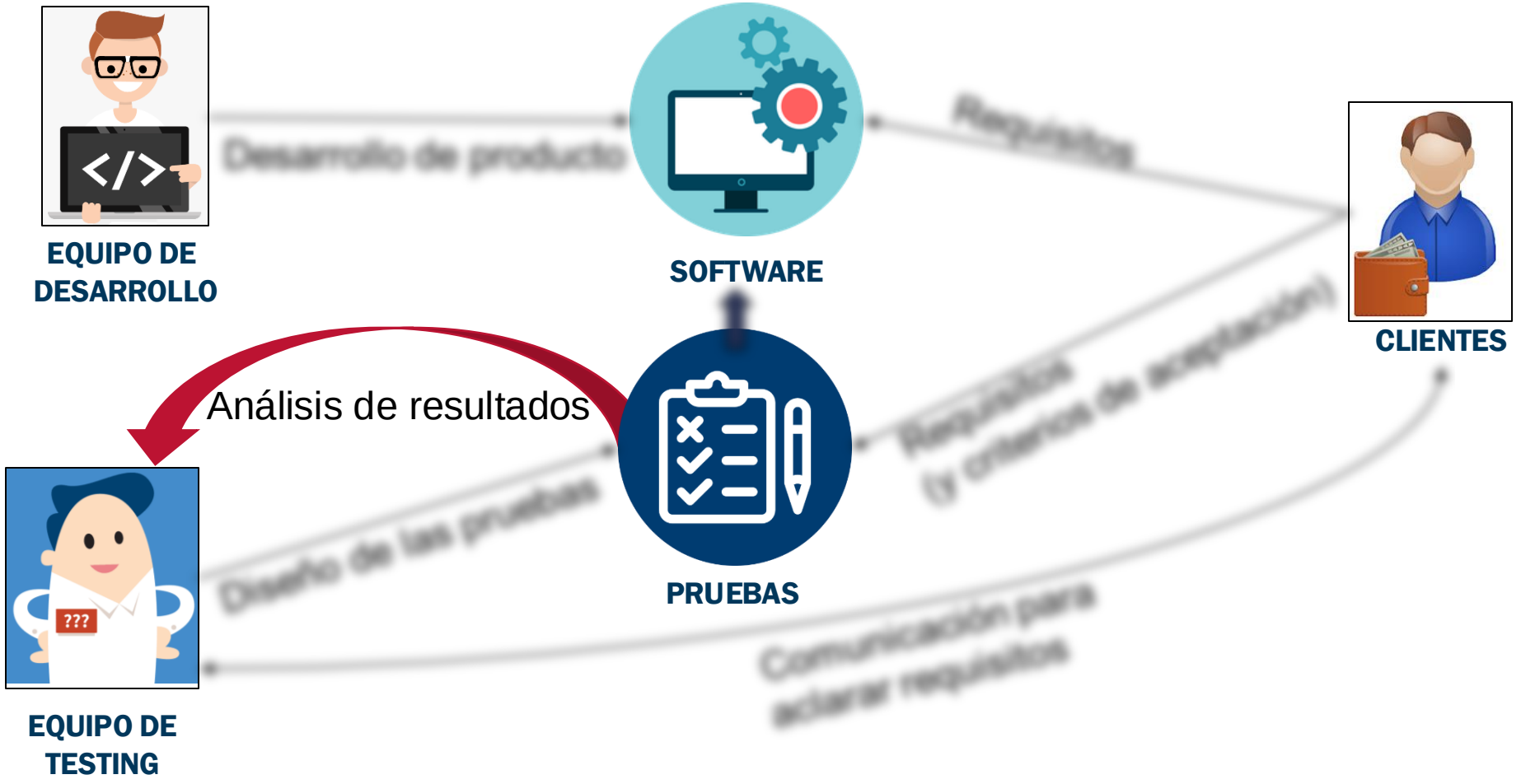
Ingeniería de pruebas



Ingeniería de pruebas

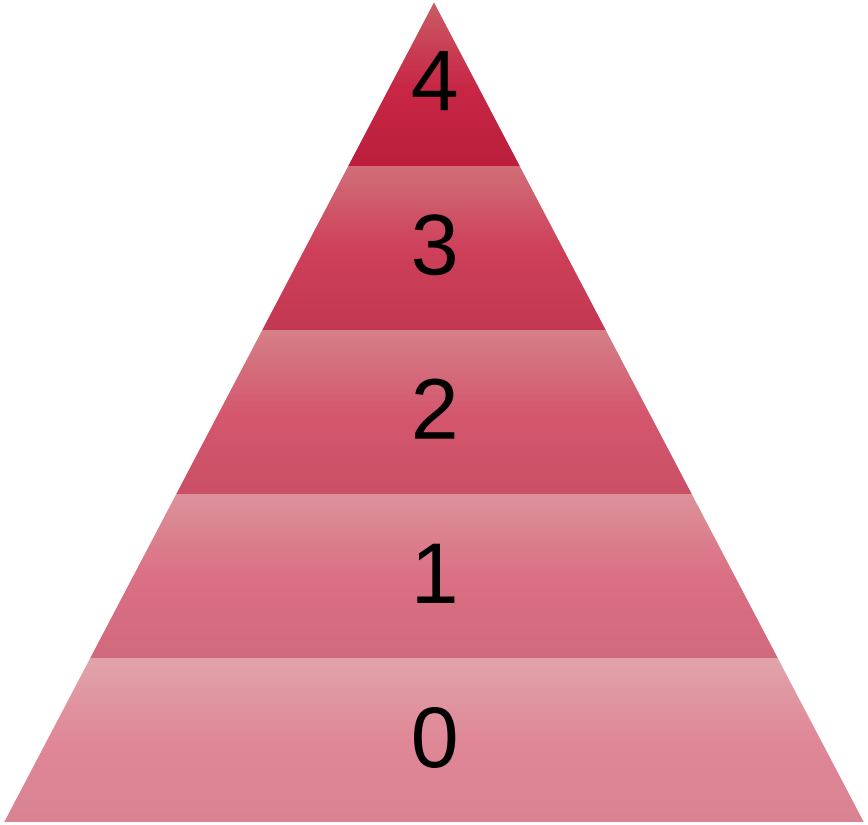


Ingeniería de pruebas



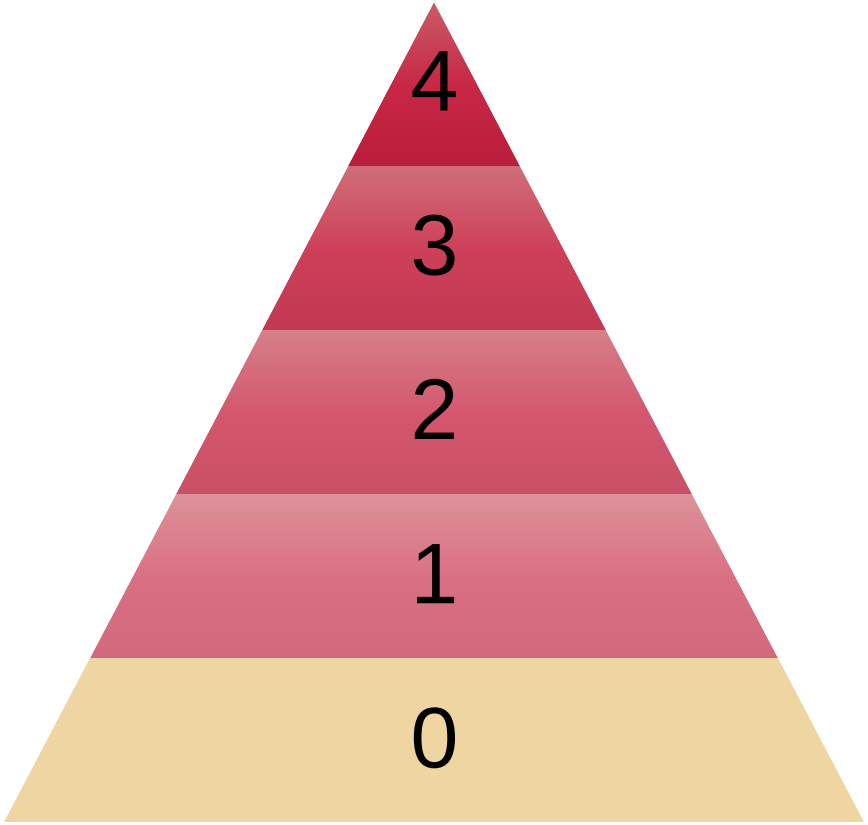
Ingeniería de pruebas

Niveles de testing



Software Testing Techniques (Beizer, 2003)

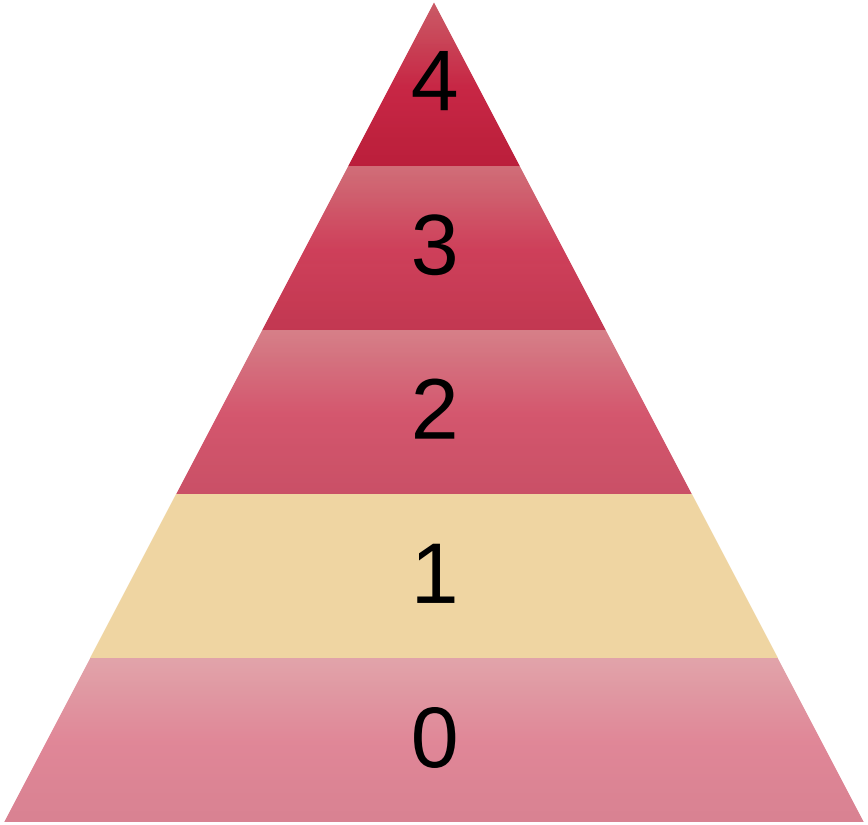
Niveles de testing



Testing como sinónimo de **depuración**

- Sólo se prueban unos pocos casos
- Habitualmente sólo se prueba el *happy path*
- Se prueba, pero no de forma metódica
- No enseña a desarrollar mejores sistemas
- Ejemplo: estudiantes de FP (el objetivo del testing es que el programa no de fallos de compilación o interpretación)

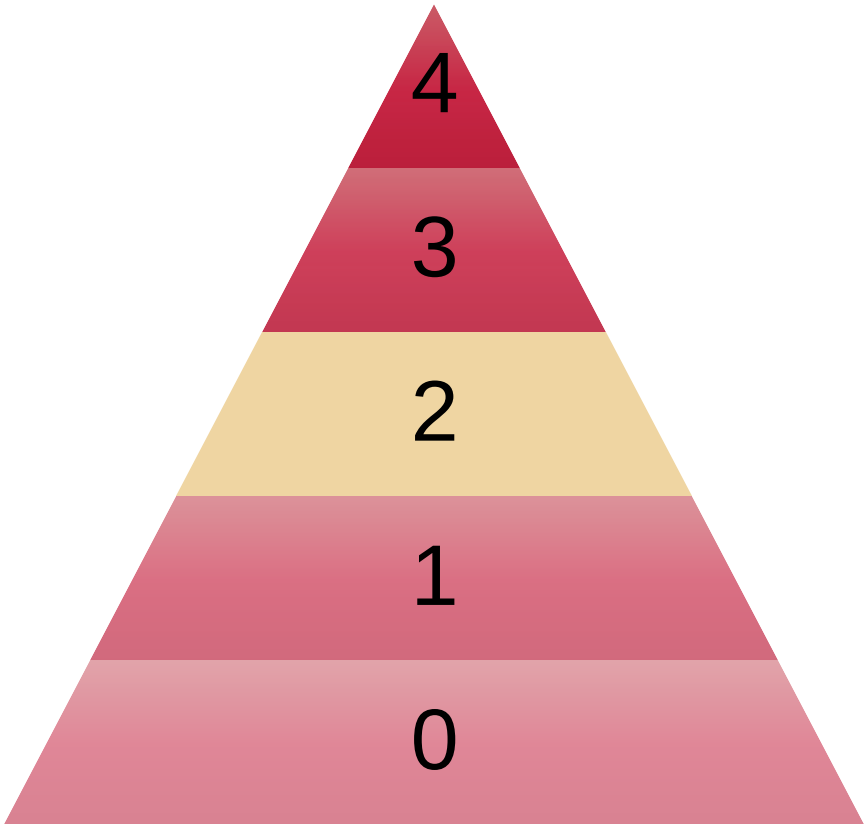
Niveles de testing



Testing como sinónimo de **validación**:

- El objetivo es probar que el sistema funciona
- Foco en el “happy path”
- Se prueba, pero no se detectan la mayoría de los fallos
- Problema: la corrección absoluta es imposible de conseguir/demostrar
- ¿Cómo saber cuándo parar de probar?

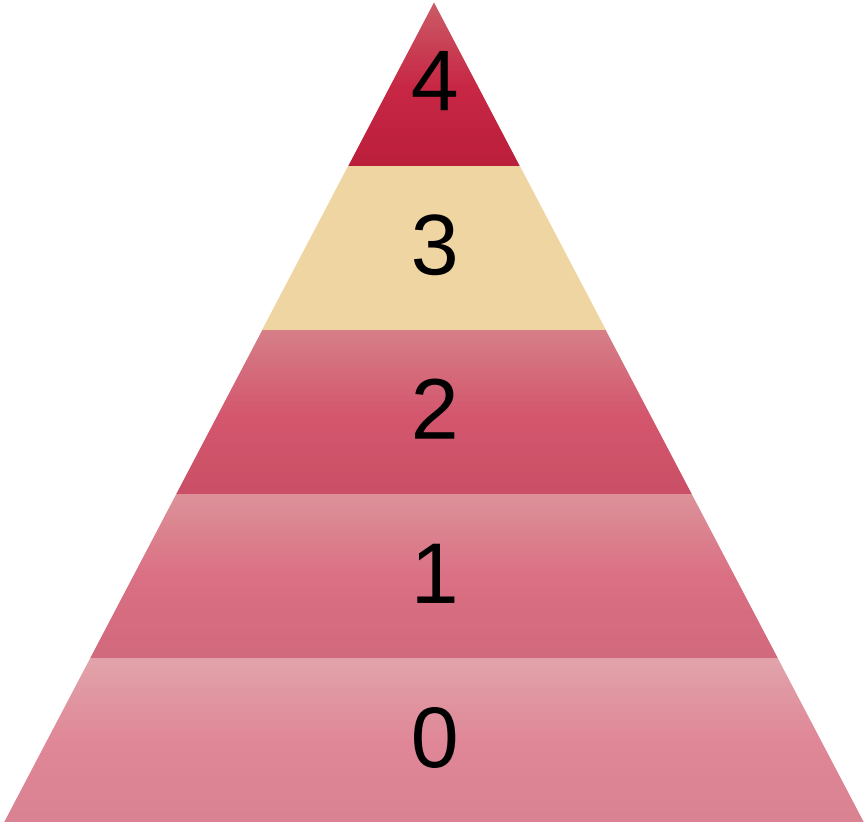
Niveles de testing



Testing **agresivo**:

- El objetivo es probar que el sistema **NO** funciona
- Muchos más casos de prueba que en los niveles anteriores
- Conflictos entre desarrolladores (a la defensiva) y testers (al ataque)
- ¿Cómo saber cuándo parar de probar?

Niveles de testing

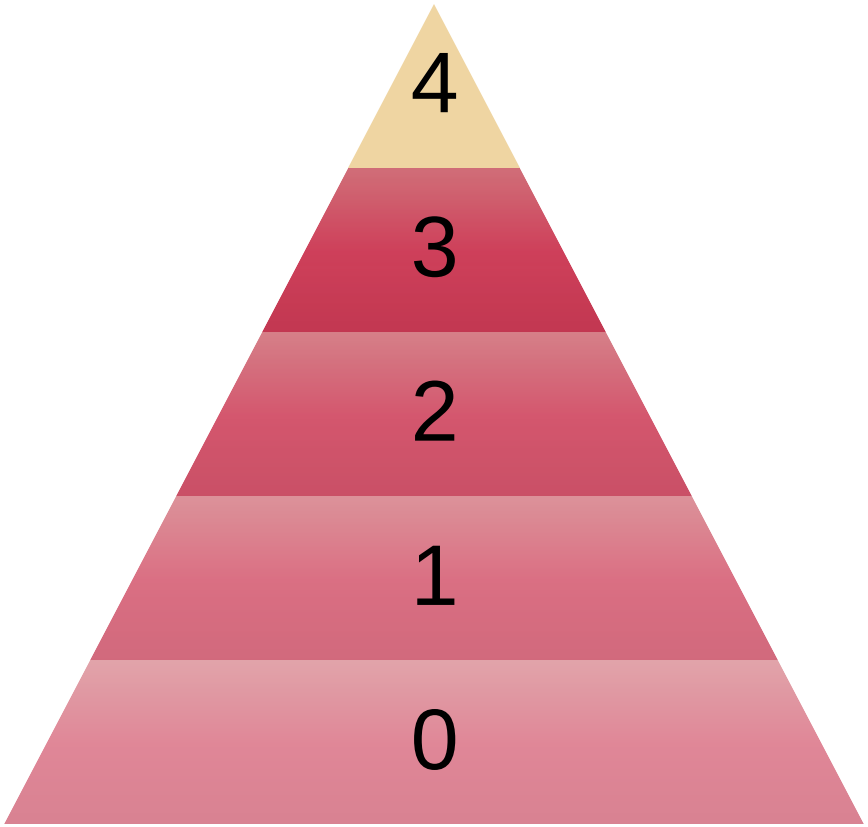


Testing **preventivo**:

- Se asume que hay riesgo de fallos al utilizar todo software
- El objetivo es encontrar cuantos más fallos mejor para minimizar este riesgo
- Trabajo cooperativo entre desarrolladores y testers

“Testing can show presence, but not the absence of bugs” (Dijkstra).

Niveles de testing



Testing como **filosofía de trabajo**

- Disciplina mental que ayuda a los profesionales de TI a desarrollar software de mayor calidad.
- Los testers definen medidas de calidad
- Analogía del corrector ortográfico (Beizer): El objetivo no es sólo detectar palabras incorrectas, sino enseñar a corregirlas

Clasificación

- Pruebas unitarias
  - Funcionales
  - Estructurales
- Pruebas de integración
- Pruebas de rendimiento
- Pruebas de aceptación
- Pruebas de usabilidad
- A/B testing
- Pruebas de regresión

Fallo vs Error

Error software: Defecto del código.

- No tiene por qué manifestarse (si nunca se llega a cumplir una condición, por ejemplo)
- Se conocen como “bugs”

Fallo software: Manifestación de un defecto en el código en tiempo de ejecución.

- Es la manifestación de un error.

Ejemplo

Error: empezar el bucle en la posición 1 en vez de en la 0

```
public static int sum (int[] x) {
        int sum = 0;
        for (int i = 1; i < x.length; i++)
                sum += x[i];
        return sum;
}
```

sum([2,5,4]) = 9

Fallo: resultado incorrecto

sum([0,5,4]) = 9

No siempre se produce un fallo como consecuencia de un error

Ejemplo

- Función que calcula el número de empleados en departamentos de una localidad pasada como parámetro

```
CREATE OR REPLACE FUNCTION
  fNumEmpleados(loc VARCHAR(64)) RETURNS INT
BEGIN
  RETURN (
    SELECT COUNT(*)
    FROM empleados E JOIN departamentos D
    ON (E.departamentoId = D.departamentoId)
);
END
```

Si en la BD sólo hay empleados en departamentos de la localidad pasada como parámetro, no se produce fallo (el resultado es el esperado)

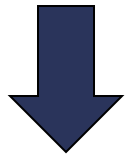
Si hay empleados en departamentos de distintas localidades, se produce fallo (el resultado no es el esperado) -> se detecta el error

Error: falta la cláusula `WHERE D.localidad = loc`

Proceso de testing

- **Resultados esperados**: resultados que debe dar la ejecución de una prueba si todo es correcto; se definen a partir de los requisitos
- **Resultados obtenidos**: resultados reales que arroja la ejecución de una prueba

Resultados esperados != resultados obtenidos



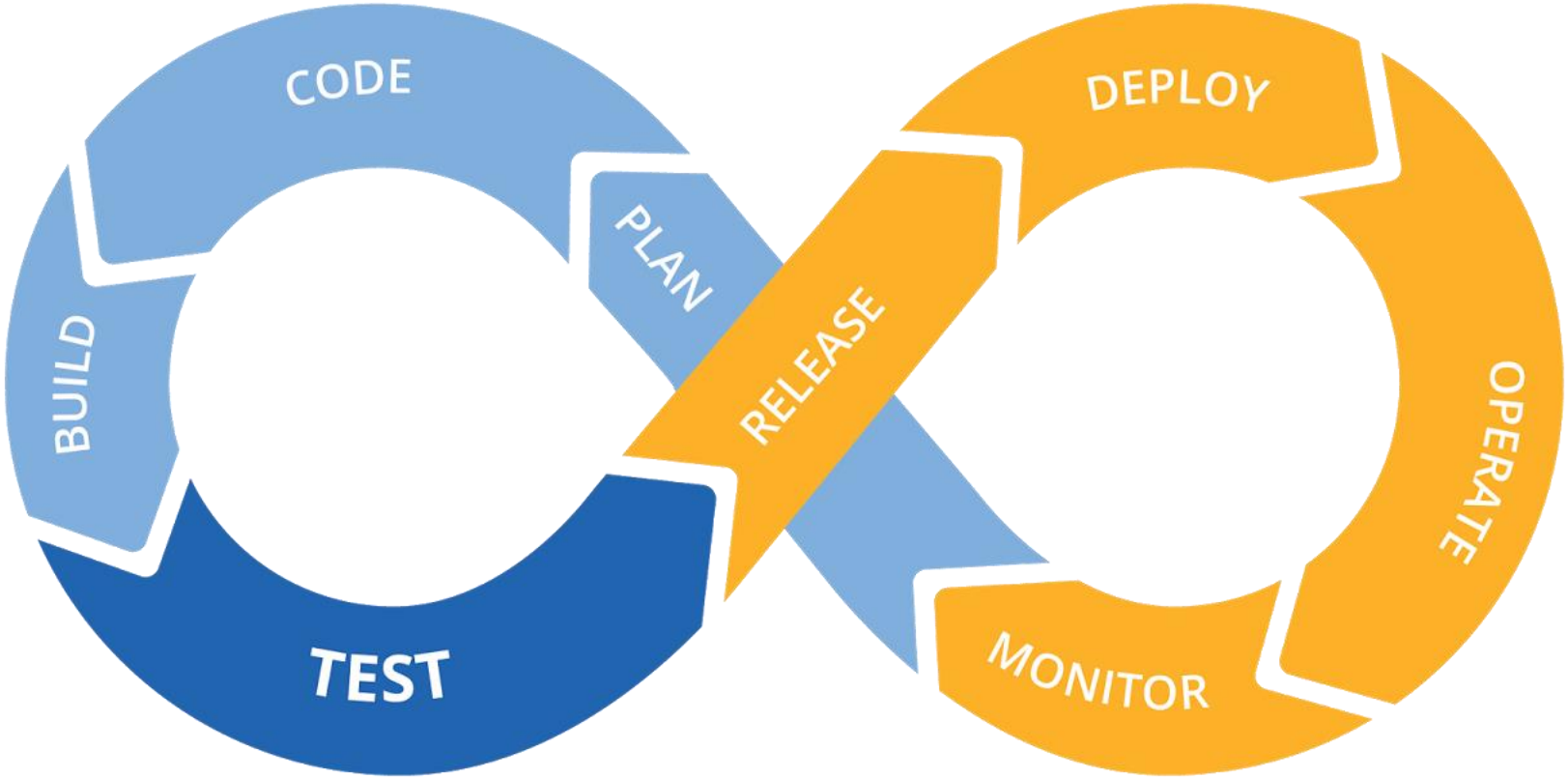
Fallo detectado

- **Depuración**: Proceso que, a partir de un fallo, analiza el código para encontrar el error que lo ha causado.

Automatización de las pruebas

- No siempre merece la pena el coste invertido
- Requiere cierto trabajo inicial adicional
- Algunas herramientas del mercado:

Integración continua



CI/CD

- CI/CD:
  - Continuous Integration (Grady Booch): Hacer *merges* frecuentes para evitar hacer un gran *merge* al final del *sprint* (reduce la probabilidad de conflictos).
  - Continuous Delivery: Tener frecuentes versiones de software “estable”.
- Idea:
  - Automatizar el proceso: el servidor automáticamente compila, pasa las pruebas y, de acuerdo con el resultado, sube el elemento compilado o bien envía informe de errores a los desarrolladores.
- Herramientas: GitHub Actions, Jenkins, CircleCI, Travis, TeamCity, Bamboo, Codeship, Coveralls…

Casos de pruebas

Caso de prueba (Test Case): Combinación de valores de entrada y resultados esperados que se usan para probar un escenario particular de un determinado sistema (o parte de un sistema)

Conjunto de pruebas (Test Set): Conjunto de casos de prueba diseñados para probar distintos escenarios posibles para un determinado sistema (o una parte)

Casos de prueba positivos:

- Simular escenarios de prueba “normales” (funcionamiento correcto)
- Datos de entrada correctos con respecto a las restricciones y reglas de negocio definidos

Casos de prueba negativos:

- Simular escenarios de prueba “anormales” (fallo esperado y controlado de la aplicación)
- Datos de entrada para intentar romper las restricciones y reglas de negocio

Cobertura

- **Cobertura**: Determina qué porcentaje del código se está probando con un conjunto de casos de prueba.
- Un testing “exhaustivo” (cobertura del 100%) supondría probar con todos los casos de prueba posibles.
- El número de casos de prueba posibles es, habitualmente, infinito.
  - Ejemplo: ¿Cuántos posibles casos podemos pasarle a la función que suma los elementos de un array de enteros?
    - []
    - [0]
    - [0,1]
    - [2,13,37,69,420]
    - [2,3,4,5,6,7,8,9,0,0,0,8,6,4,3,6,78,9,0,56,4,3,4,5,6,78,8,…]
    - …

Estrategia viable

Generar un conjunto de pruebas que garantice una **cobertura** como **mínimo**. Posibilidades:

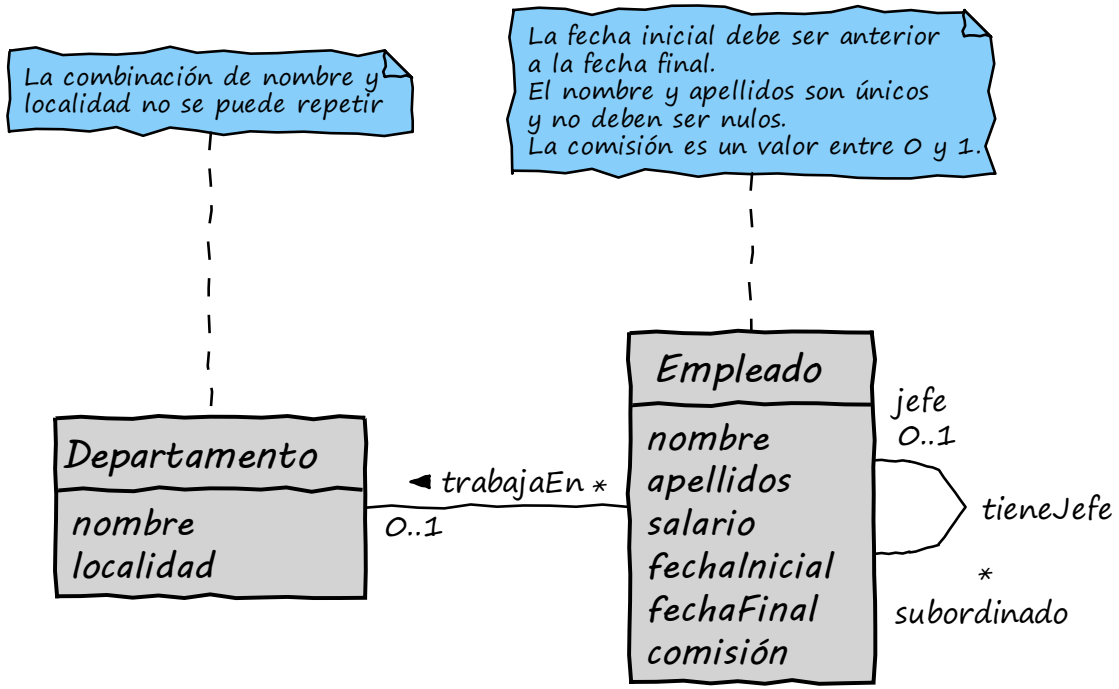
- Probar con los valores frontera de cada tipo de datos (valores extremos de cada dominio)
  - Da lugar también a un número alto de casos de prueba (todas las posibles combinaciones de valores fronteras en todos los atributos de cada entidad)
- Generación aleatoria de casos de prueba:
  - No garantiza cobertura mínima
- Generación automatizada inteligente:
  - Muy compleja
- **Aproximación guiada por reglas de negocio**
  - Probar un escenario en el que se **respetan** todas las reglas de negocio
  - Probar escenarios en los que se **violen** las reglas de negocio **una por una**

Guía para diseñar los casos de prueba

Por cada requisito de información (entidad)

- Por cada requisito funcional de modificación/inserción/borrado de la entidad:
  - 1 caso de prueba positivo
    - Dar valores a los atributos de la entidad de forma que se cumplan todas las reglas de negocio/restricciones
  - Por cada regla de negocio/restricción aplicable a la entidad, X casos de prueba negativo
    - En cada caso de prueba negativo, dar valores a los atributos de la entidad de forma que se viole **únicamente esa** regla de negocio / restricción (de todas las formas posibles)
- Por cada requisito funcional de consulta de la entidad:
  - 2 casos de prueba positivos:
    - Habiendo en la BD entidades para el listado
    - No habiendo en la BD entidades para el listado

Ejemplo: Empleados y departamentos



Requisitos de información:
RI-01: Departamentos (nombre, localidad)
RI-02: Empleados (nombre, apellidos, salario, fecha inicial, fecha final, comisión, jefe y departamento en el que trabajan)

Ejemplo: Empleados y departamentos

Requisitos funcionales:

- RF-01: Insertar un departamento nuevo
- RF-02: Modificar un departamento existente
- RF-03: Borrar un departamento
- RF-04: Consultar los departamentos existentes
- RF-05: Insertar un empleado nuevo
- RF-06: Modificar un empleado existente
- RF-07: Borrar un empleado
- RF-08: Consultar los empleados existentes

Ejemplo: Empleados y departamentos

Reglas de negocio asociadas a los **departamentos**:

- RN-01: No puede haber dos departamentos con el mismo nombre en la misma localidad
- RN-02: No se puede borrar un departamento si tiene empleados

Reglas de negocio asociadas a los **empleados**:

- RN-03: La fecha inicial no puede ser posterior a la fecha final de contrato
- RN-04: El nombre y apellidos de empleado son únicos y no pueden ser nulos
- RN-05: La comisión toma un valor entre 0 y 1

Restricciones (implícitas)

- Res-1: Un empleado no puede pertenecer a un departamento que no existe
- Res-2: Un empleado no puede tenerse a sí mismo como jefe
- Res-3: Un empleado no puede tener como jefe a otro empleado que no exista
- Res-4: Un empleado no puede tener salario negativo
- Res-5: Un empleado no puede tener email duplicado (dado que esta es la forma de hacer login en el sistema)

Estado inicial de la BD

departmentId	nameDep	city
3	Arte	Cádiz
1	Historia	(NULL)
4	Informática	(NULL)
2	Informática	Sevilla

employeeId	departmentId	bossId	firstName	lastName	salary	startDate	endDate	fee
1	1	(NULL)	Pedro	Ruiz	5.000,00	2017-09-15	(NULL)	0,2
2	1	(NULL)	José	Borrego	3.500,00	2018-08-10	(NULL)	0,5
3	2	(NULL)	Lola	Díaz	2.500,00	2018-08-10	(NULL)	0,3
4	1	1	Luis	Hernández	2.500,00	2018-08-10	2018-11-15	0
5	1	1	Ana	Ayala	2.500,00	2018-08-10	2018-11-15	0

Conjunto de pruebas - Departamentos

RF	RN	Caso de prueba	Resultado Esperado
RF-01		P01 - Insertar departamento (“Estadística”, “Jaén”)	Correcto
	RN-01	P02 - Insertar departamento (“Arte”, “Cádiz”)	Error
RF-02		P03 - Modificar departamento con departmentId=4 para ponerle localidad=“Murcia”	Correcto
	RN-01	P04 - Modificar departamento con departmentId=4 para ponerle localidad=“Sevilla”	Error
RF-03		P05 - Borrar departamento con departmentId=4	Correcto
	RN-02	P06 - Borrar departamento con departmentId=1	Error
RF-04		P07 - Consultar todos los departamentos y **comprobar que hay 4**	Correcto
		P08 - (Borrar previamente el contenido de la tabla departamentos). Consultar todos los departamentos y **comprobar que hay 0**	Correcto

Conjunto de pruebas - Empleados

RF	RN	Caso de prueba	Resultado Esperado
RF-05		P01 - Insertar empleado (1, 1, “Juan”, “Pérez”, 5000, 12/11/2019, 12/12/2019, 0.1)	Correcto
	RN-03	P02 - Insertar empleado (1, NULL, “Juan”, “Pérez” 5000, 12/03/2019, 11/02/2019, 0.1)	Error
	RN-04	P03 - Insertar empleado (1, NULL, “Lola”, “Díaz”, 5000, 12/11/2019, 12/12/2019, 0.1)	Error
		P04 - Insertar empleado (1, NULL, NULL, NULL, 5000, 12/11/2019, 12/12/2019, 0.1)	Error
		P05 - Insertar empleado (1, NULL, “Lola”, NULL, 5000, 12/11/2019, 12/12/2019, 3.5)	Error
	Res-1	P06 - Insertar empleado (10, NULL, “Juan”, “Pérez”, 5000, 12/11/2019, 12/12/2019, 0.1)	Error
	Res-2	P07 - Insertar empleado (1, 6, “Juan”, “Pérez”, 5000, 12/11/2019, 12/12/2019, 0.1)	Error
	Res-3	P08 - - Insertar empleado (1, 23, “Juan”, “Pérez”, 5000, 12/11/2019, 12/12/2019, 0.1)	Error
	…	…	…
		(…)	Correcto

Inconveniente 1

RF	RN	Caso de prueba	Resultado Esperado
RF-05		P01 - Insertar empleado (1, 1, “Juan”, “Pérez”, 5000, 12/11/2019, 12/12/2019, 0.1)	Correcto
	RN-03	P02 - Insertar empleado (1, NULL, “Juan”, “Pérez” 5000, 12/03/2019, 11/02/2019, 0.1)	Error
	RN-04	P03 - Insertar empleado (1, NULL, “Lola”, “Díaz”, 5000, 12/11/2019 12/12/2019, 0.1)	Error
		P04 - Insertar empleado (1, NULL, NU 12/12/2019, 0.1)	
		P05 - Insertar empleado (1, NULL, “L 12/12/2019, 3.5)	
	Res-1	P06 - Insertar empleado (10, NULL, “ 12/11/2019, 12/12/2019, 0.1)	
	Res-2	P07 - Insertar empleado (1 6, “Juan”, “Pérez”, 5000, 12/11/2019, 12/12/2019, 0.1)	Error
	Res-3	P08 - - Insertar empleado (1, 23, “Juan”, “Pérez”, 5000, 12/11/2019, 12/12/2019, 0.1)	Error
RF-06		(…)	Correcto

¿Y si alguien cambia la BD? ¿Se invalidan todas las pruebas?

Inconveniente 2

RF	RN	Caso de prueba	Resultado Esperado
RF-05		P01 - Insertar empleado (1, 1, "Juan", "Pérez", 5000, 12/11/2019, 12/12/2019, 0.1)	Correcto
	RN-03	P02 - Insertar empleado (1, NULL, "Juan", "Pérez" 5000, 12/03/2019, 11/02/2019, 0.1)	Error
	RN-04	P03 - Insertar empleado (1, NULL, "Lola", "Díaz", 5000, 12/11/2019 12/12/2019, 0.1)	Error
		P04 - Insertar empleado (1, NULL, NU 12/12/2019, 0.1)	
		P05 - Insertar empleado (1, NULL, "L 12/12/2019, 3.5)	
	Res-1	P06 - Insertar empleado (10, NULL, " 12/11/2019, 12/12/2019, 0.1)	
	Res-2	P07 - Insertar empleado (1, 6, "Juan", "Pérez", 5000, 12/11/2019, 12/12/2019, 0.1)	Error
	Res-3	P08 - - Insertar empleado (1, 23, "Juan", "Pérez", 5000, 12/11/2019, 12/12/2019, 0.1)	Error
RF-06		(…)	Correcto

¿Y si alguien cambia la BD? ¿Se invalidan todas las pruebas?

Buenas prácticas

El código de los casos de prueba debería ser suficientemente independiente(*) del contenido de la BD

- Ejemplo: que las pruebas no dependan de valores concretos de IDs u otros atributos
- (*) Independiente hasta cierto punto; para poder realizar las pruebas es necesario tener cierto control sobre la BD (p.ej: asumir que en el setup se han insertado un determinado número de entidades); **lo importante es que el código no dependa de valores concretos de atributos** (p.ej.: no asumir que existe un empleado con empleadoId=5 o que el empleado con empleadoId=5 pertenece al departamento con departamentoId=7)

Solución:

- Añadir código de inicialización a las pruebas para recuperar los valores que hay en un momento dado en le BD como contexto y hacer las pruebas sobre esos valores
    - Ejemplo: recuperar el primer departamento -> departmentId=4, hacer la prueba de inserción de empleado con ese departmentId

Resumen

Tema 13: Pruebas Software

Introducción a la Ingeniería del Software y los Sistemas de Información I

Ingeniería Informática – Tecnologías Informáticas

Departamento de Lenguajes y Sistemas Informáticos