# Análisis y especificación de los Requisitos del Software

- **Entrada**: Las necesidades se encuentran en la cabeza de alguien (ideas abstractas)
- **Salida** : Un detalle preciso de lo que será el sistema futuro: **SRS** (*System Requirements Specification*).


- Identificar y especificar los requisitos (involucra **interacción con la gente** y no puede automatizarse).
- **Análisis**: *Entender* al cliente. Descubrir qué se pretende del producto. Ida y vuelta entre el analista y el cliente.
  Un buen analista debe asesorar y convencer y que el cliente acepte en lo que uno es bueno.

## SRS

Debe dar una clara comprensión de lo que se espera de un determinado software.

La SRS es el medio para **reconciliar las diferencias** y **especificar las necesidades** del cliente/usuario de manera que todos entiendan.

Hay abogados y escribanos involucrados. Es un **contrato** o acuerdo entre cliente/usuario y quién suministrará el software. No debe ser ambigua.

## Requerimientos del software

Hay que tenerlos en concreto para producir el software.

---

**Requerimientos** (IEEE)

1. Una condición o capacidad necesaria de un usuario para solucionar un problema o alcanzar sus objetivos.
2. Una condición o capacidad necesaria que debe poseer o cumplir un sistema [...].

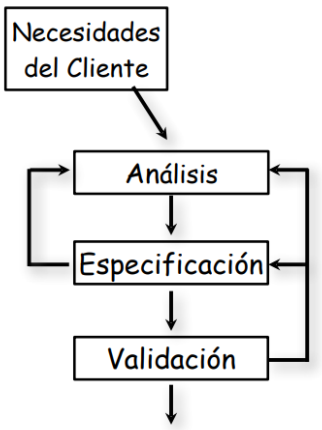
---

## ¿Por qué la SRS es necesaria?

1. **Una SRS establece el contrato entre el ciente y el proveedor respecto a lo que el software va a hacer.**
   - El cliente no comprende el proceso de desarrollo de software. Y el desarrollador no conoce el problema ni su área de aplicación.
   - La SRS ayuda a comprender las necesidades del cliente (tanto para el cliente mismo como para el desarrollador).
2. **La SRS provee un referencia para la validación del producto final.**
   - (Determina el resultado correcto). Verificación: "el software satisface la SRS".
3. **Una SRS de alta calidad es esencial para obtener un software de calidad.**
   - Los errores de requerimientos solo se manifiestan en el software final. Los defectos de requerimientos no son pocos. Ejemplo: 45% de los errores en testing correspondían a errores de requerimientos (siendo el 25% del total de defectos encontrados en el proyecto).
4. **Una buena SRS reduce los costos de desarrollo.**
   - Contribuye a minimizar cambios y errores.
   - Los cambios de requerimientos pueden costar demasiado (hasta un 40%).
   - En cada etapa es más caro el precio de encontrar y resolver un error de los requerimientos.

## Proceso de requerimientos

**Secuencia de pasos que se necesita realizar para convertir las necesidades del usuario en la SRS**




Es iterativo y en paralelo: Algunas partes pueden estar siendo especificadas mientras otras están aún bajo análisis.

**Actividades básicas**

1. Análisis del problema o requerimientos
2. Especificación de los requerimientos.
3. Validación.

La transición del análisis a la especificación es complicada.

- Objetivo del análisis: comprender la estructura del problema y su dominio (componentes, entrada, salida).
- La especificación se enfoca en el **comportamiento externo.**
- Se recolecta **más información (o distinta) de la necesaria** para la especificación.

Los métodos de análisis son similares a los de diseño, pero con **objetivos y alcances distintos**.
El análisis trata de entender. El diseño trata de resolver.

## Análisis del problema

**Objetivo**: Lograr una buena comprensión de las necesidades, requerimientos y restricciones del software.

El análisis incluye:

1. Entrevistas con el cliente y usuarios.
2. Lectura de manuales.
3. Estudio del sistema actual.
4. Ayudar al cliente/usuario a comprender las nuevas posibilidades.

Es importante:

- Obtener la información necesaria

- Lluvia de ideas: discutir. **Interactuar** con el cliente para establecer propiedades

- Relaciones interpersonales
- Habilidades de comunicación
- Organizar la información
- Asegurar **completitud**
- Asegurar **consistencia**
- Evitar diseño interno: evitar tratar de resolver el problema.

  **Particionar el problema**
  **Estrategia**: Descomponer e problema en pequeñas partes, comprenderlas y relacionarlas entre ellas.

Comprender los sub-problemas y la relación respecto a:

- Funciones (análisis estructural)
- Objetos
- Eventos del sistema

## Enfoque informal

No hay una metodología definida; la información se obtiene a través del análisis, observación, interacción, discusión,... .
No se construye un modelo formal del sistema.
La información recogida se plasma y organiza directamente en la SRS, la cual es el objeto de revisión con el cliente.

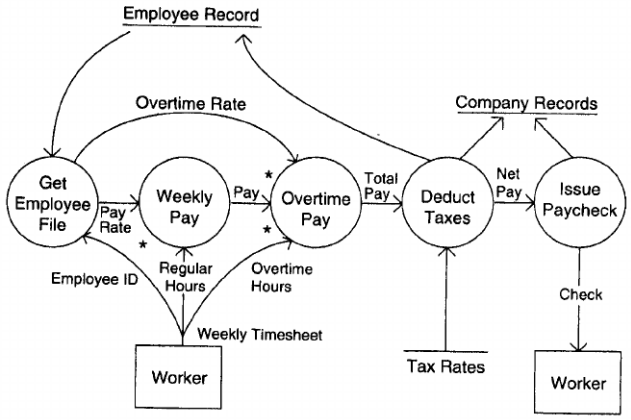
## Modelado de flujo de datos

Se enfoca en las funciones realizadas en el sistema.
Para el modelado se usan **diagramas de flujos de datos** (DFD) y **descomposición funcional**.

### DFD

Interpretación del sistema como una transformación de entradas y salidas, donde las transformaciones se realizan a través de "transformadores/procesos".
Método de análisis estructurado.

**Un DFD es una representación gráfica del flujo de datos a través del sistema.**




- **Transformadores**: Burbujas. Sus nombres son verbos.
- **Datos que fluyen**: Flechas con dirección entre burbujas. Sus nombres son sustantivos.
- **Fuentes** o **resumideros**: Rectángulos. Generador/consumidor de datos (usualmente fuera del sistema)
- **Interfaz entre el sistema y el mundo exterior**: Flechas entre burbujas y rectángulos.
- **Archivos externos**: Línea recta etiquetada. (Almacén de datos).
- **Multiples flujos de datos**: se representa con * (significa 'and'). O con + (significa 'or')

Usualmente se muestran sólo las entradas/salidas más significativas y se ignoran las de menor importancia (como los mensajes de error).
No debe decir **cómo** se realiza una transformación.
No hay loops ni razonamiento condicional. Un DFD **no** es un diagrama de control, no debería existir diseño ni pensamiento algorítmico.

## Cómo dibujar el DFD de un sistema:

1. Identificar entradas, salidas, fuentes, sumideros del sistema
2. Avanzar identificando los transformadores de más alto nivel para capturar la transformación completa.
   - Trabajar consistentemente desde la entrada hasta la salida si se complica, cambiar el sentido (de la salida a la entrada).
3. Refinar los transformadores de alto nivel con transformaciones más detalladas cuando estén definidos.
   - No mostrar nunca lógica de control, si se comienza a pensar en términos de loops/condiciones: parar y recomenzar.
   - Etiquetar cada flecha y burbuja. Identificar cuidadosamente las entradas y salidas de cada transformador.
4. Intentar dibujar grafos de flujo de datos alternativos antes de definirse por uno.

## DFD en niveles

El DFD de un sistema puede resultar muy grande → organizar jerárquicamente.
Se realiza un proceso Top-Down:

- Comenzar con un DFD de nivel superior, abstracto, conteniendo pocas burbujas.
- Luego dibujar un DFD por cada burbuja. Preservando la E/S original para ser consistente.

### Errores:

- Consistencia no preservada
- Flujo de datos irrelevantes
- Flujos de datos omitidos
- Flujos de datos sin etiquetar
- Procesos omitidos
- Inclusión de análisis de control : se resuelve el COMO

### Diccionario de datos

Muestra la estructura de los datos.
El diccionario de datos se representa con las flechas del DFD que están etiquetadas con items de datos; y define con mayor precisión los datos en un DFD.
Se puede usar expresiones regulares para identificar los datos.

Ejemplo:

```
weekly timesheet = Employee_name +
                   Employee_Id +
                   [Regular_hours + Overtime_hours] *

pay.rate = [hourly | daily | weekly] + Dollar_amount

Employee_name = Last + First + Middle.initial

Employee_Id = digit + digit + digit + digit
```

## Método de análisis estructurado

1. Dibujar el diagrama del contexto.
2. Dibujar el DFD del sistema existente: refinar el anterior. Es importante la comunicación con el cliente.
3. Modelado del sistema propuesto. Dibujar el **DFD del sistema propuesto** e identificar la frontera hombre-máquina.

## Diagrama de Contexto

- Identifica el contexto.
- Es un DFD con un único transformador (el sistema), con entradas, salidas, fuentes y sumideros del sistema.

## DFD del sistema existente

- El **sistema actual*** se modela tal como es con un DFD con el fin de comprender el funcionamiento.
- Se refina el diagrama de contexto.
- Pueden usarse DFD en niveles jerárquicos.
- Para obtenerlo se debe interactuar intensamente con el usuario.
- El DFD obtenido se valida junto a los usuarios.

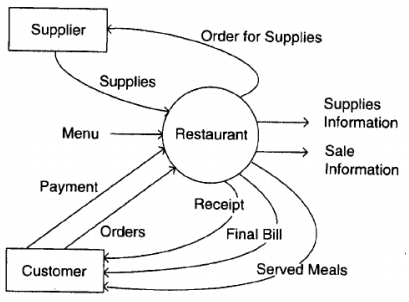
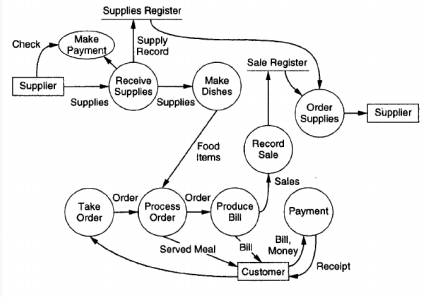
* no es necesariamente de software. Es un sistema que queremos automatizar.

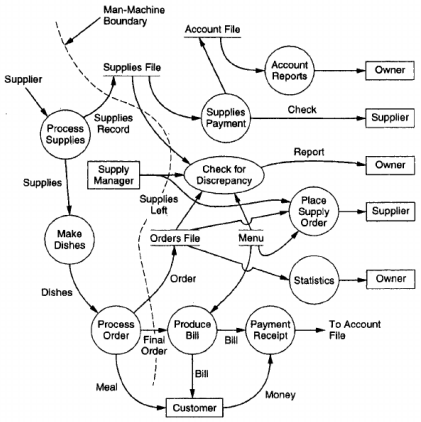
## Modelado del sistema propuesto

- El DFD debe modelar el sistema propuesto completo: ya sean procesos automatizados o manuales.
- Validar con el usuario
- Establecer la frontera hombre máquina: qué procesos se automatizarán y cuáles permanecerán manuales.
- Mostrar claramente la interacción entre los procesos manuales y los automáticos

## Ejemplo

Una dueña de restaurante, quiere automatizar algunas partes del negocio. Entrevistas y cuestionarios. Recolectamos información en general.

| Paso | Diagrama | Comentarios |
|---|---|---|
| Diagrama de contexto |  | **Partes involucradas**: clientes (dueña) y usuarios (mozos, operador de la caja). No todos los aspectos serán computables. |
| DFD del sistema existente |  | Se interactúa con el cliente. Pueden agregar nuevos aspectos. |

| Paso | Diagrama | Comentarios |
| --- | --- | --- |
| DFD del sistema propuesto |  | Se genera el sistema propuesto. |
| Diccionario de datos del DFD del sistema | Supplies_file = [date + [item_no + quantity + cost]* ]*<br>Orders_file = [date + [menu_item_no + quantity + status]* ]*<br>status = satisfied \| unsatisfied<br>order = [menu_item_no + quantity]*<br>menu = [menu_item_no + name + price + supplies_used]*<br>supplies_used = [supply_item_no + quantity]*<br>bill = [name + quantity + price]* +<br>total_price + sales_tax + service_charge + grand_total<br>discrepancy_report = [supply_item_no +<br>amt_ordered + amt_left + amt_consumed + descr]* | Formato de los datos, a través de expresiones regulares |

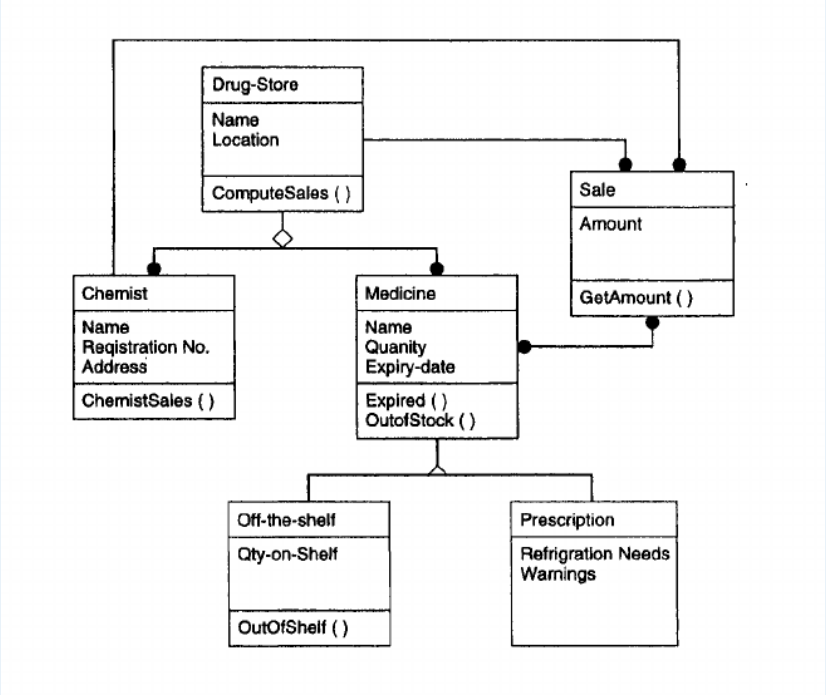
## Modelado orientado a objetos

**Ventajas**:

- Más fácil de construir y mantener
- La transición del *análisis* orientado a objetos al *diseño* orientado a objetos es más simple
- El análisis orientado a objetos es más resistente/adaptable a cambios porque **los objetos son más estables que las funciones**

Un objeto **provee servicios** o **realiza operaciones**. Solo se accede a ellos a través de estos.

### Notación




Notación del modelado orientado a objetos

- **Diagrama de clase**: Representa gráficamente la estructura del problema.
- **Clase**: Conformada por: nombre, atributos y servicios o métodos.

| clase | ejemplo |
| --- | --- |
| nombre de la clase | Prescription |
| atributos | Refrigration Needs |
| métodos | Warnings |

- **Estructura de generalización-especialización**: utilizado para representar **herencia** de objetos (líneas que unen con un triángulo).
- **Multiplicidad de una relación** se representa con un círculo negro. Si hay dos puntitos negros es una relación N a N.
- **Estructura de agregación**: modela la relación "está conformada por.." y se representa con un rombo.
- **Asociación**: Representa una relación entre objetos de distintas clases (lineas). Asociación no es herencia.

## Proceso de modelado

1. Identificar los **objetos** y las **clases** en el dominio del problema: los objetos se corresponden con sustantivos.
2. Identificar **estructuras**: herencia
3. Definir las clases identificando cuál es la información del estado que ésta encapsula (es decir, los **atributos**). Características. No agregar atributos innecesarios
4. Identificas **asociaciones**: hacer jerarquía de las clases. Las relaciones entre los objetos de las distintas clases, ya sea en la jerarquía o a través de llamadas a

métodos.
5. Definir **servicios**: proveen el elemento activo en el modelado OO

- Pasos más significativos para el modelado orientado a objetos.
  Identificar objetos y clases

En estos pasos, puede haber "huecos" por ejemplo clases que todavía no se sabe cuales son los atributos.

Clase de modelados: 2a-modelados

## Prototipado

Un prototipo de software se puede definir como una **implementación parcial** de un sistema cuyo propósito es **aprender** algo del problema a resolver o del enfoque de la solución.

El prototipado enfatiza que la experiencia práctica es la mejor ayuda para entender las necesidades.
Se necesita una constante interacción con el cliente/usuario durante el prototipado para entender sus respuestas.

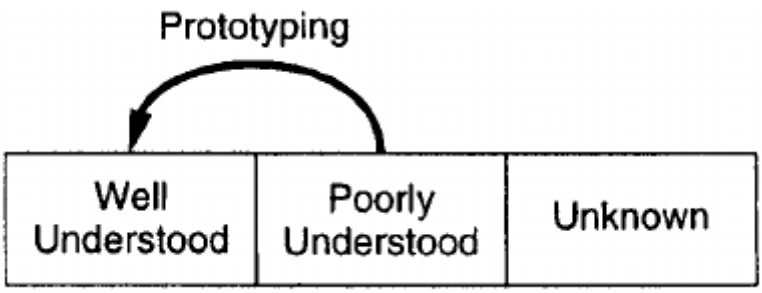
**Enfoques:**

- *Throwaway:* (Descartable) Se construye con la idea de que va a descartarse luego de que se complete el análisis.
- *Evolutionary:* (Evolutivo) Se construye con la idea de que eventualmente se va a convertir en el sistema final.

Para el análisis y entendimiento el prototipado descartable es más adecuado.

Los requerimientos se pueden dividir en tres grupos:

- Los entendidos.
- Parcialmente comprendidos: son los que deben ser incluidos en el prototipado.
  - Críticos en el diseño: En estos debería enfocarse el prototipo.
  - No críticos: Después pueden ser fácilmente incorporados en el sistema.
- Desconocidos.




Si el conjunto de requerimientos es substancial (en particular si se trata de los requerimientos críticos), entonces debería construirse un prototipo

descartable.

**Tipos de prototipado:**

- *Vertical*: Una parte elegida del sistema, que no ha sido bien comprendida, se construye completamente.
- *Horizontal*: El sistema es organizado como una serie de capas y alguna capa es el foco del prototipo.

Para que sea posible hacer un prototipo en la etapa de análisis de requerimientos, el **costo** debe mantenerse **bajo**. Por eso solo se incluyen las características valiosas para la experiencia de usuario, solo se produce mínima documentación del desarrollo y se reduce el testing. Además, la eficiencia no es una preocupación en el prototipado, y usualmente se usan lenguajes interpretados de alto nivel para hacer prototipos.

Gracias al prototipado se pueden reducir substancialmente los errores de requerimientos y la cantidad de cambios de requerimientos.

## Especificación de los requerimientos

**¿Por qué los DFD, modelos DC, o prototipado no son SRS?**

| Modelado | SRS |
|---|---|
| Se enfoca en la estructura del problema. | Se enfoca en el comportamiento **externo** del sistema. |
| Ej: UI no se modela. | Ej: UI sí está en la SRS. |
| Solo está la estructura. | Hay tratamiento de errores, requerimientos en el desempeño, conformidad de estándares recuperación, ... |

- La transición del modelo a la SRS no es directa: la SRS NO es una formalización del modelo.
- Lo que se transporta del análisis a la especificación es el **conocimiento adquirido** sobre el sistema.

## Características de la SRS

### Correcta:

Cada requerimiento representa precisamente alguna característica deseada por el cliente en el sistema final.

`SRS -> representa -> cliente`

### Completa:

Todas las características deseadas por el cliente están descritas.
`cliente -> determina -> SRS`

### No ambigua:

Cada requerimiento tiene exactamente un significado. Que no haya cosas subjetivas: "rápido" "lindo" . Los lenguajes formales ayudan a "desambiguar".

### Consistente:

Que no se contradiga. Que toda la información esté ordenada ayuda a que sea consistente (es más fácil detectar inconsistencias).

### Verificable:

Casa requerimiento es verificable (existe algún proceso efectivo que puede verificar que el software final satisface el requerimiento). Es esencial la No-ambigüedad, y que la SRS sea comprensible (debe poder ser revisada por el desarrollador, el usuario y el cliente).

### Rastreable:

Se debe poder determinar el origen de cada requerimiento y cómo éste se relaciona a los elementos del software.
`requerimiento está en SRS <=> elemento está en el producto final`

`=>` Dado un requerimiento se debe poder detectar en qué elementos de diseño o código tiene impacto.
`<=` Dado un elemento de diseño o código se debe poder rastrear que requerimientos está atendiendo.

### Modificable:

La estructura y estilo de la SRS es tal que permite incorporar cambios fácilmente preservando completitud y consistencia.
Debe suceder que no haya requerimientos en varios lugares (no haya redundancia).

### Ordenada en aspectos de importancia y estabilidad:

Los requerimientos pueden ser *críticos*, *importantes pero no críticos*, *deseables pero no importantes*.
Algunos requerimientos son *esenciales* y difícilmente cambien con el tiempo.
Otros son propensos a cambiar.

Se necesita definir un orden de prioridades en la construcción para reducir riesgos debido a cambios de requerimientos.

## Componentes de la SRS

### Requerimientos sobre funcionalidad

- Conforma la mayor parte de la especificación
- Especifica toda la **funcionalidad** que el sistema debe proveer.*
- Especifica qué **salidas** deben producir para cada entrada dada y las relaciones entre ellas.
- Describe todas las **operaciones** que el sistema debe realizar.
- Describe las **entradas válidas** y las **verificaciones de validez** de la entrada y salida.*

* Sirve para *testing*

### Requerimientos sobre desempeño

Todos los requisitos se especifican en términos medibles. Porque `medibles => verificables`.

#### Requerimientos dinámicos:

Especifican restricciones sobre la ejecución

- Tiempo de respuesta.
- Tiempo esperado de terminación de una operación dada
- Tasa de transferencia o rendimiento
- Cantidad de operaciones realizadas por unidad de tiempo
- En general se especifican los rangos aceptables de los distintos parámetros

#### Requerimientos estáticos o de capacidad

**No** imponen restricción en la ejecución

- Cantidad de terminales admitidas.
- Cantidad de usuarios admitidos simultáneamente.
- Cantidad de archivos a procesar y sus tamaños.

### Restricciones de diseño

- Factores en el entorno del cliente que pueden restringir las decisiones de diseño:
  - Estándares
  - compatibilidad con otros sistemas
  - limitaciones de hardware
  - requerimientos de confiabilidad

    - tolerancia a fallas, o respaldo
    - seguridad.

### Requerimientos de Interfaces externas

- Todas las interacciones del software con gente, hardware, y otros software deben especificase claramente.
- Estos requerimientos también deben ser precisos para asegurar verificabilidad (evitar cosas como "la interfaz debe ser amigable").

## Lenguajes de especificación

Realidad: **lenguaje natural**. No usar conjugación de verbos. La gran ventaja es que el cliente entiende.

Los **lenguajes formales** deben ser fáciles y precisos. Vemos en ingeniería II, se usan para sistemas muy críticos.

Lo ideal es mezclar ambos enfoques.

## Alcance

- ¿Qué cosas entran o no en el proyecto?
- Objetivos, entregables y requerimientos. Tiempos de entrega.
- ¿Cuáles son la prioridades? Para que hacer primero.
- Criterio de aceptación (¿qué usuario lo aceptará?)
- Limitantes presupuestarios.

## Estructura de un documento de requerimientos

Las estructuras estandarizadas ayudan a la comprensión por parte de otros y ayudan a la completitud.

---

**Guía de la IEEE**

1. Introducción:
    1. Propósito
    2. Alcance
    3. Definiciones, acrónimos y abreviaciones
    4. Referencias
    5. Visión general
2. Descripción global
    1. Perspectiva del producto
    2. Funciones del producto
    3. Características del usuario

    4. Restricciones generales
    5. Suposiciones y dependencias
3. Requerimientos específicos
    1. Requerimientos de interfaz externa
        1. Interfaces del usuario
        2. Interfaces con el hardware
        3. Interfaces con el software
        4. Interfaces de comunicación
    2. Requerimientos funcionales
        1. Modo 1: Requerimientos funcionales 1.1 y 1.n
        2. Modo m: Requerimientos funcionales m.1 y m.n'
    3. Requerimientos de desempeño
    4. Restricciones de diseño
    5. Atributos
    6. Otros requisitos

---

En cada requisito funcional se especifica la entrada requerida, salida esperada, interfaces c/u con todos sus detalles.
En general, no se especifican algoritmos solo la relación E/S.

## Especificación funcional con Casos de Uso

- Adecuado para sistemas interactivos.
- Consiste en especificar cada función provista por el sistema. (enfoque tradicional).
- Cada sistema tiene muchos casos de uso.
- Siempre usar reglas estrictas.
- Son útiles porque son muy entendibles para los usuarios y para los clientes.
- Permiten brainstorming con los clientes.

### Formato de los casos de uso

- Caso de uso # `nombre del caso de uso`
- Actor primario: `nombre del AP`
- (Pre-condición: `descripción de la pre-condición`)
- (Ámbito: `subsistema al cual se aplica el caso de uso`)
- Escenario exitoso principal: `paso 1, ..., paso n`
- Escenarios excepcionales: `excepción 1, ..., excepción n`
- (Post-condiciones)

#### Caso de uso

- Los casos de uso capturan el comportamiento del sistema como **interacción** de los actores con el sistema.
- Un caso de uso es una colección de muchos escenarios.
- El nombre del caso de uso especifica el objetivo del actor primario.
- Pueden ordenarse jerárquicamente.
- Se enfocan en el comportamiento externo.
- La forma básica es textual. Existen notaciones gráficas (diagramas) como soporte.
- Los casos de uso **no** forman la SRS completa, sólo la parte funcional.
- Los casos de uso se enumeran para referencias posteriores.
- Se pueden organizan en jerarquías (subobjetivos)

## Actor

**Persona** o **sistema** que interactúa con el sistema propuesto para alcanzar un objetivo.

Un actor es una **entidad lógica** y hay actores *receptores* y actores *transmisores* (y pueden ser mismo individuo).

- El **Actor Primario** es el actor principal que inicia el caso de uso. El caso de uso debe satisfacer su objetivo.

## Escenario

- Es un conjunto de acciones realizadas con el fin de alcanzar un objetivo bajo determinadas condiciones.
- Se representan en secuencia pero no necesariamente esa es su implementación.
- La lista de acciones puede contener acciones que no son necesarios para el objetivo del actor primario. Sin embargo el sistema deba asegurar que todos los objetivos puedan cumplirse.
- Para cada punto actúa uno de los actores, en el siguiente debe ser otro actor el que realiza las acciones.
- No hay nada del manejo interno del sistema.
- Se pueden mencionar casos de uso de otra jerarquía, que estén especificados aparte.
- `paso_n = actor1 + acción1, ..., actor1 + acción_m`
- `paso_n+1 = actor2 + respuesta1, ...`

- El **escenario exitoso principal** sucede cuando todo funciona normalmente y **se alcanza el objetivo**. Mientras más exhaustivo, mejor. Porque estos casos serán usados como oráculo.

- Los **escenarios alternativos** (de extensión o de excepción) suceden cuando algo sale mal y **el objetivo no puede ser alcanzado**.
    - Las listas de excepciones no son exhaustivas: Solo se ponen los más básicos
    - Se ponen los que son errores específicos y necesitan un salida específica.
    - `escenario de error = error + respuesta`

### Elaboración de los Casos de Uso

Pueden elaborarse haciendo refinamientos paso a paso.
Se presentan cuatro niveles de abstracción:

1. **Actores y objetivos**
    1. Preparar una lista de actores y objetivos.
    2. Proveer un breve resumen del caso de uso.
    3. Evaluar completitud.
    - Esto define el ámbito del caso de uso
2. **Escenarios exitosos principales**
    1. Para cada caso de uso, expandir el escenario principal.
    2. Revisar para asegurar que se satisface el interés de los participantes y actores.
3. **Condiciones de falla**
    - Siempre listar las que parezcan más importantes, la exhaustividad es casi nula.
    1. Por cada paso, identificar cómo y por qué puede fallar.
    2. Listar las posibles condiciones de falla para cada caso de uso.
4. **Manipulación de fallas**
    1. Especificar el comportamiento del sistema para cada condición de falla.
    - El realizar esta etapa emergerán nuevas situaciones y actores.

Se deben usar reglas de buena escritura técnica:

- Usar gramática/oraciones *simples* y *claras*.
- Especificar claramente todas las partes del caso de uso.
- Cuando sea necesario, combinar o dividir pasos.

## Validación de los requerimientos

## Errores típicos de la SRS

Hay muchas posibilidades de malentendidos en el proceso de análisis y especificación.

| Error | % | La SRS no es.. |
|---|---|---|
| Omisión | 30% | Completa |
| Inconsistencia | 10 - 30 % | Consistente |
| Hechos incorrectos | 10 - 30 % | Correcta |
| Ambigüedad | 5 - 20 % | No ambigua |

## Proceso

Proceso de *inspección estándar*.
La SRS se revisa por un grupo de personas, conformado por autor, **cliente**, representantes de **usuarios** y de desarrolladores.

Se pueden detectar entre el 40% y el 80% de los errores de requerimientos.

Son útiles las **listas de control**: aspectos que deberían haberse tenido en cuenta para ir chequeando.

Ejemplo:

| | Checkear: |
|---|---|
| 1 | ¿Se definieron todos los recursos de hardware? |
| 2 | ¿Se especificaron los tiempos de respuestas de las funciones? |
| 3 | ¿Se definió todo el hardware, el software externo y las interfaces de datos? |
| 4 | ¿Se especificaron todas las funciones requeridas por el cliente? |
| 5 | ¿Son testeables todos los requerimientos? |
| 6 | ¿Se definió el estado inicial del sistema? |
| 7 | ¿Se especificaron todas las respuestas a las condiciones excepcionales? |
| 8 | ¿Los requerimientos contienen restricciones que pueda controlar el diseñador? |
| 9 | ¿Se especifican modificaciones futuras posibles? |

Hay herramientas automáticas o semi-automáticas que soportan lenguajes de especificación formal y permiten verificar consistencia, dependencias circulares, o propiedades específicas. También permiten simular para poder comprender completitud y corrección.

# Métricas

Para poder **estimar costos y tiempos**, y planear el proyecto se necesita "medir" el esfuerzo que demandará.
Una métrica es importante solo si es útil para el seguimiento o control de costos, calendario o calidad.

## Punto función

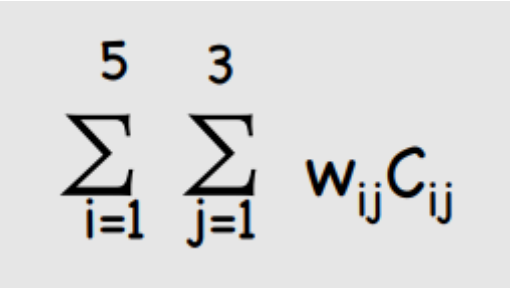
Se determina solo con la SRS.
Define el tamaño en términos de la "funcionalidad".
Métrica en términos de LOC.

### Tipo de funciones:

1. **Entradas externas:** Tipo de entrada (dato/control) externa a la aplicación.
2. **Salidas externas:** Tipo de salida que deja el sistema.
3. **Archivos lógicos internos:** Grupo lógico de dato/control de información generado/usado/manipulado.
4. **Archivos de interfaz externa:** Archivos pasados/compartidos entre aplicaciones.
5. **Transacciones externas:** Input/output inmediatos (queries)

Contar cada tipo de función diferenciado según sea *compleja*, *promedio* o *simple*. Y se pondera con un número `w_ij`.

`C_ij` determina la cantidad de funciones tipo "i" con complejidad "j"




Ajustar el UFP de acuerdo a la **complejidad del entorno**. Se evalúa según las siguientes características:

1. Comunicación de datos
2. Procesamiento distribuido
3. Objetivos de desempeño
4. Carga en la configuración de operación
5. Tasa de transacción
6. Ingreso de datos online
7. Eficiencia del usuario final
8. Actualización online
9. Complejidad del procesamiento lógico

10. Reusabilidad
11. Facilidad para la instalación
12. Facilidad para la operación
13. Múltiples sitios
14. Intención de facilitar cambios

Casa uno de estos items debe evaluarse como:

1. No presente: `p_i = 0`
2. Influencia insignificante: `p_i = 1`
3. Influencia moderada: `p_i = 2`
4. Influencia promedio `p_i = 3`
5. Influencia significativa `p_i = 4`
6. influencia fuerte: `p_i = 5

$$0.65 + 0.01 \sum_{i=1}^{14} p_i$$

```c
puntos_función = CAF * UFP;
// 1 punto función = 125 LOC en C
//                 =  50 LOC en C++ o Java
```

## Métricas de calidad

Se necesitan buenas métricas de calidad para **evaluar la calidad de la SRS**.

**Métricas de calidad directa**: Evalúan la calidad del documento estimando el valor de los atributos de calidad de la SRS.

**Métricas de calidad indirecta**: Evalúan la efectividad de las métricas de control de calidad usadas en el proceso en la fase de requerimientos.
**El proceso debe estar bajo control estadístico**.

---

Práctico