Tema 4: Pruebas funcionales PA y BVA

- **Tipos de errores:**
  - Calculo.
  - Lógica: Definición incorrecta de una condición.
  - Entrada/Salida: Descripción incorrecta, mala conversión o formato inadecuado.
  - Transformación de datos: Incorrecto acceso o transición de datos.
  - Interfaz: Comunicación incorrecta con otros componentes.
  - Definición de datos.
- Para realizar las pruebas del software es necesario acceder a especificaciones del componente, el código fuente y código objeto. Eso nos permite ver todas las posibles combinaciones entre los elementos que hay que probar.
- **Pruebas Unitarias:** Verifican la unidad mas pequeña de software, el método.
  - **Pruebas funcionales:** No conocemos el código fuente, ya que hemos escrito las pruebas, pero todavía no hemos escrito el código. Por lo que no se pueden probar todos los casos.
  - **Pruebas estructurales:** Conocemos el código fuente, por lo que podemos probar todos los casos.
- **Pruebas de Caja Gris**: Se tiene acceso a la estructura interna de datos y algoritmo con el propósito de definir los casos de prueba. Utiles para identificar clases de equivalencia y valores limite. No tenemos el código pero tenemos idea de como funciona.
- **Pruebas funcionales o de Caja Negra:** Tratan de reducir el número de casos de prueba a un nivel fácil de gestionar. Manteniendo un cobertura razonable. Se pueden usar clases de equivalencia.
  - **Clases de equivalencia:** Agrupa varias pruebas con valores que se procesan de la misma manera o deberían proporcionar el mismo resultado. Todos prueban el mismo procesamiento, si una prueba detecta un error el resto también lo hará. Hay que considerar:
    - **Clases validas:** Casos de procesamiento normal del método. Un caso de prueba puede considerar varias validas, se pueden englobar.
    - **Clases invalidas**: Casos relacionados con situaciones de error. Por cada invalida a una clase de equivalencia.
    - **Definir un caso de prueba:**
      - Identificados.
      - Valores de entrada, indicar un valor para cada parámetro de entrada y claves de equivalencia.
      - Resultados esperados.
    - **Reglas identificar clases de equivalencia:**
      - **Rangos de valores continuos:** Identificar el limite inferior, superior y N particiones validas.
      - **Valores discretos de un rango de valores permisibles:** Una clase valida y dos invalidas, una posibilidad inferior y otra superior.
      - Si el dato **no es un intervalo numérico:** Una clase valida para cada valor valido y otra no valida para el resto.
      - **Numero de valores de entrada:** Identificar el numero mínimo y máximo, y elegir una clave valida y dos invalidas.
      - Otra aproximación para utilizar clases de equivalencia consiste en considerar las salidas.
    - **Aplicabilidad y Limitaciones:**
      - Reduce significativamente el número de casos de prueba.

        - Es un sistema apropiado para valores incluido en rangos o en conjuntos preestablecidos.
        - Entradas o salida que se puedan particionar de acuerdo a requisito o precondiciones.

- **Valores en los limites:** Son muy importante, gran fuente de problemas. Primero hay que encontrar las clases de equivalencia.
    - Probabilidad de que los defectos sean más frecuentes en los valores limite.
    - Considera valores en los limite del intervalo, justo antes, en y justo después.
- **Procedimiento**:
    - Identificar las clases de prueba.
    - Identificar los limite de cada clase de equivalencia.
    - Generar los casos de prueba para cada valor limite considerando las reglas.
- **Reglas para identificar valores limite:**
    - **Valores limite para un rango continuo de entradas:** Considerar un valor antes, en y después del limite inferior, y un valor en y después del limite superior.
    - **Valores limite para un rango discreto de entrada**: Considerar el primero, segundo, penúltimo y ultimo. O el mas pequeño, el siguiente, el ultimo y su anterior.
    - **Valores limite de las salidas producidas:** Aplicar la regla anterior pero con salidas.
- **Aplicabilidad y limitaciones:**
    - Dificultad para formalizar el concepto de valores marginal y limite.
    - Este análisis es mas intuitivo y requiere heurística(para tener un método).
    - Reduce significativamente el número de pruebas.
    - Esta dirigido para valores dentro de rangos o conjuntos.
    - La entrada o salida se deben poder partición a y los limites identificar.

- **Análisis Sintáctico:** Solo se aplica para entradas, y cuando se pueden modelar como gramáticas.
    - Permite reducir el número de casos de prueba a un nivel fácil de gestiona mientras se mantiene un cobertura razonable.
    - **Aplicaciones y limitaciones:**
        - Reducen el numero de casos de prueba, que se generan y ejecutan.
        - Esta dirigido para **entradas que se pueden modelar como gramáticas**.
        - Se puede utilizar tanto para pruebas unitarias como de integración.
        - En pocos casos a nivel de sistema y no se recomienda para pruebas de aceptación.
    - **Procedimiento**:
        - Definición de la gramática.
        - Creación del árbol de derivación.
        - Identificación de los casos de prueba.
        - Automatización de los casos de prueba.
    - **Definir una gramática:**
        - Debe ser de tipo 2 o tipo 3: Regular e independiente de contexto.
        - Un único símbolo no terminal a la izquierda.
        - No existan símbolos Lambda.
        - Las gramáticas recursivas son problemáticas porque el árbol de derivación asociado seria infinito.
    - **Creación del árbol de derivación**: Se hace usando a gramática del paso anterior.

- Cada símbolo terminal o no terminal será un nodo diferente.
- Los nodos se numeran empezando por 1.
- Debe de diferenciarse por niveles que nodos son terminales y no terminales.

- **Identificación de los casos de prueba:**
  - Se obtienen del análisis del árbol y se dividen en dos partes: entradas validas e invalidas.
  - Para **identificar las entradas validas**:
    - Se producen casos de prueba de tal forma que todos los nodos no terminales estén cubiertos.
    - Se repite el anterior hasta cubrir al menos una vez todos los nodos terminales.
  - Para **identificar entradas invalidas**:
    - Hay demasiadas por lo que se consideran una muestra significativa de las mismas.
    - Para los nodos no terminales se procede a su omisión y su adición. La adición de nodos puede producir un gran número de casos de prueba, siendo semánticamente más difícil de generar.
    - Para los nodos terminales debe procederse también a su modificación. Se simula mediante errores tipográficos, siendo aconsejable no someter a pruebas grandes combinaciones de errores. La explosión combinatoria seria enorme y la prueba poco realista.
  - RECORTAR Y MIRAR LAS DOS ULTIMAS DIAPOSITIVAS.