Sesion 4

Diseño de pruebas: Caja negra

- Especificación ’S’ es complementaria a las técnicas de Código ‘P’
- Ambos técnicas de diseño (P y S) nos llevan a una tabla de casos de prueba, con el objetivo de tener un mínimo de filas de casos de prueba (EFECTIVO y EFICIENTE)
- Partimos de la especificación, método denominado Caja negra o funcional testing.
  - A partir de la especificación obtenemos un subconjunto.
  - Nunca me voy a dar cuenta que alguien a implementado fuera de la especificación.
  - No necesito que el código este implementado para los casos de prueba.
  - Que tienen en comun los métodos de este tipo:
    - Analizar la especificación ’S’ y obtengo otra representación de ’S’ en particiones para visualizar mejor los comportamientos de ’S’.
    - Obtener dichos comportamientos según algún criterio
    - Conseguir los casos de prueba.
- Hay muchos casos de prueba, el que nos interesa es el de particiones equivalentes.
- En caja negra hablamos de particiones en comportamiento
- Mis comportamientos tienen unas entradas y unas salidas, lo que voy a hacer es todas las entradas y salidas las voy a dividir en mas pequeñas donde la suma de ellas será el total.
- Voy a particiones las entradas y salidas.
  - Nunca puede ocurrir nunca es que una partición de entrada tenga varias salidas.
  - Garantiza que nuestras pruebas realice todos los comportamientos al menos una vez.
  - Estos comportamientos los vamos a denominar como Válidos e Inválidos.
- Sistematicidad y particionamientos

- Cada punto verde es lo que decido probar (conjunto T). Vamos a suponer que soy capaz de hacer división en grupos. Permite agrupar en particiones equivalentes.
  - Hay que tocar al menos una de cada partición (efectivo) y para ser eficiente habría que tocar una partición una vez.
  - Máximo numero de errores (efectivo)
  - En el menor numero de casos de prueba (eficiente)

- ¿Cómo identificamos cada partición?
  - Las particiones son Condiciones. (ej. si se cumple que blablabla eres miembro de X)
  - Cogemos entradas y salidas y aplicamos condiciones
  - En el ejemplo del triangulo:
    - Podemos coger entradas y salidas y aplicar condiciones. O puedo coger condiciones a un conjunto de variables de entradas y salidas (a,b,c <20 & >0)
    - Hay que tener muy claro cuantas entradas hay para saber que voy a particionar.
  - Invalido: todas aquellas etiquetas que nos van a dar un error (error esperado), estas las vamos a probar una a una (le damos un trato especial)
  - HEURISTICAS:
    - Rango de valores: 3 trozos, dentro del rango Validas, fuera(por la izq y por la der) Invalido
    - Numero de valores: 3 trozos, entre 1 y N valido, mayor numero y vacío Invalido
    - Conjunto de valores: Valida los que pertenecen al conjunto e invalida con los que no
    - Si cada uno de los valores del conjunto se va a tratar diferente se divide ese conjunto.
    - DEBE SER, dos particiones, cuando se cumple y cuando no.
    - 6)
  - Identificar los casos de prueba
    - Nos fijamos solo en las particiones validas y vamos escogiendo el máximo numero de particiones validas distintas.

    - Cuando acabamos con las validas, empezamos a llenar filas invalidas y únicamente UNA clase invalida por fila.

- METODOS
  - Cuantas entradas hay y cuantas salidas?
  - Quiero ver todas las entradas y todas las salidas
  - En la salida no se aplica la HEURÍSTICA.
  - Ahora hay que ponerlo en la tabla
    - Relleno con validas
    - Empiezo con las invalidas
    - Detalles:
      - Al rellenar las celdas, debemos variar (numeros, caracteres…..)
      - Siempre valores concretos.
      - Si una entrada va a depender de otra, la agrupamos.
- Consejo
  - A cada entrada una letra (la que queramos).
  - Debemos asumir que hay algunas cosas de Java que hay que añadir (como por ejemplo los objetos).
  -