# **CLASE 1: Conceptos de Ingeniería de Software**

## Software

Instrucciones (programas de cómputo), procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación (IEEE). Se desarrolla, no se fabrica.

### Software libre

El término software libre (o programas libres) se refiere a libertad, tal como fue concebido por Richard Stallman en su definición.

En concreto se refiere a **cuatro libertades**:

* Libertad para ejecutar el programa en cualquier sitio, con cualquier propósito y para siempre.
* Libertad para estudiarlo y adaptarlo a nuestras necesidades. Esto exige el acceso al código fuente.
* Libertad de redistribución, de modo que se nos permita colaborar con vecinos y amigos.
* Libertad para mejorar el programa y publicar las mejoras. También exige el código fuente.

## Ingeniería de Software

Disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema incluyendo la evolución de éste, luego que se comienza a ejecutar.

Según la IEEE: 1. El uso de métodos sistemáticos, disciplinados y cuantificables para el desarrollo, operación y mantenimiento de software

2. El estudio de técnicas relacionadas con 1

La cuantificación rigurosa de recursos, procesos y productos es una precondición para optimizar productividad y calidad. Un Ingeniero de Software debe cumplir contratos en tiempo y costos como es normal en obras de Ingeniería. La Ingeniería de Software se ocupa de todo el ciclo de vida de un producto, desde su etapa inicial de planificación y análisis de requerimientos hasta la estrategia para determinar cuándo y cómo debe ser retirado de servicio.

* La Ingeniería de Software se desarrolla en un marco económico, social y legal.
* No debe utilizar su capacidad y habilidades de forma deshonesta, o de forma que deshonre la profesión.
* Confidencialidad, competencia, derechos de la propiedad intelectual, uso apropiado de las computadoras.

## Técnicas de comunicación

Debemos ser capaces de identificar correctamente, analizar y entender cuáles son los requerimientos de los usuarios o lo que los usuarios quieren que haga el sistema. El proceso y las técnicas que un analista de sistemas usa para identificar, analizar y entender requerimientos de sistema, son referidos como la **identificación de requerimientos.**

Los requerimientos del sistema especifican lo que el sistema de información deberá hacer o cuál propiedad o cualidad debe de tener éste. Los requerimientos del sistema que especifican lo que el sistema de información debe hacer son frecuentemente llamados **requerimientos funcionales**. Aquellos que especifican una propiedad o cualidad que el sistema debe tener con frecuencia son llamados **requerimientos no funcionales.**

Esencialmente, el propósito de la identificación de los requerimientos y su administración es identificar correctamente los requerimientos de CONOCIMIENTO, PROCESO y COMUNICACIÓN para los usuarios de un sistema nuevo.

Toda esta información se escribe en un **documento de definición de requerimientos**: documento formal que comunica los requerimientos de un sistema propuesto a involucrados clave y sirve como un contrato del proyecto de sistemas.

## TIPOS DE REQUERIMIENTOS

**Requerimientos funcionales**

* Describen una interacción entre el sistema y su ambiente. Cómo debe comportarse el sistema ante determinado estímulo.
* Describen lo que el sistema debe hacer, o incluso cómo NO debe comportarse.
* Describen con detalle la funcionalidad del mismo.
* Son independientes de la implementación de la solución.
* Se pueden expresar de distintas formas

**Requerimientos no funcionales**

* Describen una restricción sobre el sistema que limita nuestras elecciones en la construcción de una solución al problema.
* Requerimientos del producto
  + Especifican el comportamiento del producto (usabilidad, eficiencia, rendimiento, espacio, fiabilidad, portabilidad).
* Requerimientos organizacionales
  + Se derivan de las políticas y procedimientos existentes en la organización del cliente y en la del desarrollador (entrega, implementación, estándares).
* Requerimientos externos
  + Interoperabilidad, legales, privacidad, seguridad, éticos.

## Fuentes de Requerimientos

* **Stakeholder:** El término stakeholder se utiliza para referirse a cualquier persona o grupo que se verá afectado por el sistema, directa o indirectamente.
* Documentación: es importante que reúna o documente la información recopilada (es decir, los requerimientos del borrador) de una manera organizada, inteligible y significativa. Estos documentos iniciales van a suministrar la dirección de las técnicas de modelación que el analista de sistemas va a usar para analizar los requerimientos y determinar los requerimientos correctos del proyecto
* Especificaciones de sistemas similares.

### Puntos de Vista

Existen tres tipos genéricos de puntos de vista:

* Punto de vista de los **interactuadores**: representan a las personas u otros sistemas que interactúan directamente con el sistema. Pueden influir en los requerimientos del sistema de algún modo.
* Punto de vista **indirecto**: representan a los stakeholders que no utilizan el sistema ellos mismos pero que influyen en los requerimientos de algún modo.
* Punto de vista del **dominio**: representan las características y restricciones del dominio que influyen en los requerimientos del sistema.

# **1.1 Elicitación de Requerimientos**

Es el proceso de adquirir (“eliciting”) [sonsacar] todo el conocimiento relevante necesario para producir un modelo de los requerimientos de un dominio de problema.

Objetivos:

* Conocer el dominio del problema para poder comunicarse con clientes y usuarios y entender sus necesidades.
* Conocer el sistema actual (manual o informatizado).
* Identificar las necesidades, tanto explícitas como implícitas, de clientes y usuarios y sus expectativas sobre el sistema a desarrollar.

La elicitación de requisitos es una actividad principalmente de carácter social, mucho más que tecnológico. Los problemas que se plantean son por tanto de naturaleza psicológica y social, más que técnicos.

## Técnicas de elicitación / exploración

Generalmente un analista aplica varias de estas técnicas durante un proyecto de sistemas individual. Para poder seleccionar la técnica más adecuada a usarse en cualquier situación dada, los analistas de sistemas necesitan aprender las ventajas y desventajas de cada una de las técnicas de exploración.

Durante la exploración, los analistas de sistemas frecuentemente encuentran o analizan información que es de naturaleza sensible. Los analistas deben tener mucho cuidado para proteger la seguridad y la privacidad de los hechos o datos que se les hayan confiado.

**Métodos discretos:** son menos perturbadores que otras formas de averiguar los requerimientos. Se consideran insuficientes para recopilar información cuando se utilizan por sí solos, por lo que deben utilizarse junto con uno o varios de los métodos.

1. Muestreo de la documentación, los formularios y los datos existentes.
2. Investigación y visitas al lugar.
3. Observación del ambiente de trabajo.

1. El analista debe recopilar y revisar documentos relevantes para entender el proyecto, identificar a los propietarios y usuarios clave mediante el organigrama, y rastrear el historial que originó el proyecto. Esto incluye:

* Documentos sobre el problema: Estudios, minutas, notas, registros contables, reportes operativos, solicitudes de proyecto, entre otros.
* Documentos sobre la función del negocio: Declaración de misión, plan estratégico, objetivos del departamento, políticas, restricciones, procedimientos, bases de datos, etc.

Toda la documentación debe analizarse para verificar si está actualizada; la información obsoleta no debe descartarse automáticamente. Dado que sería poco práctico analizar todos los registros o datos disponibles, se emplean técnicas de muestreo para obtener una visión representativa del sistema.

2. Una segunda técnica de exploración es investigar a fondo el dominio del problema. Las revistas especializadas en computación y los libros de referencia son una buena fuente de información. Pueden suministrar información sobre cómo otros han resuelto problemas similares. La exploración del Internet y del intranet vía la computadora personal puede suministrar incontables cantidades de información.

3. La observación es una de las técnicas más efectivas de recolección de datos para aprender acerca de un sistema. La observación consiste en que el analista de sistemas se convierte en un observador de las personas y de las actividades con objeto de aprender acerca del sistema.

Lineamientos de la observación: Determinar quién y cuándo será observado, obtener el permiso de la persona y explicar el porqué será observado, mantener bajo perfil, tomar nota de lo observado, revisar las notas con la persona apropiada y no interrumpir a la persona en su trabajo.

Ventajas

* Datos confiables
* El analista puede ver exactamente lo que se hace (tareas difíciles de explicar con palabras).
* Análisis de disposiciones físicas, tránsito, iluminación, ruido.
* Económica en comparación con otras técnicas.

Desventajas

* La gente se siente incómoda siendo observada.
* Algunas actividades del sistema pueden ser realizadas en horarios incómodos.
* Las tareas están sujetas a interrupciones.
* Tener en cuenta que la persona observada puede estar realizando las tareas de la forma “correcta” y no como lo hace habitualmente.

**Métodos interactivos:** la base es hablar con las personas en la organización y escuchar para comprender.

1. Cuestionarios.
2. Entrevistas.
3. Planeación conjunta de Requerimientos (JRP o JAD).
4. Lluvia de Ideas - Brainstorming.

1. Muchos analistas de sistemas objetan que las respuestas carecen de información confiable y útil. Sin embargo, los cuestionarios pueden ser un medio efectivo para recopilar los hechos, y muchas de estas críticas pueden atribuirse al uso inapropiado o poco efectivo de los cuestionarios.

Ventajas:

* Respuesta rápida
* Económicos
* Anónimos
* Estructurados de fácil análisis

Desventajas:

* Número bajo de respuestas
* No responde a todas las preguntas
* Preguntas rígidas
* No se puede realizar el análisis corporal
* No se pueden aclarar respuestas incompletas
* Difíciles de preparar

#### Tipos de Preguntas

**Abiertas:** Son las que dejan abiertas todas las posibles opciones de respuesta.

*«Describa los problemas que experimenta en la actualidad con los informes de las salidas»,  
«En su opinión, ¿qué tan útiles son los manuales de usuario para la aplicación de contabilidad del sistema actual?»*

**Cerradas:** Limitan o cierran las opciones de respuestas disponibles. Entran en juego las preguntas de opción múltiple, de calificación y de jerarquización.

*«¿Es útil el reporte que utiliza actualmente?» SI NO*

2. Las entrevistas personales consisten en preguntar los requerimientos a través de una interacción directa cara a cara. Las entrevistas pueden usarse para alcanzar alguno o todos los objetivos siguientes: indagar hechos, verificar hechos, aclarar hechos, generar entusiasmo, hacer que se involucre el usuario final, identificar los requerimientos, y solicitar ideas y opiniones.

El analista de sistemas es el entrevistador, responsable de la organización y la conducción de la entrevista. El usuario del sistema o el propietario del sistema es el entrevistado, a quien se le pide que responda a una serie de preguntas. Puede haber uno o más entrevistadores, entrevistados o ambos.

Ventajas:

* El entrevistado se siente incluido en el proyecto
* Es posible obtener una retroalimentación del encuestado
* Es posible adaptar las preguntas de acuerdo al entrevistado
* Información no verbal observando las acciones y expresiones del entrevistado

Desventajas:

* Costosas
* Tiempo y recursos humanos
* Las entrevistas dependen en gran parte de las habilidades del entrevistador
* No aplicable a distancia

#### TIPOS Y TÉCNICAS DE LA ENTREVISTA

Las **entrevistas** **no estructuradas** se caracterizan porque contienen preguntas generales que permiten al entrevistado dirigir la conversación. Frecuentemente, este tipo de entrevista se sale del camino, y el analista debe estar preparado para redirigir la entrevista hacia el principal objetivo o tema. Por esta razón, las entrevistas no estructuradas generalmente no funcionan bien para el análisis y el diseño de sistemas. Tienden a que se formulen preguntas de *respuesta abierta*. Este tipo de preguntas da a los entrevistados una gran libertad en sus respuestas.

Las **entrevistas estructuradas** consisten en que el entrevistador formula preguntas específicas diseñadas para obtener información específica de parte del entrevistado. Dependiendo de las respuestas del entrevistado, el entrevistador dirigirá preguntas adicionales para obtener una aclaración o una ampliación. Algunas de estas preguntas pueden ser planeadas y otras, espontáneas. Tienden a que se formulen más preguntas de *respuesta cerrada* que están diseñadas para suscitar respuestas cortas y directas de parte del entrevistado.

Es imprescindible realizar *sondeos*. o. El sondeo más sólido es el más simple: la pregunta “¿Por qué?”. Otros sondeos son: “¿Me puede dar un ejemplo de un momento en el que el sistema no le haya parecido confiable?” y “¿Podría explicarme eso?”. El propósito del sondeo es ir más allá de la respuesta inicial para obtener más detalles significativos, aclarar la información, y ampliar el punto del entrevistado. Los sondeos pueden ser preguntas abiertas o cerradas.

#### conducción de una entrevista

**LEER EL MATERIAL SOBRE LOS ANTECEDENTES:** Lea y comprenda todo lo que pueda sobre los antecedentes de los entrevistados y la organización. En esta fase de investigación ponga especial atención al lenguaje utilizado por los integrantes corporativos para describirse a sí mismos y a la compañía. Trate de recopilar un vocabulario común que le permita expresar preguntas de forma que sus entrevistados puedan comprender inmediatamente. Otro beneficio de investigar la organización es aprovechar al máximo el tiempo invertido en las entrevistas

**ESTABLECER LOS OBJETIVOS DE LA ENTREVISTA:** Defina los objetivos de la entrevista a partir de los antecedentes investigados y de su propia experiencia.

**DECIDIR A QUIÉN ENTREVISTAR:** Incluya personas clave de todos los niveles que se vean afectados por el sistema en cierta forma. Trate de que la muestra sea representativa para indagar sobre la mayor cantidad posible de necesidades de usuario.

**PREPARAR AL ENTREVISTADO:** Para preparar a la persona que va a entrevistar, llame por teléfono o envíe un mensaje de correo electrónico con anticipación, de manera que el entrevistado esté preparado; si la entrevista es muy detallada, envíe previamente el cuestionario por correo electrónico para que el entrevistado pueda pensar en sus respuestas. Las entrevistas se deben realizar por lo general en persona y no a través de correo electrónico. Deben durar de 45 minutos a 1 hora como máximo.

**DECIDIR SOBRE LOS TIPOS DE PREGUNTAS Y SU ESTRUCTURA:** Redacte preguntas para cubrir las áreas clave de la HCI y el proceso de toma de decisiones que haya descubierto al momento de determinar los objetivos de la entrevista. Las técnicas de interrogación apropiadas son la base de la entrevista. Las preguntas tienen ciertas formas básicas que necesitamos conocer. Los dos tipos básicos de preguntas son abiertas y cerradas.

#### organización de la entrevista

**USO DE UNA ESTRUCTURA DE PIRÁMIDE:** La organización inductiva de las preguntas de la entrevista se puede visualizar en forma de pirámide. El entrevistador empieza con preguntas muy detalladas, a menudo cerradas. Después expande los temas al permitir preguntas abiertas y respuestas más generalizadas.

Utilice una estructura de pirámide si cree que su entrevistado necesita entrar en calor en cuanto al tema. También es conveniente usar una estructura de pirámide para las secuencias de preguntas si desea una determinación sobre el tema.

**USO DE UNA ESTRUCTURA DE EMBUDO:** En el segundo tipo de estructura, el entrevistador usa un enfoque deductivo al empezar con preguntas generalizadas y abiertas, para después reducir la cantidad de respuestas posibles mediante el uso de preguntas cerradas. El método de la estructura de embudo ofrece una manera fácil y amigable de empezar una entrevista. También es conveniente usar una secuencia de preguntas en forma de embudo cuando el entrevistado está relacionado sentimentalmente con el tema y necesita libertad para expresar esos sentimientos.

**USO DE UNA ESTRUCTURA EN FORMA DE DIAMANTE:** En esta estructura la entrevista empieza de una manera muy específica y después se examinan las cuestiones generales, para finalmente llegar a una conclusión muy particularizada. El entrevistador empieza con preguntas fáciles y cerradas que permiten al entrevistado entrar en calor; a la mitad se le pregunta lo que opina sobre temas amplios que obviamente no tienen sólo una respuesta “correcta”. Después, el entrevistador restringe incluso más las preguntas para obtener respuestas específicas, con lo cual se produce un cierre tanto para el entrevistado como para el entrevistador. La estructura de diamante combina las ventajas de los otros dos métodos, pero tiene la desventaja de que toma más tiempo que las otras dos estructuras.

3. Proceso mediante el cual se conducen reuniones de grupo altamente estructurados con el propósito de analizar problemas y definir requerimientos.

* Requiere de extenso entrenamiento
* Reduce el tiempo de exploración de requisitos
* Amplia participación de los integrantes
* Se trabaja sobre lo que se va generando

Ventajas:

* Ahorro de tiempo
* Usuarios involucrados
* Desarrollos creativos

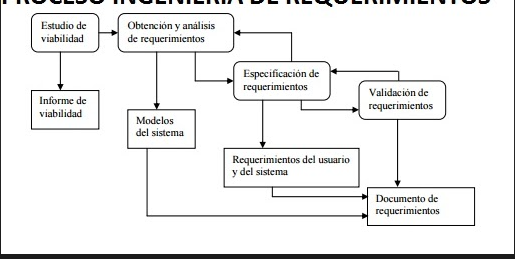
Desventajas:

* Es difícil organizar los horarios de los involucrados
* Es complejo encontrar un grupo de participantes integrados y organizados

4. Técnica para generar ideas al alentar a los participantes para que ofrezcan tantas ideas como sea posible en un corto tiempo sin ningún análisis hasta que se hayan agotado las ideas. Se promueve el desarrollo de ideas creativas para obtener soluciones. Se realizan reuniones del equipo involucrado en la resolución del problema, conducidas por un director.

* Es útil combinarlo con la toma de decisiones
* Ayuda a entender el dominio del problema
* Encara la dificultad del usuario para transmitir
* Ayuda a entender: al usuario y al analista

# **CLASE 2: Requerimientos de software**



## Estudio de viabilidad

A partir de una descripción resumida del sistema se elabora un informe que recomienda la realización o no del proceso de desarrollo. Se realiza principalmente para sistemas nuevos.

Responde a las siguientes preguntas:

* ¿El sistema contribuye a los objetivos generales de la organización? (Si no contribuye, entonces no tiene un valor real en el negocio)
* ¿El sistema se puede implementar con la tecnología actual?
* ¿El sistema se puede implementar con las restricciones de costo y tiempo?
* ¿El sistema puede integrarse a otros que existen en la organización?

Una vez que se ha recopilado la información necesaria para contestar las preguntas anteriores, se redacta un informe donde debería hacerse una recomendación sobre si debe continuar o no el desarrollo del sistema.

## Especificación de Requerimientos

Objetivos:

* **Permitir que los desarrolladores** expliquen cómo han entendido lo que el cliente pretende del sistema
* **Indicar a los diseñadores** qué funcionalidad y características va a tener el sistema resultante
* **Indicar al equipo de pruebas** qué demostraciones llevar a cabo para convencer al cliente de que el sistema que se le entrega es lo que había pedido.

Un buen requerimiento debe ser necesario, de modo que su omisión provoque una deficiencia en el sistema; conciso, para que sea fácil de leer y entender; completo, sin necesidad de ampliaciones posteriores; consistente, asegurando que no entre en contradicción con otros requerimientos; no ambiguo, de manera que tenga una única interpretación o implementación posible; y verificable, permitiendo que se pueda evaluar su cumplimiento a través de inspecciones, pruebas u otros métodos.

La especificación de requerimientos puede seguir distintos formatos, entre ellos: documento de definición de requerimientos, documento de especificación de requerimientos o documento de especificación de requerimientos de Software IEEE Std. 830-1998 (SRS).

## Validación de requerimientos

Es el proceso de certificar la corrección del modelo de requerimientos contra las intenciones del usuario. Trata de mostrar que los requerimientos definidos son los que estipula el sistema, describiendo el ambiente en el que debe operar el sistema. Es importante porque los errores en los requerimientos pueden conducir a grandes costos si se descubren más tarde.

Según el IEEE:

* **Validación:** al final del desarrollo, evaluar el software para asegurar que cumple con los requerimientos.
* **Verificación:** el software cumple los requerimientos correctamente.

Se comprende la validación de validez (para todos los usuarios), de consistencia (sin contradicciones), de completitud (todos los requerimientos), de realismo (se pueden implementar) y de verificabilidad (se puede diseñar conjunto de pruebas). Las técnicas de validación pueden ser manuales o automatizadas, informales (se deben tratar los requerimientos con tantos stakeholders como sea posible) y formales (se debe conducir al cliente, explicándole las implicaciones de cada requerimiento).

# **2.1 Técnicas de Especificación de Requerimientos de software**

* **Estáticas:** se describe el sistema a través de entidades u objetos, sus atributos y sus relaciones con otros. No describen cómo las relaciones cambian con el tiempo.
* **Dinámicas:** se describe el sistema en función de los cambios que ocurren a lo largo del tiempo. Se considera que el sistema está en un estado particular hasta que un estímulo lo obliga a cambiar su estado. Ejemplos: Historias de Usuario, Tablas de decisión, Diagramas de transición de estados, Tablas de transición de estados, Diagramas de persianas, Diagramas de transición extendidos, Redes de Petri, entre otras.

## HISTORIAS DE USUARIO

Una historia de usuario es una descripción corta y simple de un requerimiento de un sistema, que se escribe en lenguaje común del usuario y desde su perspectiva. Son utilizadas en las metodologías de desarrollo ágiles (Ejemplo: XP, SCRUM) para la especificación de requerimientos. Acompañadas de las discusiones con los usuarios y las pruebas de validación.

* **Debe ser limitada**, ésta debería poder escribirse sobre una nota adhesiva **pequeña**.
* Son **una forma rápida de administrar los requisitos** (verificables)de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos.
* Permiten **responder rápidamente a los requisitos cambiantes** (son **negociables**).
* Al momento de implementar las historias, los desarrolladores deben tener la posibilidad de **discutirlas con los clientes.**
* Generalmente se espera que la **estimación de tiempo** (estimables)de cada historia de usuario se sitúe entre unas **10 horas y un par de semanas.**
* Estimaciones **mayores a dos semanas** son indicativo de que la historia es muy compleja (épica) y debe ser **dividida en varias historias** (**independientes** unas de otras)**.** Se denomina Épica a un conjunto de Historias de usuario que se agrupan por algún denominador común.

Esquema: Como (**rol**) quiero (**algo**) para (**beneficio**).

Los **criterios de aceptación** son utilizados para expresar el resultado de las conversaciones del cliente con el desarrollador. El cliente debería ser quien las escriba más que el desarrollador. Representan el inicio de la definición del cómo. No están diseñados para ser tan detallados como una especificación de diseño tradicional. Puede añadirse un número de escenario para identificar al criterio, asociado a la historia de usuario en cuestión

Beneficios:

* Al ser muy corta, ésta representa requisitos del modelo de negocio que pueden implementarse rápidamente (días o semanas).
* Necesitan poco mantenimiento.
* Mantienen una relación cercana con el cliente.
* Permite dividir los proyectos en pequeñas entregas.
* Permite estimar fácilmente el esfuerzo de desarrollo.
* Es ideal para proyectos con requisitos volátiles o no muy claros.

Limitaciones:

* Sin criterios de aceptación pueden quedar abiertas a distintas interpretaciones haciendo difícil utilizarlas como base para un contrato.
* Se requiere un contacto permanente con el cliente durante el proyecto lo cual puede ser difícil o costoso.
* Podría resultar difícil escalar a proyectos grandes.
* Requiere desarrolladores muy competentes.

# **clase 3: casos de uso**

Proceso de modelado de las “funcionalidades” del sistema en término de los eventos que interactúan entre los usuarios y el sistema. Se centra en los usuarios del sistema, por ende, el analista puede concentrarse en cómo se va a usar el sistema y no en cómo va a construirse. La modelación de casos de uso es un enfoque que facilita el desarrollo centrado en el usuario. La modelación de casos de uso tiene sus raíces en la modelación orientada a objetos.

Beneficios:

* Proporciona una herramienta para capturar los requerimientos funcionales.
* Ayuda a descomponer el alcance del sistema en piezas más manejables.
* Proporciona un medio de comunicación con los usuarios y con otros involucrados en relación con la funcionalidad del sistema.
* Los casos de uso presentan un lenguaje común que se entiende fácilmente por los diferentes involucrados.
* Proporciona un medio para identificar, asignar, rastrear, controlar y administrar las actividades de desarrollo de sistemas, especialmente un desarrollo por incrementos e iterativo.
* Proporciona una línea de base para pruebas en términos de la definición de los planes y casos de prueba.
* Proporciona una herramienta para el seguimiento de los requerimientos.
* Proporciona un punto inicial para la identificación de los objetos o entidades de datos.
* Proporciona especificaciones funcionales para el diseño de las interfaces entre el usuario y el sistema.
* Contienen una enorme cantidad de detalle de la funcionalidad del sistema, y serán un recurso constante para validar al sistema.

Existen dos elementos primordiales cuando se realiza la modelación de casos de uso. El primero es el **diagrama de casos de uso**, que ilustra gráficamente al sistema como una colección de casos, actores (usuarios) y sus relaciones. Este diagrama comunica a un alto nivel el alcance de los eventos de negocios que el sistema debe procesar. Muestra cada una de las funciones del sistema, o eventos del negocio (dibujados en las elipses), y a los actores, o usuarios del sistema, quienes interactúan con esas funciones.

El segundo elemento, llamado la **narración del caso de uso**, describe los detalles de cada evento de negocios y especifica cómo interactúan los usuarios con el sistema durante ese evento.

## DIAGRAMA DE CASOS DE USO

### CASOS DE USO

Describen las funciones del sistema desde la perspectiva de los usuarios externos de una forma y con una terminología que ellos entienden. Los casos de uso se representan gráficamente con una **elipse horizontal con el nombre del caso** que aparece encima, debajo o dentro de la elipse.

Un caso de uso representa un objetivo individual del sistema y describe una secuencia de actividades y de interacciones del usuario para tratar de alcanzar el objetivo. Para que el CU sea considerado un requerimiento debe estar acompañado de su respectivo escenario.

### ACTORES

Un actor inicia la actividad del sistema, un caso de uso, con el propósito de terminar alguna tarea de negocios que produzca algo con valor apreciable. Un actor representa un papel desempeñado por un usuario que interactúa con el sistema y no significa que retrate a una persona o un puesto de trabajo. De hecho, un actor no tiene que ser humano, podría ser un evento temporal (evento del sistema que es activado por el tiempo).

* Actor primario de negocios: Es quien se beneficia directamente del caso de uso al recibir algo de valor (medible u observable). No necesariamente inicia el evento, pero es el principal receptor del beneficio.
* Actor primario del sistema: Es quien interactúa directamente con el sistema para iniciar o facilitar el evento de negocio. A menudo actúa en beneficio del actor primario de negocios.
* Actor externo servidor: Responde a solicitudes hechas desde el caso de uso, actuando como un proveedor de información o servicios.
* Actor externo receptor: Recibe un resultado o salida del caso de uso, aunque no sea el actor principal.

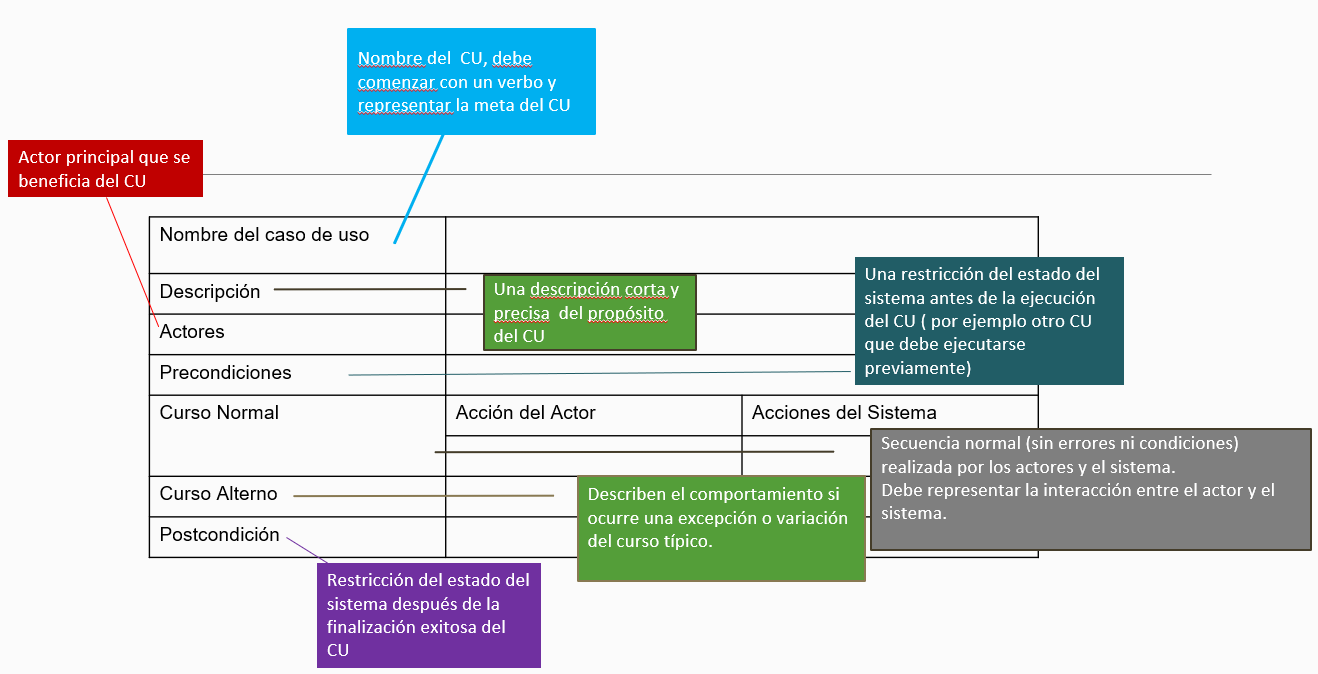
### relaciones

Una relación se ilustra como una línea entre dos símbolos en el diagrama de casos de uso. El significado de las relaciones puede diferir dependiendo de cómo se dibujen las líneas y qué tipos de símbolos conectan.

* **Asociaciones:** Existe una relación entre un actor y un caso de uso siempre que el caso describa una interacción entre éstos. Se modela como una línea continua que conecta al actor y al caso de uso. Una asociación que contiene una cabeza de flecha en el extremo que toca al caso de uso indica que el caso fue iniciado por el actor en el otro extremo de la línea. Las asociaciones sin cabezas de flecha indican una interacción entre el caso de uso y un actor externo servidor o receptor.
* **Extensión:** Un CU extiende la funcionalidad de otro CU. La relación entre el caso de uso de extensión y el que se está extendiendo se llama una relación de extensión. Un caso de uso puede tener muchas relaciones de extensión, pero un caso de uso de extensión puede ser invocado solamente por el caso que se esté extendiendo.
* **Usos (o inclusión):** Lo mejor es extraer estos pasos comunes para formar un caso de uso separado que sea propio llamado un caso de uso resumen. Un caso de uso resumen representa una forma de “reuso” y es una herramienta excelente para reducir la redundancia entre los casos de uso. Un caso de uso resumen está disponible como referencia (o uso) para cualquier otro caso de uso que requiera su funcionalidad. La relación entre el caso de uso resumen y el caso de uso que lo usa se llama una relación de uso.
* **Dependencia:** Relación entre casos de uso que indica que un caso de uso no puede realizarse hasta que se haya realizado el otro caso de uso. La relación de dependencia se ilustra con una línea con cabeza de flecha que comienza en un caso de uso y que apunta al caso de uso del cual depende.
* **Herencia**: Relación entre actores donde un actor hereda las funcionalidades de uno o varios actores. Se representa con una flecha hacia el actor del que se esté heredando.

## ESCENARIOS

En el escenario se describen: la interacción del escenario y los eventos alternativos.



### PROCESO DE MODELADO

* **Identificar a los actores. ¿Dónde buscar actores potenciales?**
  + Un diagrama de contexto que identifique el alcance o las fronteras del sistema.
  + La documentación del sistema y los manuales del usuario existentes.
  + Minutas de las juntas y los talleres del proyecto.
  + Documentos de los requerimientos, carta constitutiva del proyecto, o declaración del trabajo existente.
  + *¿Quién o qué proporciona las entradas al sistema? ¿Quién o qué recibe las salidas del sistema? ¿Se requieren interfaces con otros sistemas? ¿Existen eventos que son originados automáticamente en un instante predeterminado? ¿Quién mantendrá la información en el sistema?*
* **Identificar los CU para los requerimientos.** 
  + ¿Cuáles son las principales tareas del actor?
  + ¿Qué información necesita el actor del sistema?
  + ¿Qué información proporciona el actor al sistema?
  + ¿Necesita el sistema informar al actor de eventos o cambios ocurridos?
  + ¿Necesita el actor informar al sistema de eventos o cambios ocurridos?
* **Construir el diagrama del modelo de casos de uso.**
  + Una vez que los casos de uso y los actores han sido identificados, puede usarse un diagrama de modelo de casos de uso para ilustrar gráficamente el alcance y las fronteras del sistema.
* **Realizar los escenarios.**

# **CLASE 4: Diagrama de transición y estados**

Herramienta que se usa para ilustrar la secuencia y variación de pantallas que pueden ocurrir durante una sesión de usuario.

## Máquinas de Estado Finito

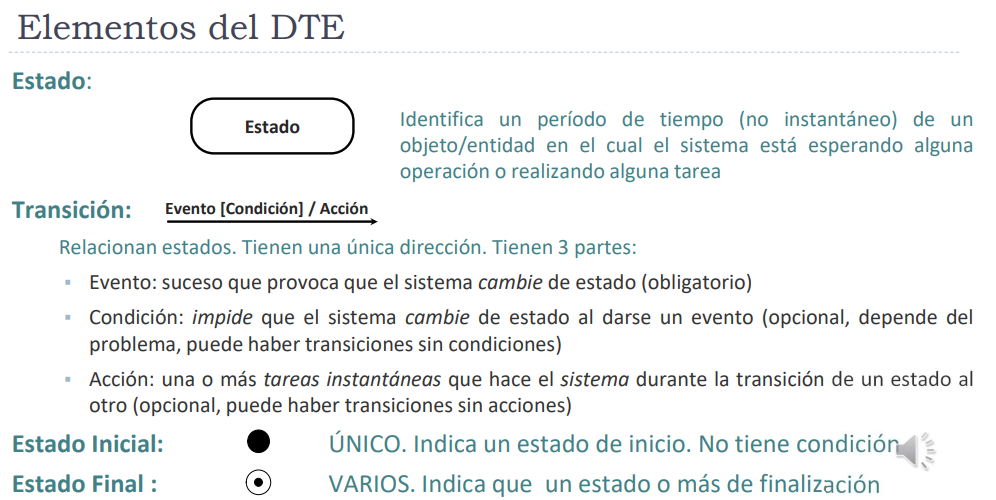
Describe al sistema como un conjunto de estados donde el sistema reacciona a ciertos eventos posibles (externos o internos).

*f(Si, Cj) = Sk*: Al estar en el estado Si, la ocurrencia de la condición Cj hace que el sistema cambie al estado Sk.

Formalmente, un autómata finito (AF) puede ser descrito como una 5-tupla (S,Σ,T,s,A) donde:

* Σ es un alfabeto;
* S un conjunto de estados;
* T es la función de transición;
* s es el estado inicial;
* A es un conjunto de estados de aceptación o finales.

## ELEMENTOS DEL DTE



Nombre de los estados: Verbos en gerundio (ando-endo: ingresando, esperando)

Eventos: Manifiestan la ocurrencia de un estímulo que conlleva la salida del estado (no puede ser verbo en infinitivo, no confundir con una acción). Tiene forma impersonal.

Condición: Condición Lógica que puede evaluar el sistema. Se escribe entre corchetes [condicion].

Acción: Verbo en infinitivo con sustantivo en función del sistema. Ej: entregar billetes (no confundir con acciones del usuario. Ej: “presionar tecla” no es algo que hace el sistema)

## construcción del dte

Es importante tener en cuenta que el modelado se realiza desde el punto de vista del sistema y NO desde el punto de vista del usuario.

1. Identificar los estados
2. Si hay un estado complejo se puede explotar
3. Identificar el estado inicial
4. Desde el estado inicial, se identifican los cambios de estado con flechas
5. Se analizan las condiciones y las acciones para pasar de un estado a otro
6. Se verifica la consistencia:

* Se han definido todos los estados
* Se pueden alcanzar todos los estados
* Se pueden salir de todos los estados
* En cada estado, el sistema responde a todas las condiciones posibles (normales y anormales)

# **clase 5: redes de petri**

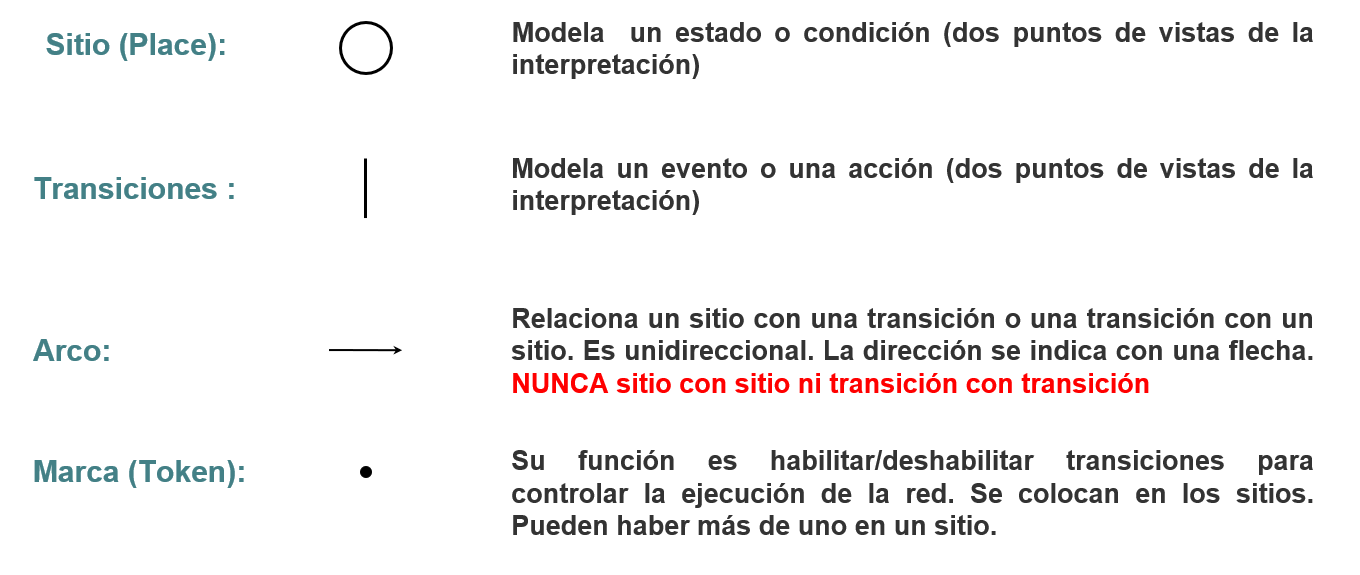
Una Red de Petri es un multigrafo (de un nodo puede partir más de un arco), bipartito, dirigido que permite modelar sistemas dinámicos y concurrentes mediante una representación gráfica de eventos discretos. Las tareas concurrentes deben estar sincronizadas para permitir la comunicación entre ellas (pueden operar a distintas velocidades, deben prevenir la modificación de datos compartidos o condiciones de bloqueo). Pueden realizarse varias tareas en paralelo, pero son ejecutados en un orden impredecible.

Una estructura de Red de Petri es una 4-upla C=(P, T, I, O):

* Lugares: P={P1,P2,...Pm}
* Transiciones: T={T1,T2,...Tn}
* Función de entrada I: I:T->P
* Función de salida O: O:T->P

## COMPONENTES DE UNA RED DE PETRI

La red cuenta con 4 componentes principales: sitios, transiciones, arcos y tokens.



## FUNCIONAMIENTO DE UNA RED DE PETRI

* La ejecución de una Red de Petri se realiza disparando transiciones habilitadas.
* La ejecución es controlada por el número y distribución de los tokens.
* Disparar una transición habilitada implica remover tokens de los lugares de entrada y distribuir tokens en los lugares de salida (teniendo en cuenta la cantidad de arcos que llegan y la cantidad de arcos que salen de la transición).
* La ejecución es asincrónica, no determinística y se acepta que el disparo de una transición es instantáneo.
* Las condiciones para habilitar transiciones dependen de los tokens en los lugares de entrada (V: con token; F: sin token), y las transiciones satisfacen las postcondiciones al ocurrir.

**Transición habilitada:** una transición se encuentra habilitada cuando al menos hay un token por cada arco que llega a la transición.

**Propagación de tokens:** Cuando una transición se encuentra habilitada, en un instante de tiempo i absorberá tantos tokens como arcos llegan y producirá tantos tokens como arcos salen en el instante i+Δ.

## Convenciones

Convención de inicio

* Para indicar que se pueden generar una cantidad ilimitada de tokens se utiliza una transición sin entradas (transición fuente). Puede haber más de una en la Red de Petri.

No bloquear la red

* Toda transición debe tener oportunidad de ser habilitada alguna vez.

Nombres obligatorios y expresados en el diagrama

* Todos los estados y transiciones deben tener nombres distintos. Además, las transiciones pueden llamarse según la etapa anterior (porque termina) o siguiente (porque empieza).

Convención de fin

* Una transición sin lugares de salida elimina tokens de la Red de Petri (transición final o de salida). Puede haber más de una.

## sincronización

Se refiere a la coordinación de la ejecución de varios procesos para que colaboren en la resolución de un problema común, asegurando que se comparten recursos e información de forma segura y controlada.

* Exclusión mutua: asegura que un recurso compartido (en este caso el token) sea accedido por un único proceso a la vez. Es una forma de evitar conflictos cuando múltiples procesos intentan modificar el mismo recurso al mismo tiempo.
* Productor - Consumidor: los productores generan tokens y los consumidores se encargan de tomar los mismos y utilizarlos.
* Condición de bloqueo: ocurre cuando un proceso no puede continuar porque está esperando que se libere un recurso necesario, que está siendo utilizado por otro proceso. Si esta espera se convierte en un ciclo cerrado entre varios procesos, puede llevar a un deadlock (interbloqueo).
* Alternancia: permite que dos procesos accedan a un recurso compartido de forma ordenada, turnándose para evitar conflictos.

# **clase 6: tablas de decisión – análisis estructurado**

## tablas de decisión

Es una herramienta que permite presentar de forma concisa las reglas lógicas que hay que utilizar para decidir acciones a ejecutar en función de las condiciones y la lógica de decisión de un problema específico.

Describe el sistema como un conjunto de: Posibles **CONDICIONES** satisfechas por el sistema en un momento dado **REGLAS** para reaccionar ante los estímulos que ocurren cuando se reúnen determinados conjuntos de condiciones y **ACCIONES** a ser tomadas como un resultado.

Puntos a tener cuenta:

* **Cantidad de reglas:** 2N, donde N es la cantidad de condiciones encontradas.
* Cada regla posee una combinación de valores de verdad (V o F).
* No pueden quedar reglas sin acciones marcadas.
* Las condiciones que se asumen como precondiciones desde el enunciado no van en la tabla.
* Las reducciones se realizan entre 2 reglas que comparten las mismas acciones y difieren en 1 solo valor de sus condiciones.
* Las reglas inconsistentes deben eliminarse.

## construcción de una tabla de decisión

Para construir tablas de decisión, el analista necesita determinar el tamaño máximo de la tabla; eliminar cualquier situación imposible, inconsistencia o redundancia, y simplificar la tabla lo más que pueda.

1. Determinar el número de condiciones que pueden afectar la decisión. Combinar filas que se traslapan, como las condiciones mutuamente excluyentes.
2. Determinar el número de acciones posibles que se pueden realizar.
3. Determinar el número de alternativas de condiciones para cada condición. En la forma más simple de la tabla de decisión, habría dos alternativas (V o F) para cada condición.
4. Calcular el número máximo de columnas en la tabla de decisión, para lo cual hay que multiplicar el número de alternativas para cada condición.
5. Completar la tabla según las condiciones.
6. Revisar cualquier situación imposible, contradicción o redundancia en la tabla.
   * **Especificaciones completas**: Aquellas que determinan acciones (una o varias) para todas las reglas posibles.
   * **Especificaciones redundantes**: Aquellas que marcan para reglas que determinan las mismas condiciones acciones iguales. Se expresa con un guión (-): significa que la condición 2 puede ser S o N, y que aun así se realizará la acción.
   * **Especificaciones contradictorias**: Aquellas que especifican para reglas que determinan las mismas condiciones acciones distintas.

## análisis estructurado

La técnica de análisis estructurado permite lograr una representación gráfica que permite lograr una comprensión más profunda del sistema a construir y comunicar a los usuarios lo comprendido. Hace énfasis en el procesamiento o la transformación de datos conforme estos pasan por distintos procesos.

### Diagrama de Flujo de Datos (DFD)

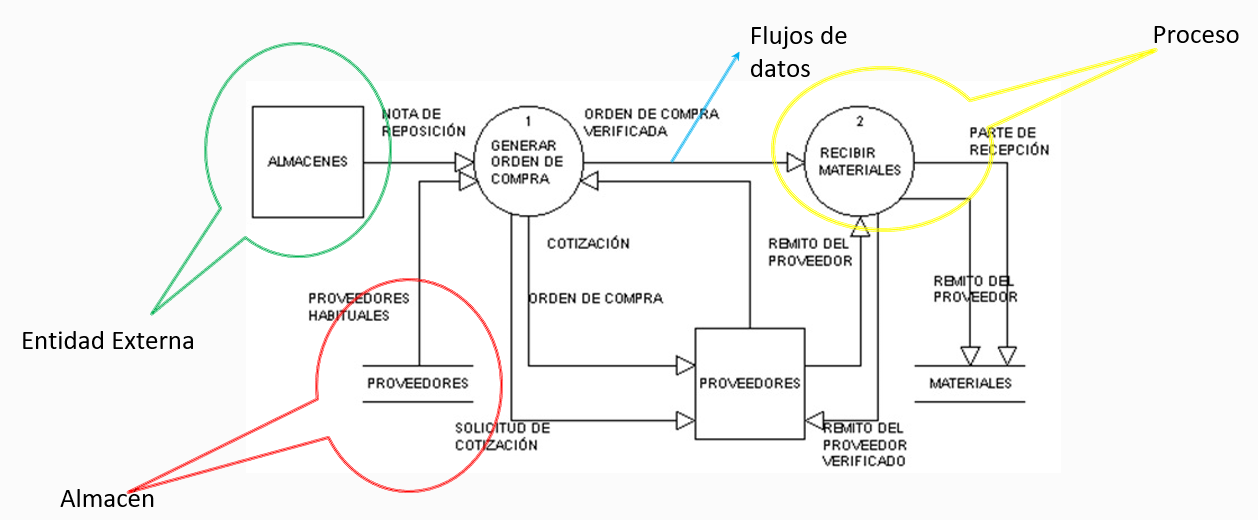
Es una herramienta que permite visualizar un sistema como una red de procesos funcionales, conectados entre sí por “conductos” y almacenamientos de datos. Representa la transformación de entradas a salidas y es también llamado diagrama de burbujas.

Se utiliza un rectángulo para representar una **entidad externa**, esto es, un elemento del sistema (por ejemplo, un elemento hardware, una persona, otro programa) u otro sistema que produce información para ser transformada por el software, o recibe información producida por el software.

Un círculo (también llamado burbuja) representa un **proceso** o transformación que es aplicado a los datos (o al control) y los modifica.

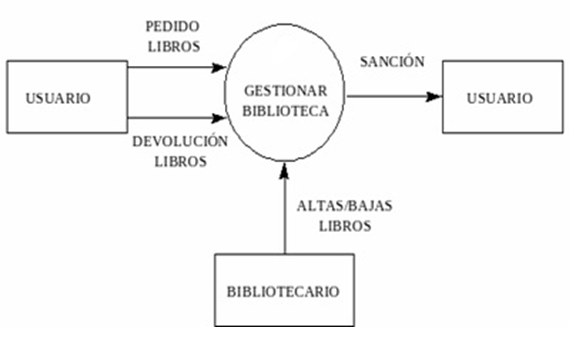
Una flecha representa un “conducto” para uno o más **elementos de datos** (objetos de dato).

Un rectángulo abierto (lado izquierdo y derecho) que representa un **almacén de datos**.



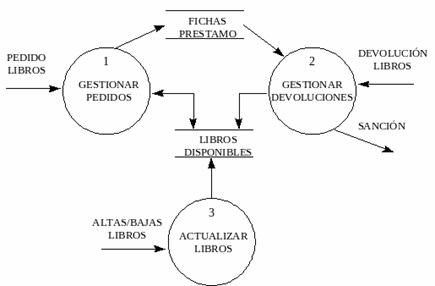
#### diagrama de contexto

Se muestra un panorama global que muestre las entradas básicas y las salidas. Es el nivel más alto en un DFD y contiene un solo proceso que representa a todo el sistema.



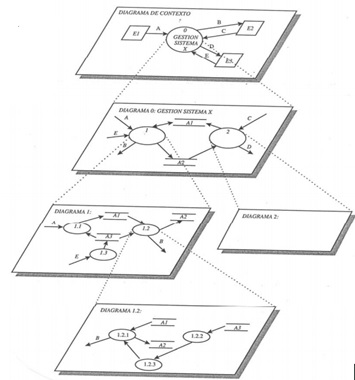
#### NIVEL 0

Es la ampliación del Diagrama de contexto. Las entradas y salidas del Diagrama de contexto permanecen, sin embargo, se amplía para incluir hasta 9 procesos (como máximo) y mostrar los almacenes de datos y nuevos flujos.



#### NIVELACIÓN DE UN DFD

Cada proceso se puede a su vez ampliar para crear un diagrama hijo más detallado. Las entradas y salidas del proceso padre permanecen, sin embargo, pueden aparecer nuevos almacenes de datos y nuevos flujos.



# **clase 7: modelos de proceso**

Un proceso es un conjunto ordenado de tareas que nos marcan diferentes formas de desarrollo de software.

## proceso de software

Un proceso de software es un conjunto de actividades y resultados asociados que producen un producto de software. Está entre las necesidades y requisitos del cliente, y el producto final.

Actividades fundamentales de los procesos de software:

* **Especificación:** consiste en el proceso de comprender y definir qué servicios se requieren del sistema, así como la identificación de las restricciones sobre la operación y desarrollo del sistema. También llamada ingeniería de requerimientos.
* **Desarrollo:** corresponde al proceso de convertir una especificación en un sistema ejecutable. Incluye los procesos de diseño y programación del software.
* **Validación:** verificar que el sistema cumpla con sus especificaciones y con las expectativas del cliente.
* **Evolución:** como se lleva adelante el mantenimiento del software una vez puesto en producción.

## modelo de proceso de software

Es una representación simplificada que nos presenta una visión de ese proceso de software que queremos llevar adelante. Estos modelos pueden incluir actividades que son partes de los procesos y productos de software, y el papel de las personas involucradas.

*Marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso (norma ISO 12207-1) [ISO/IEC, 1995]*

Características:

* Establece todas las actividades.
* Utiliza recursos, está sujeto a restricciones y genera productos intermedios y finales.
* Puede estar compuesto por subprocesos.
* Cada actividad tiene entradas y salidas definidas.
* Las actividades se organizan en una secuencia.
* Existen principios que orientan sobre las metas de cada actividad.
* Las restricciones pueden aplicarse a una actividad, recurso o producto.

## TÉRMINOS

* Ciclo de vida: proceso que implica la construcción de un producto.
* Ciclo de vida del software: describe la vida del producto de software desde su concepción hasta su implementación, entrega, utilización y mantenimiento.
* Modelos de proceso de software: representación abstracta de un proceso del software.
  + Modelo de proceso, paradigma de software y ciclo de vida del software son términos equivalentes.

**Modelos prescriptivos:** prescriben un conjunto de elementos del proceso. Cada modelo de proceso prescribe también un flujo de trabajo, es decir, de qué forma los elementos del proceso se interrelacionan entre sí.

**Modelos descriptivos:** describen la forma en que se realizan los procesos en la realidad, pero no hay una prescripción.

Ambos modelos deberían ser iguales (tanto lo que prescriben como lo que describen).

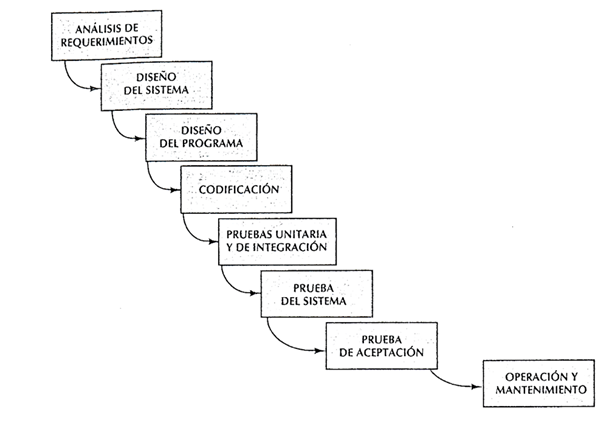
## TIPOS

* Modelos tradicionales: formados por un conjunto de fases o actividades en las que no tienen en cuenta la naturaleza evolutiva del software. Por ejemplo: en cascada, modelo en V, basado en prototipos.
* Modelos evolutivos: son los modelos que se adaptan a la evolución que sufren los requisitos del sistema en función del tiempo. Por ejemplo: en espiral, evolutivo, incremental.
* Procesos ágiles: scrum, xp, etc.

### modelo en cascada

Éste toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y, luego, los representa como fases separadas del proceso, tal como especificación de requerimientos, diseño de software, implementación, pruebas, etcétera.

Las etapas se representan cayendo en cascada. Cada etapa de desarrollo se debe completar antes de que comience la siguiente.



Etapas:

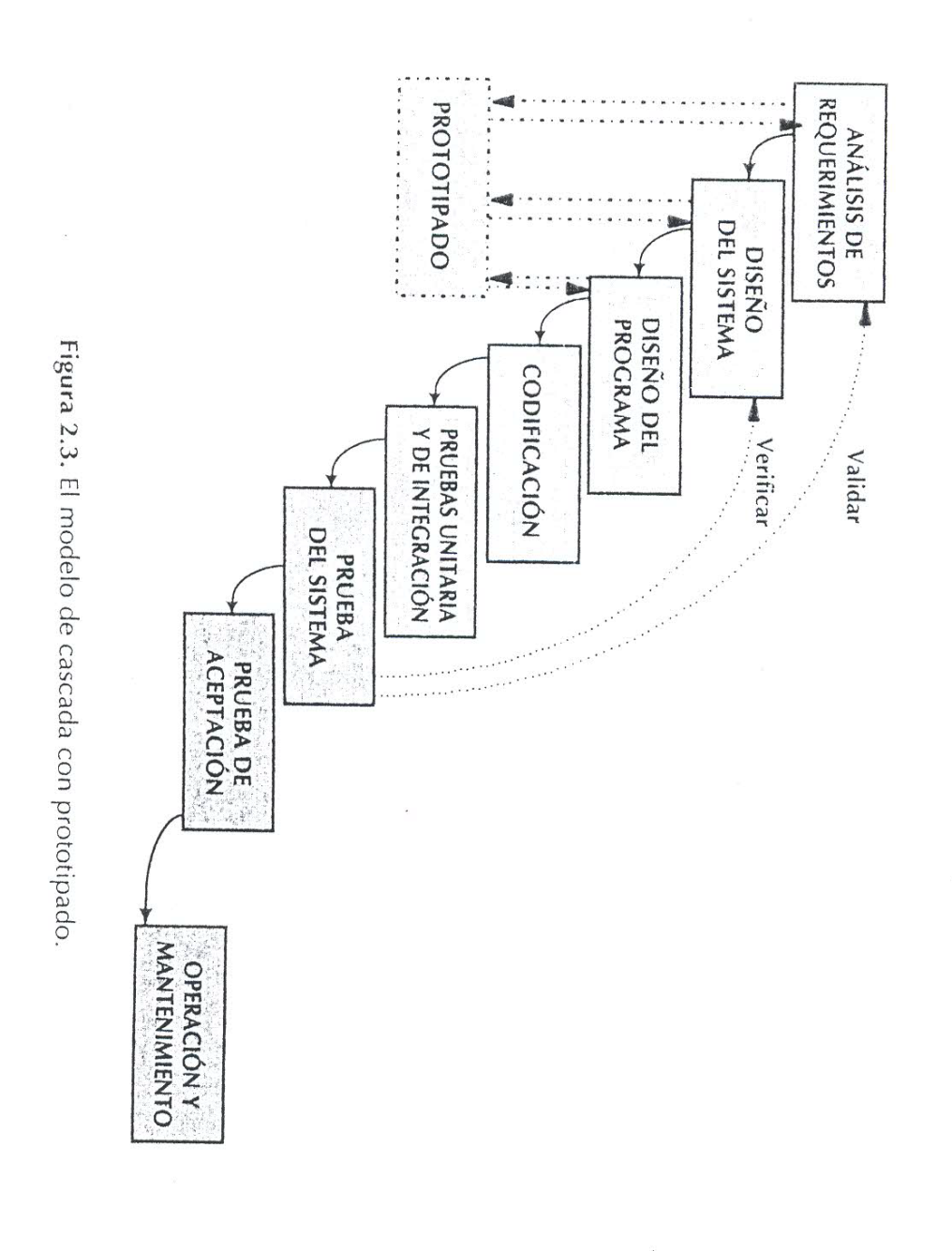
1. **Análisis y definición de requerimientos:** Los servicios, las restricciones y las metas del sistema se establecen mediante consulta a los usuarios del sistema. Luego, se definen con detalle y sirven como una especificación del sistema.
2. **Diseño del sistema y del software:** El proceso de diseño de sistemas asigna los requerimientos, para sistemas de hardware o de software, al establecer una arquitectura de sistema global.
3. **Implementación y prueba de unidad:** Durante esta etapa, el diseño de software se realiza como un conjunto de programas o unidades del programa. La prueba de unidad consiste en verificar que cada unidad cumpla con su especificación.
4. **Integración y prueba de sistema:** Las unidades del programa o los programas individuales se integran y prueban como un sistema completo para asegurarse de que se cumplan los requerimientos de software. Después de probarlo, se libera el sistema de software al cliente.
5. **Operación y mantenimiento:** Por lo general (aunque no necesariamente), ésta es la fase más larga del ciclo de vida, donde el sistema se instala y se pone en práctica. El mantenimiento incluye corregir los errores que no se detectaron en etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema e incrementar los servicios del sistema conforme se descubren nuevos requerimientos.

El problema de este modelo es que no se adapta a la realidad del desarrollo de software. No responde bien a los cambios y a la detección de errores. Debido a los costos de producción y aprobación de documentos, las iteraciones suelen ser onerosas e implicar un rediseño significativo. Los problemas se dejan para una resolución posterior, se ignoran o se programan.

En principio, el modelo en cascada sólo debe usarse cuando los requerimientos se entiendan bien y sea improbable el cambio radical durante el desarrollo del sistema. Útil para diagramar lo que se necesita hacer. Su simplicidad hace que sea fácil explicarlo a los clientes.

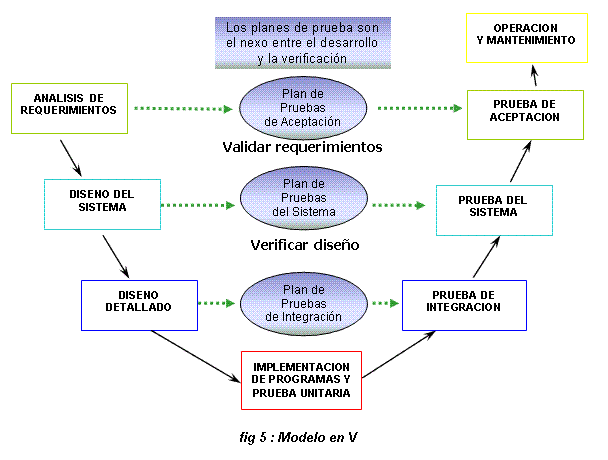
#### modelo en cascada con prototipo

Se incorporan etapas tempranas de prototipado que nos permiten ir y volver entre los pasos, sobre todo en la etapa de especificación de requerimientos.



### modelo en v

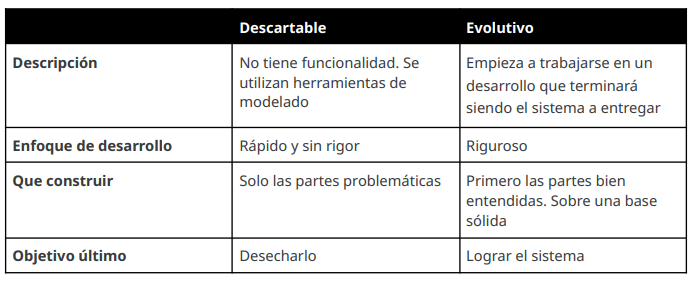
Relaciona las actividades de prueba con las de análisis y diseño. La vinculación entre el lado derecho e izquierdo implica que, si se encuentran problemas durante la verificación y la validación, entonces el lado izquierdo puede ser ejecutado nuevamente para solucionar el problema.



### Modelo de prototipos

Un prototipo es un producto parcialmente desarrollado que permite a clientes y desarrolladores examinar algunos aspectos del sistema propuesto y decidan si este es adecuado para el producto terminado.

Existen dos tipos:



Proyectos candidatos:

* Usuarios que no examinarán los modelos abstractos
* Usuarios que no determinarán sus requerimientos inicialmente
* Sistemas con énfasis en los formatos de E/S más que en los detalles algorítmicos
* Sistemas en los que haya que explorar aspectos técnicos
* Si el usuario tiene dificultad al tratar con los modelos gráficos para modelar los requerimientos y el comportamiento
* Si se enfatiza el aspecto de la interfaz humana

### Modelo de desarrollo por fases

El sistema se desarrolla de manera tal que puede ser entregado en piezas. Esto implica que existen **dos sistemas funcionando** en paralelo: el **sistema operacional** y el **sistema en desarrollo**.

figura3

Puede ser de dos tipos:

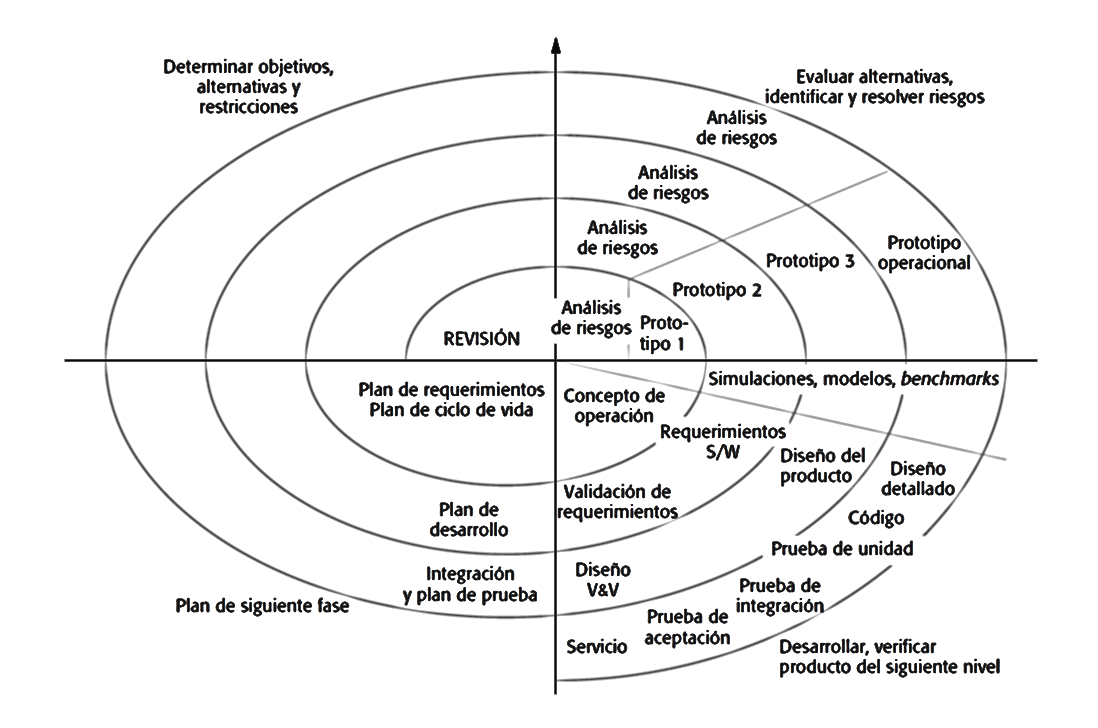
* **Incremental:** el sistema es particionado en subsistemas de acuerdo con su funcionalidad, donde cada entrega agrega un subsistema e incorpora algunas de las funciones que necesita el cliente. El desarrollo incremental refleja la forma en que se resuelven problemas. Resulta más barato y fácil realizar cambios en el software conforme éste se diseña.
* **Iterativo:** entrega un sistema completo desde el principio y luego aumenta la funcionalidad de cada subsistema con las nuevas versiones.

### Modelo en espiral (Boehm)

* Combina las actividades de desarrollo con la gestión del riesgo.
* Trata de mejorar los ciclos de vida clásicos y prototipos.
* Incorpora objetivos de calidad.
* Elimina errores y alternativas no atractivas al comienzo.
* Permite iteraciones, vuelta atrás y finalizaciones rápidas.
* Una restricción es que cada ciclo se completa con una revisión que incluye todo el ciclo anterior y el plan para el siguiente.

Cada ciclo de la espiral se divide en cuatro sectores:

1. **Establecimiento de objetivos:** se identifican restricciones, se traza un plan de gestión y se identifican riesgos.
2. **Valoración y reducción del riesgo:** se analiza cada riesgo identificado y se determinan acciones.
3. **Desarrollo y validación:** se determina el modelo de desarrollo.
4. **Planeación:** el proyecto se revisa y se toman decisiones para la siguiente fase.



# **CLASE 8: METODOLOGÍAS ÁGILES**

El éxito de un desarrollo está dado por la metodología empleada. Generalmente, esta metodología lleva asociado un marcado énfasis en el control del proceso; definiendo roles, actividades, herramientas y documentación detallada.

Este enfoque no resulta muy adecuado para proyectos actuales, donde el entorno del sistema es muy cambiante y se exige una reducción de tiempo. En este contexto, las metodologías ágiles emergen como una posible solución. Las metodologías ágiles tienen un enfoque iterativo e incremental del desarrollo de software.

Una Metodología Ágil es aquella en la que *“se da prioridad a las tareas que dan resultados directos y que reducen la burocracia tanto como sea posible”* [Fowler], adaptándose además rápidamente al cambio de los proyectos.

## valores

* **Individuos e interacciones:** Enfatiza la importancia de la comunicación personal y el trabajo en equipo más que procesos y herramientas.
* **Software operante:** Prioriza la entrega de software funcional sobre la documentación extensa.
* **Colaboración con el cliente:** Se centra en asociarse con los clientes en lugar de negociar contratos formales.
* **Respuesta al cambio:** Fomenta la adaptabilidad en lugar de la estricta rigurosa planificación.

## PRINCIPIOS

1. Nuestra mayor prioridad es satisfacer al cliente a través de fáciles y continuas entregas de software valuable.
2. Los cambios de requerimientos son bienvenidos, aún tardíos, en el desarrollo. Los procesos ágiles capturan los cambios para que el cliente obtenga ventajas competitivas.
3. Entregas frecuentes de software, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre una entrega y la siguiente.
4. Usuarios y desarrolladores deben trabajar juntos durante todo el proyecto.
5. Construir proyectos alrededor de motivaciones individuales.
6. Darles el ambiente y el soporte que ellos necesitan y confiar el trabajo dado. El diálogo cara a cara es el método más eficiente y efectivo de intercambiar información entre el equipo de desarrolladores.
7. El software que funciona es la medida clave de progreso.
8. Los procesos ágiles promueven un desarrollo sostenible. Los stakeholders, desarrolladores y usuarios deberían ser capaces de mantener un paso constante indefinidamente.
9. Atención continua a la excelencia técnica y buen diseño incrementa la agilidad.
10. Simplicidad (el arte de maximizar la cantidad de trabajo no dado) es esencial.
11. Las mejores arquitecturas, requerimientos y diseños surgen de la propia organización de los equipos.
12. A intervalos regulares, el equipo reflexiona sobre cómo volverse más efectivo, entonces afina y ajusta su comportamiento en consecuencia.

## DESVENTAJAS

* **Aunque es atractiva la idea de involucrar al cliente en el proceso de desarrollo,** los representantes del cliente están sujetos a otras presiones, y no intervienen por completo en el desarrollo del software.
* **Priorizar los cambios podría ser difícil,** sobre todo en sistemas donde existen muchos participantes. Cada uno por lo general ofrece diversas prioridades a diferentes cambios.
* **Mantener la simplicidad requiere trabajo adicional.** Bajo la presión de fechas de entrega, es posible que los miembros del equipo carezcan de tiempo para realizar las simplificaciones deseables al sistema.
* **Muchas organizaciones, especialmente las grandes compañías, pasan años cambiando su cultura, de tal modo que los procesos se definan y continúen**. Para ellas, resulta difícil moverse hacia un modelo de trabajo donde los procesos sean informales y estén definidos por equipos de desarrollo.
* Como se minimiza la documentación, es **más complejo reglamentar el documento de requerimientos** del software.

## principales metodologías ágiles

### XP - Extreme Programming

Se basa en los valores de sencillez, comunicación, retroalimentación, coraje y respeto. Su acción consiste en llevar al equipo a prácticas simples, con suficiente información para ver dónde están y para ajustar las prácticas a su situación particular.

* Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
* Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión.
* Programación en parejas.
* Frecuente integración del equipo de programación con el cliente o usuario.
* Corrección de todos los errores antes de añadir nueva funcionalidad.
* Refactorización del código.
* Propiedad del código compartida.
* Simplicidad en el código.

Las características esenciales son:

* Historias de usuarios
* Roles
* Proceso
* Prácticas

#### roles

**Programador (Programmer)**

* Responsable técnico: Toma decisiones técnicas y construye el sistema.
* Sin jerarquías: No hay distinción entre analistas, diseñadores o codificadores.
* Tareas en XP: Diseñar, programar y realizar pruebas.

**Jefe de Proyecto (Manager)**

* Organiza y guía reuniones.
* Crea condiciones adecuadas para el proyecto.

**Cliente (Customer)**

* Miembro activo del equipo: Participa en el desarrollo.
* Define qué construir y cuándo.
* Diseña y aprueba las pruebas funcionales.

**Entrenador (Coach)**

* Responsable de supervisar y garantizar que se siga el proceso.
* A medida que el equipo madura, tiende a tomar un rol más pasivo.

**Encargado de Pruebas (Tester)**

* Asiste al cliente en las pruebas funcionales.
* Verifica que las pruebas funcionales se superen.

**Rastreador (Tracker)**

* Observa el progreso sin interferir.
* Lleva un registro histórico de métricas y datos del proyecto.

#### proceso

1. **Exploración** 
   * Los clientes plantean las historias de usuario que son de interés para la primera entrega del producto.
   * El equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.
   * Se construye un prototipo.
2. **Planificación** 
   * El cliente establece la prioridad de cada historia de usuario.
   * Los programadores realizan una estimación del esfuerzo.
   * Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente.
3. **Iteraciones** 
   * El Plan de Entrega está compuesto por iteraciones de no más de tres semanas.
   * El cliente es quien decide qué historias se implementarán en cada iteración.
   * Al final de la última iteración el sistema estará listo para entrar en producción.
4. **Producción** 
   * Esta fase requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente.
   * Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.
5. **Mantenimiento** 
   * Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones.
   * La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.
6. **Muerte** 
   * Es cuando el cliente no tiene más historias para ser incluidas en el sistema.
   * Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura.
   * La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

#### prácticas

**Testing**

* Los programadores escriben y ejecutan pruebas unitarias constantemente para garantizar que el desarrollo avance sin problemas.
* Los clientes diseñan pruebas para verificar que las funcionalidades cumplen los requisitos establecidos.

**Refactoring**

* Proceso continuo de reorganización del código para eliminar duplicación, mejorar la legibilidad, simplificar y hacer el sistema más flexible para futuros cambios.

**Programación de a Pares**

* Dos programadores trabajan juntos en una sola máquina para escribir código de producción, mejorando la calidad y fomentando el aprendizaje compartido.

**Propiedad Colectiva del Código**

* Todos los programadores pueden modificar cualquier parte del código en cualquier momento.
* Promueve la colaboración, evita dependencias críticas de una sola persona y fomenta nuevas ideas.

**Integración Continua**

* Cada fragmento de código se integra inmediatamente al sistema cuando está listo, permitiendo múltiples integraciones por día.
* Minimiza errores, mejora la comunicación y evita esfuerzos fragmentados.

**Semana de 40 horas**

* Máximo de 40 horas de trabajo por semana para evitar el agotamiento y mantener la motivación del equipo.
* Si los plazos no se pueden cumplir, es preferible renegociar el alcance o la fecha de entrega.

**Cliente en el Lugar de Desarrollo**

* El cliente debe estar disponible todo el tiempo para colaborar con el equipo, resolver dudas y priorizar tareas.

**Estándares de Codificación**

* Los programadores siguen reglas uniformes para escribir código, priorizando la comunicación y legibilidad a través del mismo.

### scrum

Scrum se define como “una manera simple de manejar problemas complejos”, proporcionando un paradigma de trabajo que soporta la innovación y permite que equipos auto-organizados entreguen resultados de alta calidad en tiempos cortos. En Scrum se realizan entregas parciales y regulares del resultado final del proyecto, priorizadas por el beneficio que aportan al receptor del proyecto.

#### PRINCIPIOS

* **Eliminar el desperdicio:** no generar artefactos, ni perder el tiempo haciendo cosas que no le suman valor al cliente.
* **Construir la calidad con el producto:** la idea es inyectar la calidad directamente en el código desde el inicio.
* **Crear conocimiento:** en la práctica no se puede tener el conocimiento antes de empezar el desarrollo.
* **Diferir las decisiones:** tomar las decisiones en el momento adecuado, esperar hasta ese momento, ya que uno tiene más información a medida que va pasando el tiempo. Si se puede esperar, mejor.
* **Entregar rápido:** debe ser una de las ventajas competitivas más importantes.
* **Respetar a las personas:** la gente trabaja mejor cuando se encuentra en un ambiente que la motive y se sienta respetada.
* **Optimizar el todo:** optimizar todo el proceso, ya que el proceso es una unidad, y para lograr tener éxito y avanzar, hay que tratarlo como tal.

#### roles

* El **Product Owner** (Propietario) conoce y marca las prioridades del proyecto o producto.
* El **Scrum Master** (Jefe) es la persona que asegura el seguimiento de la metodología guiando las reuniones y ayudando al equipo ante cualquier problema que pueda aparecer. Su responsabilidad es entre otras, la de hacer de paraguas ante las presiones externas.
* El **Scrum Team** (Equipo) son las personas responsables de implementar la funcionalidad o funcionalidades elegidas por el Product Owner.
* Los **Usuarios o Cliente**, son los beneficiarios finales del producto, y son quienes, viendo los progresos, pueden aportar ideas, sugerencias o necesidades.

#### PROCESO

Scrum es un proceso iterativo e incremental donde se busca poder atacar todos los problemas que surgen durante el desarrollo del proyecto. El nombre Scrum se debe a que, durante los Sprints, lo que serían las fases de desarrollo, se solapan, de manera que no es un proceso de cascada por cada iteración, si no que tenemos todas estas etapas juntas que se ejecutan una y otra vez, hasta que se crea suficiente.

Scrum está pensado para ser aplicado en proyectos en donde el “caos” es una constante, aquellos proyectos en los que tenemos requerimientos dinámicos, y que tenemos que implementar tecnología de punta.

1. **Product backlog:** Es la lista maestra que contiene toda la funcionalidad deseada en el producto. La característica más importante es que la funcionalidad se encuentra ordenada por un orden de prioridad.
2. **Sprint planning meeting:** Se planifica la entrega en el tiempo acordado para la duración del sprint.
3. **Sprint backlog:** Es la lista que contiene toda la funcionalidad que el equipo se comprometió a desarrollar durante un Sprint determinado.
4. **Daily scrum:** Reunión ágil, corta, de 15 a 30 minutos, donde se da visibilidad a los avances diarios
5. **Finished work.**
   * **Sprint review:** Se lleva a cabo al final del sprint y se muestra lo implementado en dicha iteración.
   * **Sprint retrosprective:** Se toma en cuenta los resultados del sprint, discutidos en el sprint review.

### Kanban

Es un **enfoque lean de desarrollo de software ágil**. La filosofía Lean hace hincapié en eliminar cualquier actividad que no agregue valor a través de la mejora continua para agilizar las operaciones. Está centrada en el cliente y en ofrecer una mayor calidad, reducir el tiempo de ciclo y reducir los costos.

A diferencia de otras metodologías, no define ningún rol ni ninguna secuencia de pasos para llevar a cabo el desarrollo del proyecto. Kanban se presenta como una herramienta de gestión que prescribe solo 3 prácticas:

* Limitar el trabajo en curso
* Visualizar el flujo de trabajo
* Medir el tiempo promedio de entrega.

Un tablero Kanban permiten visualizar las tareas a realizar, las realizadas y las pendientes, y permiten visualizar explícitamente el proceso de desarrollo. Está compuesto por una serie de columnas que representan los diversos estados que atraviesa un requerimiento durante el proceso de desarrollo. Las tarjetas se mueven de un estado a otro, mostrando la evolución, hasta que hayan sido aceptadas por el cliente.

Cada columna tendrá un límite para el trabajo en curso. Cuando una tarea es completada en una columna, esta se mueve a la siguiente. De esta forma, se crea un espacio libre en la columna actual representando capacidad de trabajo disponible. Es habitual combinar estos tableros en las prácticas de otras metodologías ágiles.

# **CLASE 9: Desarrollo de Software Basado en Modelos (MBD)**

Un **modelo** del sistema consiste en una conceptualización del dominio del problema y actúa como una especificación precisa de los requerimientos que el sistema de software debe satisfacer. (Abstracción de elementos del problema, comunicación, negociación con el usuario)

## Desarrollo de Software Dirigido por Modelos (MDD)

El adjetivo «dirigido» en MDD, a diferencia de «basado», enfatiza que este paradigma asigna a los modelos un rol central y activo: son al menos tan importantes como el código fuente. Los modelos pasan de ser entidades contemplativas (es decir, artefactos que son interpretados por los diseñadores y programadores) para convertirse en entidades productivas a partir de las cuales se deriva la implementación en forma automática.

Model Driven Development (MDD) promueve enfatizar los siguientes puntos claves:

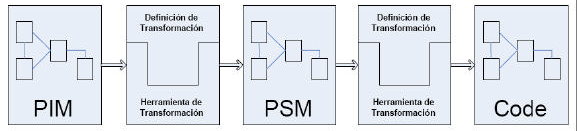
* Mayor nivel de **abstracción en la especificación**, tanto del problema a resolver como de la solución correspondiente.
* Aumento en la **confianza en la automatización** asistida por computadora para soportar el análisis, diseño y ejecución.
* Uso de **estándares industriales** como medio para facilitar las comunicaciones, la interacción entre diferentes aplicaciones y productos, y la especialización tecnológica.
* Los **modelos son los conductores primarios** en todos los aspectos del desarrollo de software.

### Modelos de MDD. (PIMs, y PSMs)

* **Platform independent model (PIM):** no contiene información sobre la plataforma y la tecnología con la que se implementará.
* **Platform specific model (PSM):** contiene información sobre la plataforma y la tecnología con la que se implementará.

### transformación

Una transformación consiste en un conjunto de reglas (especificaciones no ambiguas) de las formas en que un modelo, o parte de él, puede ser usado para crear otro modelo, o parte de él. Cada transformación incluye (al menos): un PIM, un Modelo de la Plataforma, una Transformación, y un PSM.



El patrón MDD normalmente es utilizado sucesivas veces para producir una sucesión de transformaciones.

### BENEFICIOS

* Incremento en la productividad (modelos y transformaciones).
* Adaptación a los cambios tecnológicos.
* Adaptación a los cambios de requisitos.
* Consistencia (automatización).
* Re-uso (de modelos y transformaciones).
* Mejoras en la comunicación con los usuarios y la comunicación entre los desarrolladores (los modelos permanecen actualizados).
* Captura de la experiencia (cambio de experto).
* Los modelos son productos de larga duración (resisten cambios).
* Posibilidad de demorar decisiones tecnológicas.

# **9.1 calidad**

El significado de calidad es ambiguo y muchas veces su uso depende de lo que cada uno entiende por calidad. Algunas definiciones formales son:

1. Propiedad o conjunto de propiedades inherentes a algo, que ***permiten juzgar su valor***.
2. Condición o requisito que se pone en un contrato.
3. …

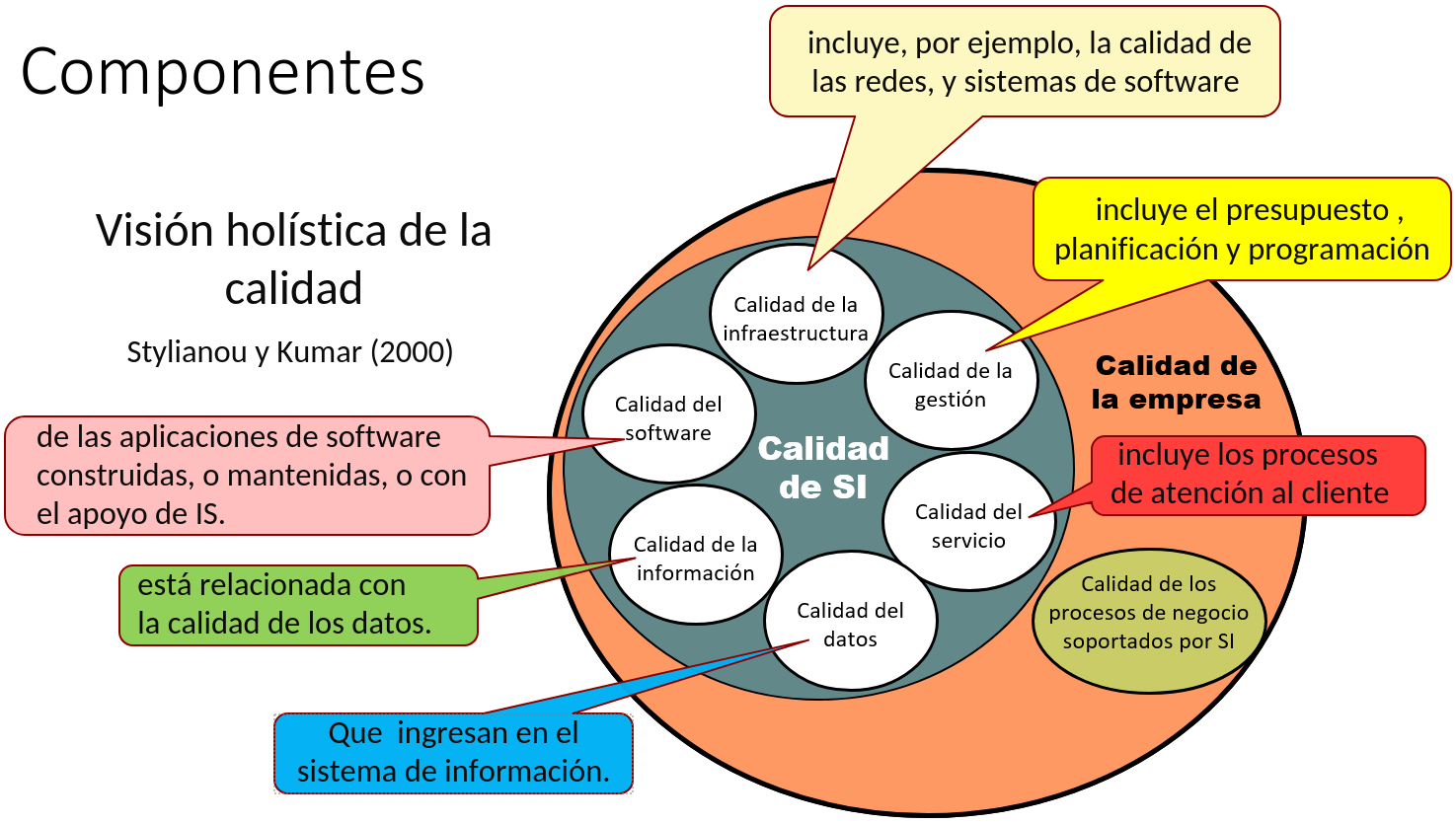
De esto se puede inferir que, la calidad, es un término totalmente subjetivo, que depende del juicio de la persona que intervenga en la evaluación. Según normas internacionales:

* **ISO 9000:** el grado en el que un conjunto de características inherentes cumple con los requisitos.
* **ISO 8402:** conjunto de propiedades o características de un producto o servicio que le confieren aptitud para satisfacer unas necesidades expresadas o implícitas.

Una **norma** es un documento establecido por consenso y aprobado por un **organismo reconocido** que proporciona una serie de reglas, directrices o características para un uso común y repetido.

## sistemas de información

Un **sistema de información** es el conjunto de personas, datos, procesos y tecnología de información que interactúan para recopilar, procesar, guardar, y proporcionar como salida la información necesaria para brindar soporte a una organización.

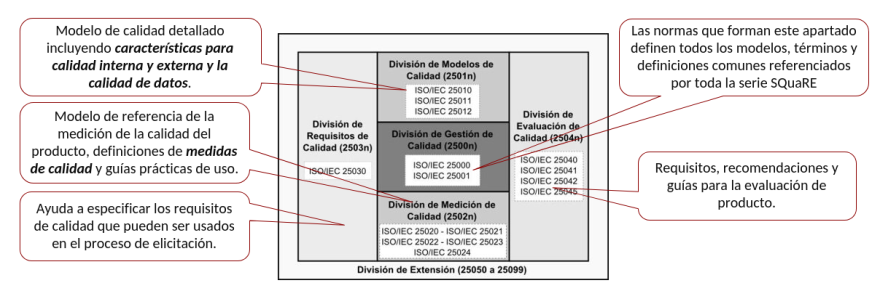


## calidad de software

Se divide en:

* **Calidad del producto obtenido:** define las propiedades que debe satisfacer el producto de software resultante.
* **Calidad del proceso de desarrollo:** define la manera de desarrollar el producto de software.

ISO/IEC 25000 SQuaRE: Software product Quality Requeriment and Evaluation:



### Mejora de la calidad de los procesos de software

* **Modelo de procesos**
  + **ISO/IEC 12207:2008:** ISO/IEC 12207 establece un modelo de procesos para el ciclo de vida del software.
* **Modelo de evaluación**
  + **ISO/IEC 15504:** es una norma internacional para establecer y mejorar la capacidad y madurez de los procesos de las organizaciones en la adquisición, desarrollo, evolución, y soporte de productos y servicios.
  + **Familia de normas ISO/IEC 33000:** proporciona un marco de trabajo coherente para la evaluación de procesos de software que sustituye las diferentes partes de la norma ISO/IEC 15504.

## CMM (1993) - CMMI (2000)

**CMM** proporciona un marco estructurado para evaluar los procesos actuales de la organización, establecer prioridades de mejora e implementar esas mejorar.

**CMMI** posee dos vistas que permiten un enfoque diferente según las necesidades de quien vaya a implementarlo:

* **Escalonado:** centra su foco en la madurez de la organización (igual que CMM)
* **Continuo:** enfoca las actividades de mejora y evaluación en la capacidad de los diferentes procesos. Presenta 6 niveles de capacidad que indican qué tan bien se desempeña la organización en un área de proceso individual.



## FAMILIA DE LOS ISO 9000

* **ISO – 9001:2015** - Quality management system – Requirements
* **IRAM – ISO 9001:2015** – Sistema de gestión de la calidad – Requisitos
* **ISO 90003:2004**
  + Basada ISO 9001:2000 (se espera una actualización para el próximo año)
  + Directrices para la interpretación en el proceso de software
  + Proporciona una guía para identificar las evidencias dentro del proceso de software para satisfacer los requisitos de la ISO 9001.

## Beneficios de trabajar con un sistema de gestión de calidad (SGC)

* ISO 9001 asegura que su negocio cumpla con los requisitos legales y del cliente.
* Aumenta el rendimiento de su organización. El SGC, ayuda a implementar procesos simplificados y mejorar la eficiencia operacional.
* Asegura la toma de decisiones y mejora la satisfacción del cliente.
* Optimiza sus operaciones para así cumplir y superar los requisitos de sus clientes.
* Mejora su rendimiento financiero.

## RESUMEN

