**INGENIERÍA DE SOFTWARE 1**

**SOFTWARE**

**¿Qué es?**

El software es mucho más que solo instrucciones, también son procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema que computación y hacer que operen de manera correcta.

El software es un elemento lógico, se desarrolla, no se fabrica como otros productos, por consiguiente, trae un mayor costo en la ingeniería que en la producción. El software no se desgasta, no sigue la misma curva de envejecimiento del hardware. Es inmune a los males que desgastan el hardware, debido a que el problema no está en el tiempo de operación, sino en los cambios.

**Tipos de productos**

Genéricos: Sistemas aislados producidos por organizadores de desarrolladoras de software y que se venden en un mercado abierto.

Personalizados: Sistemas requeridos por un cliente en particular.

**INGENIERIA DE SOFTWARE**

**¿Qué es?**

La ingeniería de software es una disciplina de ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste después de que se utiliza**.** La misma se encarga de asegurar la producción de productos correctos, utilizables, de alta calidad y costo efectivos.

**Características**

* Usa métodos sistemáticos cuantificables: La cuantificación rigorosa de recursos, procesos y productos es una precondición para optimizar productividad y calidad. La “metrificación” y el control estadístico de procesos son claves en ingeniería de software.
* Dentro de tiempos y costos estimados: Un ingeniero de software debe cumplir contratos en tiempo y costos como es normal en obras de ingeniería. Ello presupone la capacidad me medir, estimar, planificar y administrar proyectos.
* Para el “Desarrollo, operación y mantenimiento”: 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 noción de ingeniería de software fue propuesta inicialmente en 1968 en una conferencia para discutir lo que en ese entonces se llamó la “crisis del software”. Esta crisis del software fue el resultado de la introducción de las nuevas computadoras hardware basadas en circuitos integrados. Su poder hizo que las aplicaciones hasta ese entonces irrealizables fueran una propuesta factible. El software resultante fue de órdenes de magnitud más grande y más complejo que los sistemas de software previos.

**EVOLUCIÓN**

* Años 60 - Era funcional: Se estudia como explotar la tecnología para hacer frente a las necesidades funcionales de las organizaciones.
* Años 70 - Era de control: Aparece la necesidad de desarrollar software en tiempo, planeado y controlado. Se introduce el modelo de ciclo de vida en fases.
* Años 80 - Era de costos: La importancia de la productividad en el desarrollo de software se incrementa sustancialmente. Se ponen en práctica varios modelos de costos.
* Años 90 y actualidad – Era de calidad: Se intensifica la necesidad de que el producto tenga atributos que satisfagan las necesidades explicitas e implícitas del usuario: mantenibilidad, confiabilidad, eficiencia, usabilidad.

**¿Qué conocimientos debe tener un Ingeniero de software?**

* El ingeniero debe conocer las tecnologías y productos (sistemas operativos, lenguajes, base de datos, sistemas generadores de interfaces, bibliotecas de código)
* El ingeniero debe conocer técnicas de administración de proyectos (planificación, análisis de riesgo, control de calidad, seguimientos de proyectos, etc.)

**Responsabilidad profesional y ética de un Ingeniero de software**

* Respetar la confidencialidad de sus empleados y clientes.
* No falsificar el nivel de competencia y aceptar responsabilidades fuera de su capacidad.
* Conocer las leyes vigentes sobre las patentes y copyright.
* No debe utilizar sus habilidades técnicas para utilizar de forma inapropiada otras computadoras.

**Técnicas de Comunicación**

Al iniciar un proyecto se debe saber lo que el usuario quiere, como lo quiere, cuándo y por qué.

La comunicación es la base para la obtención de las necesidades del cliente. Esta Esla principal fuente de error. Al hablar de necesidades, en términos más técnicos, estamos hablando de **requerimientos.**

Un *Requerimiento* (o requisito) es una característica del sistema o una descripción de algo que el sistema es capaz de hacer con el objeto de satisfacer el propósito del sistema.

**Stakeholder**

El término stakeholder se utiliza para referirse a cualquier persona o grupo que se verá afectado por el sistema, directa o indirectamente. Entre los stakeholder se encuentran: usuarios finales, ingenieros, gerentes, etc.

**Fuentes de Información**

Son: Documentación, Stakeholders, Especificaciones de sistemas similares.

**Puntos de vistas genéricos**

Interactuadores: representan a las personas u otros sistemas que interactúan directamente con el sistema. Pueden influir en los requerimientos del sistema de algún modo.

Indirecto: representan a los stakeholders que no utilizan el sistema ellos mismos pero que influyen en los requerimientos de algún modo.

Dominio: representan las características y restricciones del dominio que influyen en los requerimientos del sistema.

Los más específicos son: Los proveedores de servicios a sistema / Los receptores de servicios del sistema / Los sistemas que deben interactuar / Las regulaciones y estándares a aplicar / Las fuentes de requerimientos / Los puntos de vista de las personas que lo van a desarrollar, administrar y mantener.

**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. 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.

Con el objetivo de: 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.

**Problemas de comunicación:**

Dificultad para expresar claramente las necesidades. / No ser conscientes de sus propias necesidades. / No entender cómo la tecnología puede ayudar. / Miedo a parecer incompetentes por ignorancia tecnológica. / No tomar decisiones por no poder prever las consecuencias, no entender las alternativas o no tener una visión global. / Cultura y vocabulario diferentes. / Intereses distintos en el sistema a desarrollar. / Medios de comunicación inadecuados (diagramas que no entienden los clientes y usuarios). / Conflictos personales o políticos.

* Limitaciones cogniticas (del desarrollador)
* No conocer el dominio del problema.
* Hacer suposiciones sobre el dominio del problema.
* Hacer suposiciones sobre aspectos tecnológicos.
* Hacer simplificaciones excesivas.
* Conducta humana
* Conflictos y ambigüedades en los roles de los participantes.
* Pasividad de clientes, usuarios o ingenieros de requisitos.
* Temor a que el nuevo sistema lo deje sin trabajo.
* Técnicos
* Complejidad del dominio del problema.
* Complejidad de los requisitos.
* Múltiples fuentes de requisitos.
* Fuentes de información poco claras.

**Técnicas de elicitación**

Recopilación de información:

*Métodos discretos:* 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 interactivos.

* + Muestreo de la documentación, los formularios y los datos existentes:
* Documentos que describen la funcionalidad del negocio que está siendo analizada:

Declaración de la misión y plan estratégico de la organización. / Objetivos formales

del departamento en cuestión. / Políticas, restricciones, procedimientos operativos /

Bases de Datos. / Sistemas en funcionamiento.

* Documentación de sistemas anteriores: Diagramas. / Diccionario o Repositorios de proyecto. / Documentos de diseño. / Manuales de operación y/o entrenamiento.
  + Investigación y visitas al lugar: Investigar el dominio. / Patrones de soluciones (mismo problema en otra organización). / Revistas especializadas. / Buscar problemas similares en internet. / Consultar otras organizaciones.
  + Observación del ambiente de trabajo: El analista se convierte en observador de las personas y actividades con el objeto de aprender acerca del sistema. (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. / 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:* Hay métodos interactivos que pueden usarse para obtener los requerimientos de los miembros de la organización, aunque son distintos en su implementación, estos métodos tienen muchas cosas en común. La base es hablar con las personas en la organización y escuchar para comprender. Cada uno cuenta con su propio proceso establecido

* + Cuestionarios: Documento que permite al analista recabar información y opiniones de los encuestados. Permite comprender actitudes, comportamientos, creencias y características

sobre las personas.

* 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) y Cerradas (SI NO).
* Tipo de información obtenida: Actitud (Lo que las personas dicen que quieren) / Creencias (Lo que las personas creen que es verdad) / Comportamiento (Lo que realmente hacen) / Características (De las personas o cosas)
* Cuándo usar Cuestionarios: Las personas están dispersas geográficamente: Diferentes oficinas o ciudades / Muchas personas involucradas: Clientes o usuarios / Queremos obtener opiniones generales / Queremos identificar problemas generales
* Entrevistas: Técnica de exploración mediante la cual el analista de sistemas recolecta información de las personas a través de la interacción cara a cara. Es una conversación con un propósito específico, que se basa en un formato de preguntas y respuestas en general. Tiene como objetivo conocer opiniones y sentimientos del entrevistado.
* 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 entrevistad0.
* Desventajas: Costosas / Tiempo y recursos humanos / Las entrevistas dependen en gran parte de las habilidades del entrevistador / No aplicable a distancia
* Tipo de entrevistas:

1. Estructuradas (Cerradas) : El encuestador tiene un conjunto específico de preguntas para hacérselas al entrevistado. Se dirige al usuario sobre un requerimiento puntual.
2. No Estructuradas (Abiertas) : El encuestador lleva a un tema en general. Sin preparación de preguntas específicas.
   * Tipos de Preguntas:
3. Abiertas: Permite al encuestado responder de cualquier manera

*«Que, como*»

Ventajas: Revelan nuevas líneas de preguntas / Hacen más

Interesante la entrevista / Permite espontaneidad

Desventajas: Pueden dar muchos detalles irrelevantes / Se puede

Perder el control de la entrevista / Parece que el entrevistador no tiene los objetos claros

2) Cerradas: Las respuestas son directas, cortas o de selección

especifica.

*«Cuantas, quien*»

Ventajas: Ahorran tiempo / Se mantiene más fácil el control de la

Entrevista / Se consiguen datos relevantes

Desventajas: Pueden aburrir al encuestado / No se obtienen detalles

3) Sondeo: Permite obtener más detalle sobre un tema puntual.

*«¿Podría dar detallas de…?*»

* + Como conducir la entrevista:

1. Selección del entrevistado: Según el requerimiento a analizar / Conocer sus fortalezas, prejuicios y motivaciones / Armar la entrevista en base a las características de la persona / Hacer una cita (no llegar sin avisar) / Respetar el horario de trabajo / Establecer la duración de la entrevista / Cuanto mayor es el cargo del entrevistado menor tiempo se debe utilizar / Obtener el permiso del supervisor o jefe / La entrevista es personal y debe realizarse en un lugar privado
2. Preparación de la entrevista: Informar al entrevistado el tema a tratar antes de la reunión / Definir un “Guion de Entrevista” / Se deben evitar preguntas sesgadas o con intención, amenazantes o críticas / Usar lenguaje claro y conciso / No incluir opinión como parte de la pregunta / Evitar realizar preguntas largas y complejas
3. Conducción de la entrevista: Respete el horario y los tiempos definidos / Si es en una sala de reunión llegue antes para asegurar las condiciones de la misma / Inicie la entrevista saludando, presentándose y agradeciendo la atención / Mencione el propósito de la misma y la duración / Escuche con atención y observe al entrevistado, tome nota de las respuestas verbales y no verbales / Concluya la entrevista expresando su agradecimiento / Haga una breve conclusión de la entrevista para ganar la confianza del entrevistado.
   * Planeación conjunta de Requerimientos (JRP o JAD): 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. Si se incorporan prototipos, los mismos ya confirman el diseño del sistema
   * 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.

* Lluvia de Ideas – Brainstorming: 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.
* Ventajas: Clave para resolver la falta de consenso entre usuarios / Ayuda a entender el dominio del problema / Ayuda a entender el usuario y el analista

**PROCESO DE SOFTWARE**

Es el conjunto de métodos, técnicas y prácticas que guían a los ingenieros de software en el desarrollo y evolución del software.

El proceso de software es también llamado ciclo de vida del software, porque describe la vida de un producto de software desde su concepción hasta su implementación, entrega, utilización y mantenimiento. El conjunto de procedimientos debe estar organizado de tal modo que los productos se construyen para satisfacer un conjunto de metas o estándares.

Consta de cuatro pasos genéricos para llevarse a cabo:

1. Especificación del software: donde los ingenieros de sistema hablan con los usuarios finales, clientes (stakeholders) y toda persona relacionada con el sistema para ver cuál es el propósito del mismo, las metas que se proponen y las restricciones.
2. Desarrollo del software: en esta etapa los ingenieros diseñan y programan el sistema.
3. Validación del software: aquí el sistema se válida para asegurar que se hizo lo que el cliente requería.
4. Evolución del software: donde el software se va modificando para adaptarse a los cambios requeridos por el cliente y los cambios del mercado.

**MODELO DE PROCESO DE SOFTWARE**

Es una representación simplificada de un proceso de software que representa una visión de ese proceso. Estos modelos pueden incluir actividades que son partes de los procesos y productos de software, y el papel de personas involucradas.

Existen 3 tipos de modelos generales de proceso de software:

1. Modelo de Cascada: las etapas se presentan cayendo en cascada. Cada etapa de desarrollo se debe completar antes que comience la siguiente.
2. Desarrollo iterativo: un sistema inicial se desarrolla rápidamente a partir de una especificación abstracta. Este se refina basándose en las peticiones del cliente.
3. Desarrollo basado en componentes: este enfoque se basa en la existencia de un número significativo de componentes reutilizables. El proceso de desarrollo del sistema se basa en integrar estos componentes en el sistema más que en desarrollarlos desde cero.

**REQUERIMIENTOS**

Im

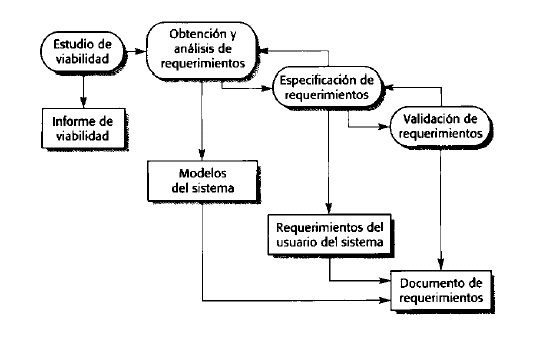
+pacto de los errores en la etapa de requerimientos:

* El software resultante puede no satisfacer a los usuarios.
* Las interpretaciones múltiples de los requerimientos pueden causar desacuerdos entre clientes y desarrolladores.
* Pueden gastarse tiempo y dinero construyendo el sistema erróneo.

**Tipos de requerimientos**

* Requerimientos funcionales:
* Describen una interacción entre el sistema y su ambiente. Como debe comportarse el sistema ante determinado estimulo.
* Describen lo que el sistema debe hacer, o incluso como NO debe comportarse.
* Describen con detalle la funcionabilidad 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 externos: 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.
* Requerimientos del dominio:
* Reflejan las características y restricciones del dominio de la aplicación del sistema. Pueden ser funcionales o no funcionales y pueden restringir a los anteriores. Como se especializan en el dominio son complicados de interpretar.
* Requerimientos por prioridad:
* Que deben ser absolutamente satisfechos.
* Que son deseables, pero no indispensables.
* Que son posibles, pero que podrían eliminarse.
* Requerimientos del Usuario:
* Son declaraciones en lenguaje natural y en diagramas de los servicios que se espera que el sistema provea y de las restricciones bajo las cuales debe operar.
* Pueden surgir problemas por falta de claridad, confusión de requerimientos, conjunción de requerimientos.
* Requerimientos del sistema:
* Establecen con detalle los servicios y restricciones del sistema.
* Es difícil excluir toda la información de diseño (arquitectura inicial, interoperabilidad con sistemas existentes, etc.)

**INGENIERIA DE REQUERIMIENTOS**



**¿Qué es?**

La ingeniería de requerimientos es el proceso por el cual se trasforman los requerimientos declarados por los clientes, ya sean hablados o escritos, a especificaciones precisas, no ambiguas, consistentes y completas del comportamiento del sistema, incluyendo funciones, interfaces, rendimiento y limitaciones.

También es el proceso mediante el cual se intercambian diferentes puntos de vistas para recopilar y modelar lo que el sistema va realizar. Este proceso utiliza una combinación de métodos, herramientas y actores, cuyo producto es un modelo del cual se genera un documento de requerimientos.

**Importancia**

* Permite gestionar las necesidades del proyecto en forma estructurada.
* Mejora la capacidad de predecir cronogramas de proyectos.
* Disminuye los costos y retrasos del proyecto.
* Mejora la calidad del software.
* Mejora la comunicación entre equipos.
* Evita rechazos de usuarios finales.

**ESTUDIO DE VIABILIDAD**

**¿Qué es?**

Para todos los sistemas nuevos, el proceso de ingeniería de requerimientos debería empezar con un estudio de viabilidad. La entrada de este es un conjunto de requerimientos de negocio preliminares, una descripción resumida del sistema y de cómo este pretende contribuir a los procesos de negocio. Los resultados del estudio de viabilidad deberían ser un informe que recomiende si merece o no la pena seguir con la ingeniería de requerimientos y el proceso de desarrollo del sistema.

Un estudio de viabilidad es un estudio corto y debería resolver las siguientes preguntas:

* ¿El sistema contribuye a los objetivos generales de la organización?
* ¿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?

**VALIDACIÓN DE REQUERIMIENTOS**

**¿Qué es?**

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. Se describen 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 mas tarde.

Validación: Al final del desarrollo evaluar el software para asegurar que el software cumple los requerimientos. La validación solo se puede hacer con la activa participación del usuario (hacer el software correcto).

Verificación: El software cumple los requerimientos correctamente (hacer el software correctamente)

**¿Es eficiente validar después del desarrollo del software?**

Cuanto más tarde se detecta, mas cuesta corregir (Boehm). Validar en la fase de especificación de requerimientos puede ayudar a evitar costosas correcciones después el desarrollo.

**¿Contra qué se verifican los requerimientos?**

No existen “Los requerimientos de los requerimientos”. No puede probarse formalmente que un Modelo de Requerimientos es correcto. Puede alcanzarse una convicción de que la solución especificada en el modelo de requerimientos es el correcto para el usuario.

**Comprenden:**

* Verificaciones de validez (para todos los usuarios).
* Verificaciones de consistencias (sin contradicciones).
* Verificaciones de completitud (todos los requerimientos).
* Verificaciones de realismo (se pueden implementar).
* Verificabilidad (se puede diseñar conjunto de pruebas).

**Técnicas de Validación: Pueden ser manuales o automatizadas**

* Revisiones de requerimientos: Informales (los desarrolladores deben tratar los requerimientos con tantos stakeholders como sea posible). Formal (el equipo de desarrollos debe conducir al cliente, explicándole las implicaciones de cada requerimiento).
* Construcción de Prototipos.
* Generación de casos de prueba.

**ESPECIFICACIÓN DE REQUERIMIENTOS**

**Propiedades de los requerimientos**

* Necesaria: Su omisión provoca una deficiencia.
* Conciso:Fácil de leer y entender:
* Completo: No necesita ampliarse.
* Consistente: No contradictorio con otro.
* No ambiguo:Tiene una sola implementación.
* Verificable:Puede testearse a través de inspecciones, pruebas, etc.

**Objetivos**

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

**Aspectos Básicos**

* Funcionabilidad (¿Qué debe hacer el software?)
* Interfaces externas (¡como interactuara el software con el medio externo?)
* Rendimiento (velocidad, disponibilidad, mantenibilidad, eficiencia)
* Atributos (portabilidad, seguridad, mantenibilidad, eficiencia)
* Restricciones de Diseño (estándares requeridos, lenguaje, límite de recursos, etc.)

(teoría 2, página 28, hay un cuadro de “usuarios de un documento de requerimientos”)

**TECNICAS DE ESPECIFICACIÓN DE REQUERIMIENTOS**

* Estática: Se describe el sistema a través de las entidades u objetos, sus atributos y sus relaciones con otros. No describen como las relaciones cambian con el tiempo. Cuando el tiempo no es un factor mayor en las operaciones del sistema, es una descripción útil y adecuada. Define QUE se hace.

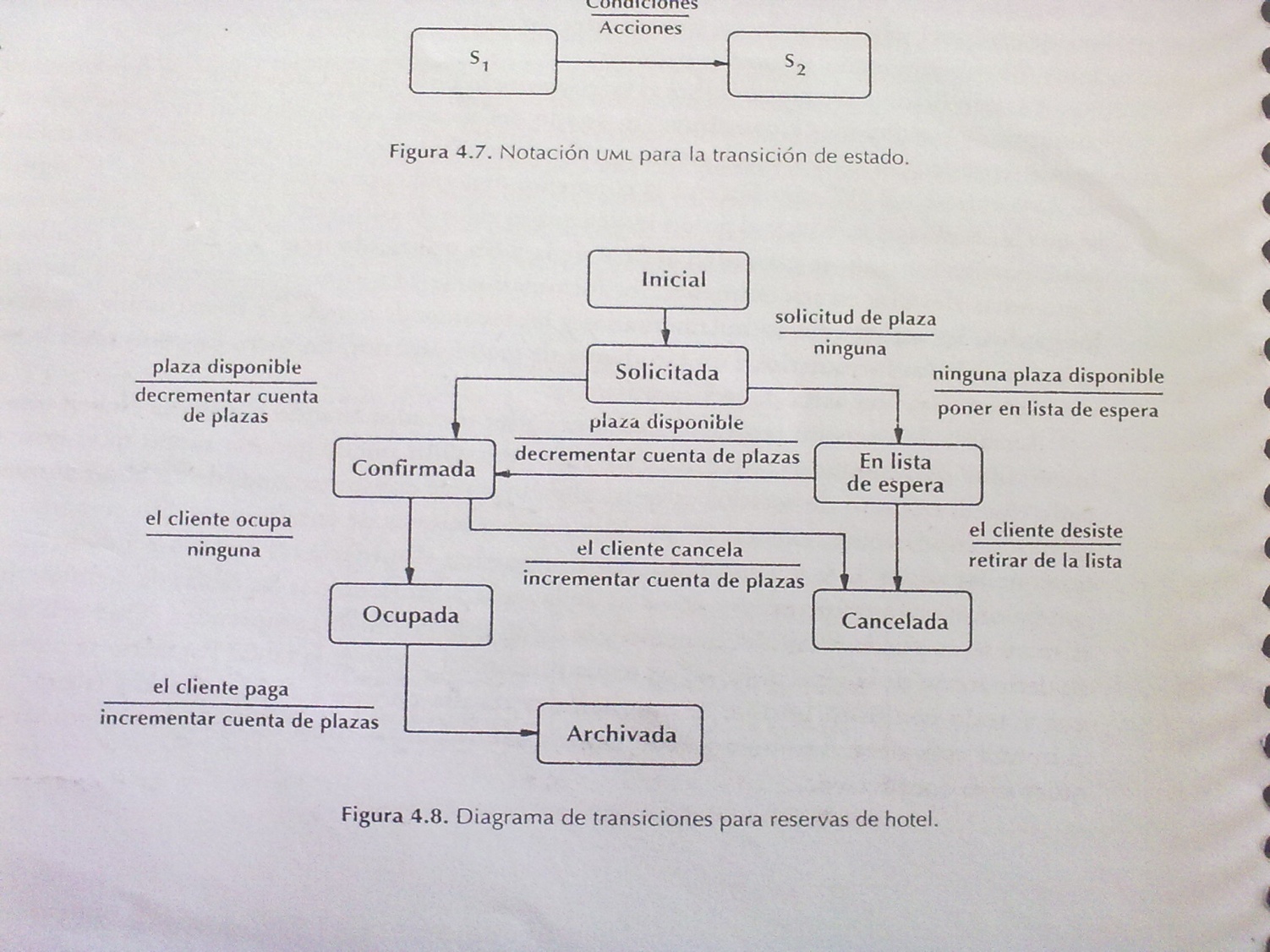
Ejemplos: Referencia indirecta, Relaciones de recurrencia, Definición axiomática, Expresiones regulares, Abstracciones de datos, entre toras.

* Dinámicas: Se consideran un 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: Tablas de decisión, **Diagramas de transición de estados**, Tablas de transición de estados, **Diagrama/Gráficos de persianas**, Diagramas de transición extendidos**, Redes de Petri,** entre otras.

**Diagramas de transición de estado/ Autómata finito / Tablas de decision** (es una técnica de especificación de requerimientos)

* 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
* Una table de decision describe el Sistema como un conjunto de posibles condiciones satisfechas por el Sistema en un momento dado y acciones a ser tomadas como resultado y de reglas para reaccionar ante los estimulos que ocurren cuando se reunen determinados conjuntos de condiciones
* Tienen su origen en la máquina de Estado Finito donde se describe al sistema como un conjunto de estados donde el sistema reacciona a ciertos eventos posibles (externos o internos). Se pueden modelizar como una función en donde aplicada un estado y un estímulo nos va a devolver un nuevo estado de nuestro sistema.
* Como puede ser escrito como una 5-tupla (S,￡,T,s,A) donde: **￡** es un alfabeto (representa uno de los posibles eventos que pueden ocurrir en nuestro problema); **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.
* Tiene un SOLO estado inicial y puede tener MAS de un estado final.En cada estado, el sistema debe responder a todas las condiciones posibles (normales y anormales).
* La reducción de complejidad esta relacionado con la redundancia y no con la contradicción.
* Las tablas de decisión se usan para representar la descripción de situaciones, se representan las distintas alternativas, estados de la naturaleza y las consecuencias, proporcionan una descripción completa, correcta, clara y concisa de una situación que se resuelve por una decisión tomada en un momento específico del tiempo



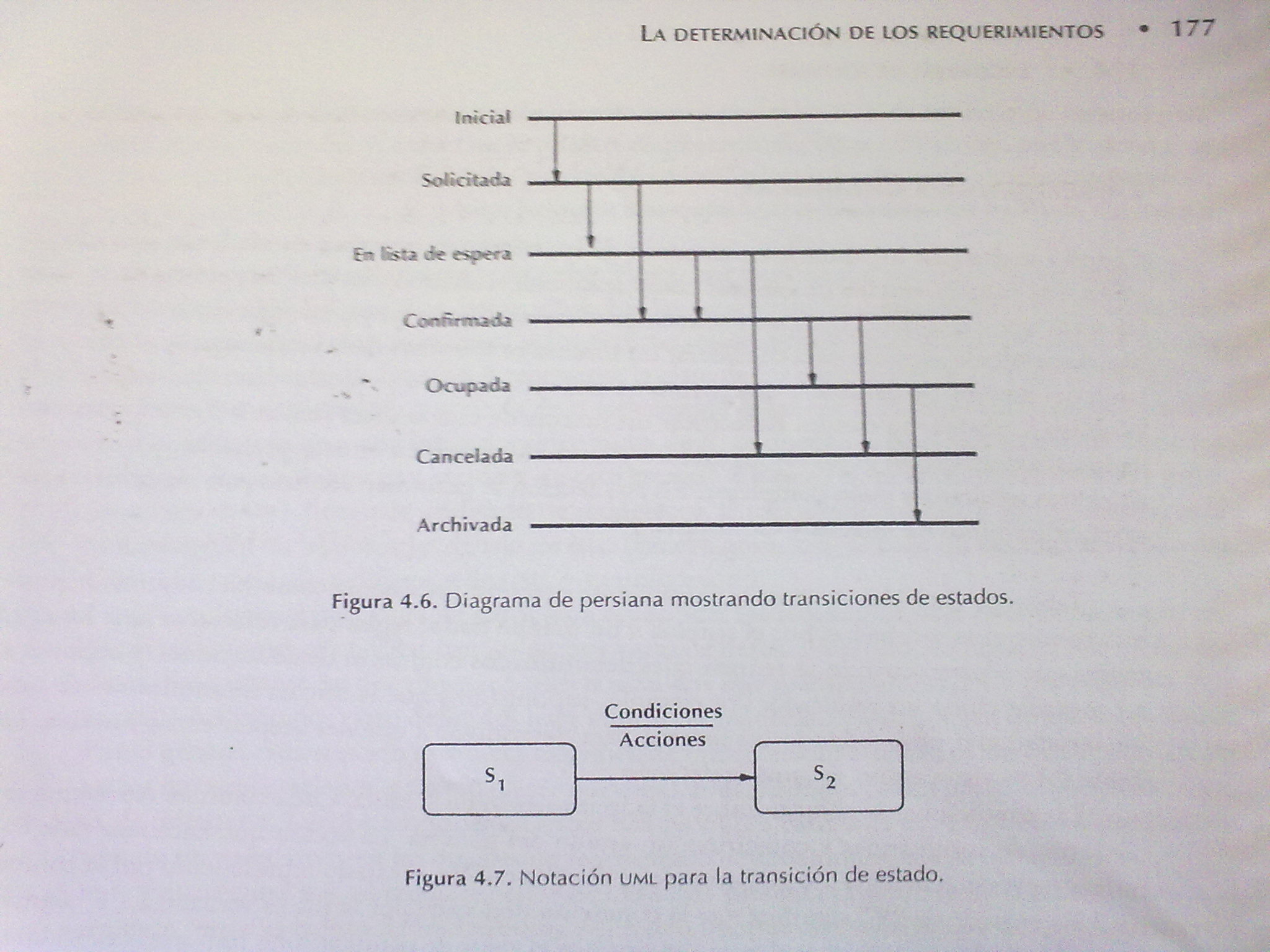
⬆ ⬆

Estados Transicion

Construcción de un Diagrama de Transición de Estado: (ejemplo en la página 15, teoría 4)

* Identificar los estados.
* Si hay un estado complejo se puede explotar.
* Desde el estado inicial, se identifican los cambios de estados con flechas.
* Se analizan las condiciones y las acciones para pasar de un estado a otro
* Se verifica la consistencia (Se han definido todos los estados. Se puede alcanzar a todos los estados. Se pueden salir de todos los estados. En cada estado, el sistema responde a todas las condiciones posibles)

**Diagrama/Gráficos de persianas** (es una técnica de especificación de requerimientos)



⬆ ⬆

Estados Transiciones

**REDES DE PETRI** (es una técnica de especificación de requerimientos)

Fueron inventadas por Carl Petri en la Universidad de Bonn, Alemania Occidental. Utilizadas para especificar sistemas de tiempo real en los que son necesarios representar aspectos de concurrencia. Los sistemas concurrentes se diseñan para permitir la ejecución simultánea de componentes de programación, llamadas tareas o procesos, en varios procesadores o intercalados en un solo procesador. 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. Éstas NO son secuenciales. Va a tener dos elementos: **los eventos** (transiciones(T)) y **estados** (sitios (P)).Son asincrónicas. La ejecución es NO DETERMINISTA. El disparo de una transición es instantáneo.

**Tiene 3 formas para cambiar de estado:**

**\*** f(EstadoA,Evento) = EstadoS.

\* Se requieren varios eventos para pasar de un estado a otro. Los eventos NO ocurren en un orden determinado.

\* Se requieren varios eventos para habilitar el paso del estado a otros varios estados que se ejecutan en paralelo.

**Una Estructura de Red de Petri es una 4-upla:** P (lugares). T (transiciones). I (función de entradas). O (función de salida)

**Los Elementos de un Gráfico de Red de Petri:** Multigrafos (de un nodo puede partir más de un arco), bipartito, dirigido. Los arcos indican a través de una flecha la relación entre sitios u transiciones y viceversa. A los sitios(circulos) se les asignan tokens (fichas) que se representan mediante un numero o puntos dentro del sitio. Esta asignación de tokens a lugares constituyen la marcación. Luego de una marcación inicial se puede simular la ejecución de la red. El número de tokens asignados a un sitio es limitado. El conjunto de tokens asociado a cada estado sirve para manejar la coordinación de eventos y estados. Una vez que ocurre un evento, un token puede “viajar” de uno de los estados a otro. Las reglas de disparo provocan que los tokens “viajen” de un lugar a otro cuando se cumplen las condiciones adecuadas. La ejecución es controlada por el número y distribución de los tokens. Una transición está habilitada cuando cada lugar de entrada tiene al menos tantos tokens como arcos hacia la transición. 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).

**Caracteristicas:**

* Es importante desarrollar modelos de los sistemas de eventos discretos para estudiarlos y comprender su comportamiento.
* Existen herramientas computacionales que permiten analizar este tipo de sistemas, las cuales están basadas en análisis estadísticos y ofrecen soluciones con ciertos grados de incertidumbre.
* Por otro lado, las RdP pueden ser aplicadas para la modelación de sistemas de eventos discretos, las cuales ofrecen una forma de representación gráfica y matemática de los sistemas modelados.
* La formalidad matemática de la RdP proporcionan herramientas de análisis para analizar los posibles estados a los que el sistema modelado pudiera alcanzar.

**HISTORIA DE USUARIO** (es una técnica de especificación de requerimientos)

**Definición:**

Una historia de usuario es una representación de un requisito de software escrito en una o dos frases utilizando el lenguaje común del usuario.

**Conceptos:**

* Debe ser limitada, esta debería poder escribirse sobre una nota adhesiva pequeña.
* Son una forma rápida de administrar los requisitos 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.
* 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 de cada historia de usuario se situé entre unas 10 horas y un par de semanas.
* Estimaciones mayores a dos semanas son indicativo de que la historia es muy compleja y debe ser dividida en varias historias.

**Forma de Redactarlas**

* Si bien el estilo puede ser libre, la historia de usuario debe responder a tres preguntas: ¿Quién se beneficia? , ¿Qué se quiere? , ¿Cuál es el beneficio?
* Esquema: Como (ROL) quiero (ALGO) para poder (BENEFICIO).
* Ejemplos:

-Como usuario registrado deseo loguearme para poder empezar a utilizar la aplicación.

-Como cliente, quiero suscribirme por medio del sitio web y obtener un nuevo plan de T.V por cable.

**Características**

* Independientes unas de otras: De ser necesario, combinar las historias dependientes o buscar otra forma de dividir las historias de manera que resulten independientes.
* Negociables:La historia en sí mismo no es lo suficientemente explicita como para considerarse un contrato, la discusión con los usuarios debe permitir esclarecer su alcance y este debe dejarse explicito bajo la forma de pruebas de validación.
* Valoradas por los clientes y usuarios:Los intereses de los clientes y de los usuarios no siempre coinciden, pero en todo caso, cada historia debe ser importante para alguno de ellos más que para el desarrollador.
* Estables: Un resultado de la discusión de una historia de usuario es la estimación del tiempo que tomara completarla. Esto permite estimar el tiempo total del proyecto.
* Pequeñas:Las historias muy largas son difíciles de estimare imponen restricciones sobre la planificación de un desarrollo iterativo. Generalmente se recomienda la consolidación de historias muy cortas en una sola historia.
* Verificables: Las historias de usuario cubren requerimientos funcionales, por lo que generalmente son verificables. Cuando sea posible, la verificación debe automatizarse, de manera que pueda ser verificada en cada entrega del proyecto.

**Criterios de Aceptación**

* Un criterio de aceptación es el criterio por el cual se define si una historia de usuario fue desarrollada según la expectativa del Product/Manager/Owner (como representante de los criterios del cliente) y si se puede dar como hecha.
* Deben ser definidos durante la etapa inicial antes de la codificación, acompañan a la historia de usuario, porque implementan la historia de usuario y ayudan al equipo de desarrollo a entender mejor como se espera que el producto se comporte.
* Los criterios de aceptación son utilizados para expresar el resultado de las conversiones 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.
* Si una historia de usuario tiene más de 4 criterios de aceptación, debe evaluarse subdividir la historia
* Puede añadirse un numero de escenario para identificar al criterio, asociado a la historia de usuario en cuestión.

**Beneficios**

* Al ser muy corta, esta 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 contrato 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

**Épicas**

* Se denomina Épica a un conjunto de Historia de usuario que se agrupan por algún denominador común

**CASOS DE ESO** (es una técnica de especificación de requerimientos) (ejemplo pag 25 teoría 3)

**¿Qué es?**

Proceso de modelado de las “Funcionalidades” del sistema en términos de los eventos que interactúan entre los usuarios y el sistema.

Tiene sus orígenes en el modelado orientado a objetos (Jacobson 1992) pero su eficiencia en modelado de requerimientos hizo que se independice de la técnica de diseño utilizada, siendo aplicable a cualquier metodología de desarrollo.

El uso de Casos de Uso facilita y alienta la participación de los usuarios.

**Beneficios**

* Herramienta para capturar requerimientos funcionales.
* Descompone el alcance del sistema en piezas más manejables.
* Medio de comunicación con los usuarios.
* Utiliza lenguaje común y fácil de entender por las partes.
* Permite estimar el alcance del proyecto y el esfuerzo a realizar.
* Define una línea base para la definición de los planes de prueba.
* Define una línea base para toda la documentación del sistema.
* Proporciona una herramienta para el seguimiento de los requisitos.

**Componentes**

* Diagramas de Casos de Uso: (teoría 3 tiene diagramas sobre las RELACIONES para entenderlas mejor)
* ilustra las interacciones entre el sistema y los actores.
* Representa un objetivo (funcionabilidad) individual del sistema y describe la secuencia de actividades y de interacciones para alcanzarlo.
* Actores: Un actor inicia una actividad (CU) en el sistema. Representa un papel desempeñado por un usuario que interactúa (rol). Puede ser una persona, sistema externo o dispositivo externo que dispare un evento (sensor, reloj).
* Relaciones:*Asociaciones* (relación entre un actor y un CU en el que interactúa entre sí). *Extensiones* (un CU extiende la funcionabilidad de otro CU. Un CU puede tener muchos CU extensiones. Los CU extensiones solo son iniciados por un CU), *Uso o Inclusión* (reduce la redundancia entre dos o más CU al combinar los pasos comunes de los CU). *Dependencia* (relación entre CU que indica que un CU no puede realizarse hasta que se haya realizado otro CU). *Herencia* (relación entre actores donde un actor hereda las funcionabilidades de uno o varios actores)
* Escenarios (narración del CU):
* descripción de la interacción entre el actor y el sistema para realizar la funcionabilidad y los eventos alternativos.
* Pasos:

Identificar a los actores: Para encontrar actores potenciales debemos buscar en documentación o manuales existentes, minutas de reunión, documentación de requerimientos. Debemos saber responder a: ¿Quién o que proporciona las entradas al sistema?, ¿Quién o que recibe las salidas del sistema?, ¿Se requieren interfaces con otros sistemas?, ¿Quién mantendrá la información en el sistema? Deberían nombrarse con un sustantivo o frase sustantiva.

Identificar los CU para los requerimientos: Saber responder a: ¿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?

Construir el diagrama de CU.

Realizar los escenarios: Un CU debe representar una funcionabilidad concreta. La descripción de los pasos en los escenarios debe contener más de un paso, para representar la interacción entre los componentes. El uso de condicionales en el curso normal, es limitado a la invocación de excepciones, ya que este flujo representa la ejecución del caso sin alteraciones. Las pre-condiciones no deben representarse en los curos alternativos, ya que al ser una pre-condición no va a ocurrir. Los “uses” deben ser accedidos por los menos desde dos CU.

**TABLAS DE ESTADO(TD)** (es una técnica de especificación de requerimiento)

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. No permiten representar aspectos de concurrencia. Son utilizadas para especificar sistemas de lógica compleja.

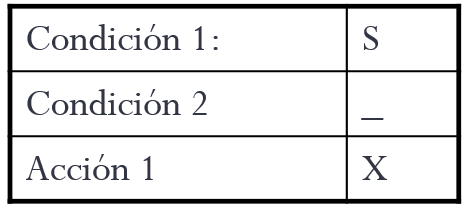
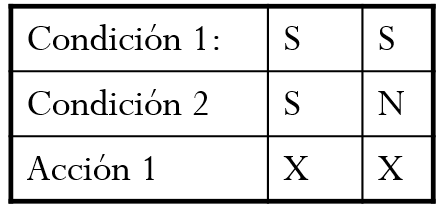
Describe el sistema como un conjunto de: Posibles CONDICIONES satisfechas por el sistema en un momento dado. REGLAS para reaccionar ante los estímulos que ocurren cuando se reúnen determinados conjuntos de condiciones y ACCIONES a ser tomadas como un resultado.

Construiremos las tablas con:

* + - *condiciones simples y acciones simples*
    - *Las condiciones toman sólo valores Verdadero o Falso*
    - *Hay 2N Reglas donde N es la cantidad de condiciones*

¿Cómo se llena la tabla?

A partir de un enunciado se debe:

* 1. Identificar las condiciones y las acciones.
  2. Completar la tabla teniendo en cuenta:
     1. *Si hay condiciones que son opuestas, debe colocarse una de ellas porque por la negativa se “obtendrá” la otra. (Si son n condiciones excluyentes, colocar n-1 en la tabla).*
     2. *Las condiciones deben ser atómicas. (un solo valor simple)*
     3. *Las especificaciones deben ser COMPLETAS (Aquellas que determinan acciones (una o varias) para todas las reglas posibles).*
     4. *Pueden aparecer especificaciones RENDUNDANTES: Aquellas que marcan para reglas que determinan las mismas condiciones acciones iguales, o que si solo hay dos condiciones por ejemplo y la condición 1 es V y la acción siempre es la misma sin importar que valor tome la condición 2, entonces en la acción 2 se la reemplaza por un guion:*
     5. *También existen las especificaciones CONTRADICTORIA (Aquellas que especifican para reglas que determinan las mismas condiciones acciones distintas) la cual NO debemos tener en nuestra tabla, ya que no es lo correcto.*
  3. Se construyen las reglas

(ejemplo pagina 8 y 14 de la teoría 6-TD)

**ANALISIS ESTRUCTURADO** (es una técnica de especificación de requerimiento)

Es una actividad de construcción de modelos. Mediante una notación creamos modelos que representan el contenido y flujo de la información (datos y control).

No es un método sencillo siempre de la misma forma por todos los que lo usan.

El modelo de análisis debe lograr TRES objetivos primarios:

* Describir lo que se requiere el cliente.
* Establecer una base para la creación de un diseño de Software
* Definir un conjunto de requisitos (pruebas) que se pueda validar una vez que se construye el software.

Está compuesto por TRES modelos que deben trabajar de manera conjunta y están unidos por el Diccionario de datos:

* Modelo de datos del sistema: Diagrama de Entidad-Relación (Si me quiero acercar al diseño seria hacer una descripción de objetos de datos)
* Modelado de funciones del sistema: vamos a usar Diagrama de Flujo de Datos que es una herramienta que permite visualizar un sistema como una red de procesos funcionales, conectados entre sí por “conductos” y almacenamientos de datos, esta herramienta lo que hace es representar la transformación de entradas a salidas y es también llamado diagrama de burbujas. Es una herramienta comúnmente utilizada por sistemas operacionales en los cuales las funciones del sistema son de gran importancia y son más complejas que los datos que este maneja. (Si me quiero acercar al diseño seria hacer una especificación de procesos)

**Para su representación vamos a usar:**

* 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 Circulo (también llamado burbuja): que representa un proceso o transformación que es aplicado a los datos (o al control) y los modifica.
* Una Flecha: que representa uno o más elementos de datos (objetos de datos)
* Modelado de comportamiento del sistema: Diagrama de Transición de Estados (Si me quiero acercar al diseño seria hacer una especificación de control)

Diccionario de Datos:



Listado organizado de todos los datos pertinentes al sistema. Es una definición sin ambigüedad de los datos y elementos del sistema

Características

* + - *Permite revisar consistencia*
    - *Representa el contenido de la información*
    - *Define el significado de los flujos y los almacenes*
    - *Un Dato debe contener* 
      * Tipo
      * Nombre
      * Descripción

**Propuesta de Yourdon**

* Modelo Esencial

Debe indicarse lo que el sistema *debe* hacer para satisfacer los requerimientos del usuario, con una mínima (en lo posible nula) explicación de cómo lo hace.

Evitar el detalle de cualquier restricción o aspecto derivado de la implementación.

Pensar el modelo esencial "*suponiendo que se dispone de tecnología perfecta*", lo que permite

que sobreviva cambios tecnológicos.

La mayoría de los usuarios están metidos en los detalles de la implantación de su sistema

actual y les es difícil enfocar un sistema "DE TECNOLOGIA PERFECTA".

*El modelo esencial esta formado por el modelo ambiental y el modelo de comportamiento*.

* + Componentes

**1- Modelo Ambiental**

* + La construcción de un modelo ambiental es lo primero y más importante en la construcción del modelo de requerimientos del usuario para el nuevo sistema
  + Pero a medida que encaramos un proyecto mayor, hay cientos de flujos, decenas de terminadores y la lista de acontecimientos crece y es difícil de manejarla.
  + Una vez concluido el modelo ambiental hay que chequearlo con los usuarios clave y con el grupo de análisis para que sea la base del modelo de comportamiento del sistema.

Define las interfaces entre el sistema y el ambiente donde el mismo se ejecuta.

**1.1 DECLARACIÓN DE PROPÓSITO**

* + - * En forma sintética (1 párrafo con 2 o 3 frases) debe indicarse el objetivo del sistema, de que es responsable el sistema

**1.2 DIAGRAMA DE CONTEXTO**

* + - * Es un caso especial de DFD donde el sistema se representa en una sola burbuja vinculada con las entidades externas y los almacenamientos externos

**1.3 LISTA DE ACONTECIMINTO**

* + - * Se trata de un listado de eventos (“estímulos”) a los que el sistema debe responder.
      * Tipos de Acontecimientos:

Flujo (F): llega algún o algunos datos al sistema

Temporales (T): comienzan con la llegada de un momento dado en el tiempo.

Control (C).

**2- Modelo de comportamiento**

El modelo de comportamiento es la representación del comportamiento final que el sistema debe tener para manejar con éxito el ambiente, dentro de las especificaciones requeridas por el usuario.

* + - El modelo preliminar de comportamiento contiene:
      * Un diagrama preliminar de flujo de datos del sistema (DFD)
      * Un diagrama preliminar de entidad-relación (DER)
      * Una primera versión del diccionario de datos (DD)
      * Un diagrama de transición de estados (DTE)
    - El desarrollo descendente del modelo preliminar propone partir directamente del diagrama de contexto y obtener una primera versión (Nivel 0) del DFD. Problema: Parálisis del análisis.
    - Yourdon propone partir de la lista de acontecimientos y obtener una primera versión (Nivel “N”) del DFD. Luego ir obteniendo los niveles superiores (N-1, N-2,…) hasta llegar al Nivel 0 y los niveles inferiores (N+1, N+2,…) hasta que no se pueda descender más.
    - ***Nivelación de un DFD***
      * A partir del DFD preliminar se realizan nivelaciones
      * **Ascendentes** 
        + Agrupa las burbujas con algún criterio
      * **Descendentes** 
        + Descompone las burbujas funcionalmente

**MODELO DE PROCESOS**

(termino equivalente: paradigma de software y ciclo de vida de software)

* Es un conjunto de actividades y resultados asociados que producen un producto de software.
* Es una representación simplificada de un proceso de software que presenta una visión de ese proceso.
* Estos modelos pueden incluir actividades que son partes de los procesos y productos de software, y el papel de las personas involucradas.

**También podemos llamar a un proceso como:**

* 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
  + Es una representación abstracta de un proceso del software.

**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.*

**Podemos establecer dos tipos de Modelo de Proceso de Software**

* Modelos prescriptivos

Prescriben un conjunto de elementos del proceso: actividades del marco de trabajo, acciones de la ingeniería del software, tareas, aseguramiento de la calidad y mecanismos de control.

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

Descripción en la forma en que se realizan en la realidad (modelo de cascada)

* **Ambos modelos deberían ser iguales**

**Tipos de Modelo de Procesos**

* Modelo en Cascada
  + Las etapas se representan cayendo en cascada.
  + Cada etapa de desarrollo se debe completar antes que comience la siguiente.
  + Útil para diagramar lo que se necesita hacer.
  + Su simplicidad hace que sea fácil explicarlo a los clientes
  + **Dificultades:**
    - No existen resultados concretos hasta que todo esté terminado.
    - Las fallas más triviales se encuentran al comienzo del período de prueba y las más graves al final.
    - La eliminación de fallas suele ser extremadamente difícil durante las últimas etapas de prueba del sistema.
    - Deriva del mundo del hardware y presenta una visión de manufactura sobre el desarrollo de software.
    - La necesidad de pruebas aumenta exponencialmente durante las etapas finales.
    - "CONGELAR" una fase es poco realista.
    - Existen errores, cambios de parecer, cambios en el ambiente*.*
* **Modelo en cascada con prototipo**: comunicación ordenada, reduce errores
* Modelo en V
  + Demuestra cómo se relacionan las actividades de prueba con las de análisis y diseño.
  + Sugiere que la prueba unitaria y de integración también sea utilizada para verificar el diseño del programa
  + La vinculación entre los lados derecho e izquierdo implica que, si se encuentran problemas durante la verificación y validación, entonces el lado izquierdo de la V puede ser ejecutado nuevamente para solucionar el problema.
* Modelo de Prototipos
  + Un prototipo es un producto parcialmente desarrollado que permite que clientes y desarrolladores examinen algunos aspectos del sistema propuesto, y decidan si éste es adecuado o correcto para el producto terminado.
  + Esta es una alternativa de especificación para tratar mejor la incertidumbre, la ambigüedad y la volubilidad de los proyectos reales.
  + Tipos
  + **Evolutivos:** El objetivo es obtener el sistema a entregar. Permite que todo el sistema o alguna de sus partes se construyan rápidamente para comprender o aclarar aspectos y asegurar que el desarrollador, el usuario y el cliente tengan una comprensión unificada tanto de lo que se necesita como de lo que se propone como solución
* **Descartables:** No tiene funcionalidad. Se utilizan herramientas de modelado
* 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*
  + Para asegurar el éxito
    - Debe ser un sistema con el que se pueda experimentar
    - Debe ser comparativamente barato (< 10%)
    - Debe desarrollarse rápidamente
    - Énfasis en la interfaz de usuario
    - Equipo de desarrollo reducido
    - Herramientas y lenguajes adecuados
* Desarrollo por fases. Incremental. Iterativo (grafico teoría 7 pág. 35)
* Se desarrolla el sistema de tal manera que puede ser entregado en piezas. Esto implica que existen dos sistemas funcionando en paralelo: el sistema operacional y el sistema en desarrollo.
* **Tipos**
* **Incremental:** El sistema es particionado en subsistemas de acuerdo con su funcionalidad. Cada entrega agrega un subsistema.
* **Iterativo:** Entrega un sistema completo desde el principio y luego aumenta la funcionalidad de cada subsistema con las nuevas versiones.
* El modelo espiral
* 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
* Cada ciclo empieza identificando:

*Los objetivos de la porción correspondiente*

*Las alternativas*

*Restricciones*

* Cada ciclo se completa con una revisión que incluye todo el ciclo anterior y el plan para el siguiente

**METODOLOGIAS AGILES**

* En los años 80 y principios de los 90, existía una opinión general de que la mejor forma de obtener un mejor software era a través de una planificación cuidadosa del proyecto, la utilización de métodos de análisis y diseño, y procesos de desarrollo de software controlados y rigurosos.
* En general se realizaban sistemas críticos, desarrollados por grandes equipos, a menudo dispersos geográficamente.
* Sin embargo, cuando este enfoque fue aplicado a sistemas de negocio pequeños y de tamaño medio, el esfuerzo invertido era grande, y cuando cambiaban los requerimientos, se hacía esencial rehacer el trabajo.
* Del descontento nacieron las metodologías ágiles.
* El éxito de un desarrollo está dado por la metodología empleada la cual nos da una dirección a seguir para su correcta conclusión.
* Generalmente esta metodología lleva asociado un marcado énfasis en el control del proceso, definiendo roles, actividades, herramientas y documentación detallada.
* Este enfoque no resulta ser muy adecuado para proyectos actuales donde el entorno del sistema es muy cambiante y se exige una reducción de tiempo.
* Ante estas dificultades, muchos equipos se resignan a prescindir de las buenas prácticas, asumiendo los riesgos.
* En este contexto, las metodologías ágiles emergen como una posible solución.

**¿Qué son las metodologías agiles?**

Es un enfoque *iterativo* (es una estrategia de reproceso en la que el tiempo se separa para revisar y mejorar partes del sistema) e *incremental* (es una estrategia programada y en etapas, en las que las diferentes partes del sistema se desarrollan en diferentes momentos o a diferentes velocidades, y se integran a medida que se completan) de 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”, adaptándose además rápidamente al cambio de los proyectos.

**Objetivos:**

* + Producir software de alta calidad con un costo efectivo y en el tiempo apropiado.
  + Esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.
  + Ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

**Un Poco de Historia...**

Así es como nace “The Agile Alliance”(AA), [AAlliance-www], una organización dedicada a promover los conceptos de desarrollo de software ágil, y de ayudar a las organizaciones a adoptar dichos conceptos.

La definición moderna de desarrollo ágil de software evolucionó a mediados de los años 1990 y en el año 2001, miembros prominentes de la comunidad se reunieron en Snowbird, Utah”

**Valores**

* + ***Individuos e interacciones*** más que procesos y herramientas.
  + ***Software operante*** más que documentaciones completas.
  + ***Colaboración con el cliente*** más que negociaciones contractuales.
  + ***Respuesta al cambio*** más que apegarse a una rigurosa planificación.

(verlo como preferencias y no alternativas)

**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.

Metodologia Agil: Pocos artefactos / Pocos Roles / No existe un contrato tradicional o al menos es bastante flexible / El cliente es parte del equipo de desarrollo (además in-situ) / Grupos pequeños (< 10 integrantes) y trabajando en el mismo sitio / Menos énfasis en la arquitectura

Metodologia NO Agil: Mas artefactos /Mas Roles / Existe un contrato prefijado / El cliente interactúa con el equipo de desarrollo mediante reuniones / Grupos grandes / La arquitectura es esencial

**Desventajas**

En la práctica, los principios que subyacen a los métodos ágiles son a veces difíciles de cumplir:

**. 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.

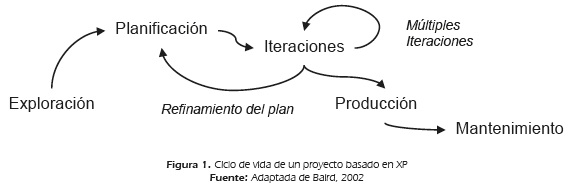
* Por lo general, el documento de requerimientos del software forma parte del contrato entre el cliente y el proveedor. Como en los métodos ágiles se minimiza la documentación, suele ser complejo reglamentarlo.
* La mayoría de los libros que describen los métodos ágiles y las experiencias con éstos hablan del uso de dichos métodos para el desarrollo de nuevos sistemas. Sin embargo, una enorme cantidad de esfuerzo en ingeniería de software se usa en el mantenimiento y la evolución de los sistemas de software existentes. Al no existir documentación se complejizaría.

**Principales Metodologias Agiles**

* XP Extreme Programming [www.extremeprogramming.org](http://www.extremeprogramming.org/)

Es una disciplina de desarrollo de software basado en los valores de la *sencillez*, la *comunicación*, la *retroalimentación*, la *valentía* y el *respeto*. Su acción consiste en llevar a todo el equipo reunido en la presencia de 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: *Programador* (Responsable de decisiones técnicas / Responsable de construir el sistema / Sin distinción entre analistas, diseñadores o codificadores / En XP, los programadores diseñan, programan y realizan las pruebas) – *Jefe de* *Proyecto* (Organiza y guía las reuniones / Asegura condiciones adecuadas para el proyecto) - *Cliente* (Es parte del equipo / Determina qué construir y cuándo / Establece las pruebas funcionales) – *Entrenador “Coach”* (Responsable del proceso /Tiende a estar en un segundo plano a medida que el equipo madura) – *Encargado De Pruebas “Tester”* (Ayuda al cliente con las pruebas funcionales / Se asegura de que las pruebas funcionales se superan) – *Rastreador* *“Tracker*”
* Proceso:



*El ciclo de vida consiste en:*

* + - * 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.

La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

* + - * 1. *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.

Esta fase dura unos pocos días.

* + - * 1. *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.

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado.

* + - * 1. *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.
        2. *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.

* + - * 1. *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.

* Practicas:

1. Testing: Los programadores continuamente escriben pruebas unitarias, las cuales deben correr sin problemas para que el desarrollo continúe. Los clientes escriben pruebas demostrando que las funcionalidades están terminadas.
2. Refactoring: Actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios.
3. Programación de a Pares:

Todo el código de producción es escrito por dos programadores en una máquina.

1. Propiedad Colectiva del Código: Cualquiera puede cambiar código en cualquier parte del sistema en cualquier momento. Motiva a contribuir con nuevas ideas, evitando a la vez que algún programador sea imprescindible.
2. Integración Continua: Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día. Reduce la fragmentación de los esfuerzos de los desarrolladores por falta de comunicación sobre lo que puede ser reutilizado compartido.
3. Semana de 40-horas: Se debe trabajar un máximo de 40 horas por semana. El trabajo extra desmotiva al equipo. Los proyectos que requieren trabajo extra para intentar cumplir con los plazos suelen al final ser entregados con retraso. En lugar de esto se puede realizar el juego de la planificación para cambiar el ámbito del proyecto o la fecha de entrega.
4. Cliente en el Lugar de Desarrollo: El cliente tiene que estar presente y disponible todo el tiempo para el equipo.
5. Estándares de Codificación: Los programadores escriben todo el código de acuerdo con reglas que enfatizan la comunicación a través del mismo.

* Scrum
* es un proceso en el que se aplican, de manera regular, un conjunto de mejores prácticas para trabajar en equipo y obtener el mejor resultado posible de un proyecto.
* Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.
* En Scrum se realizan entregas parciales y regulares del resultado final del proyecto, priorizadas por el beneficio que aportan al receptor del proyecto

Principios

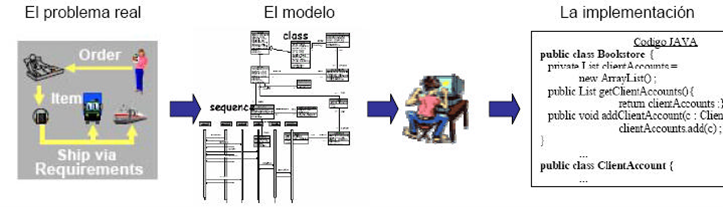
* **Eliminar el desperdicio**: no generar artefactos, ni perder el tiempo haciendo cosas que no le suman valor al cliente.
* **Construir la calidad con el producto:** la idea es inyectar la calidad directamente en el código desde el inicio.
* **Crear conocimiento:** En la práctica no se puede tener el conocimiento antes de empezar el desarrollo.
* **Diferir las decisiones:** tomar las decisiones en el momento adecuado, esperar hasta ese momento, ya que uno tiene mas 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.
* Las características esenciales son:
* 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.
* Artefactos: **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. **Sprint Backlog:** es la lista que contiene toda la funcionalidad que el equipo se comprometió a desarrollar durante un Sprint determinado. **Burndown Chart:** muestra un acumulativo del trabajo hecho, día-a-día. Entre otros…
* Proceso: Scrum es **iterativo e incremental** 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 éstas etapas juntas que se ejecutan una y otra vez, hasta que se crea suficiente.

¿Cuándo usar Scrum?

* 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.
* Esos proyectos difíciles, que con los enfoques tradicionales se hace imposible llegar a buen puerto.
* DSDM (Dynamic Systems Development Method) [www.dsdm.org](http://www.dsdm.org/)
* Crystal Methods (Cockburn’s Crystal Family Methodologies) [www.crystalmethodologies.org](http://www.crystalmethodologies.org/)
* ASD Adaptative Software Development [www.adaptivesd.com](http://www.adaptivesd.com/)
* FDD Feature-Driven Development

**El Desarrollo de Software Basado en Modelos. (MBD)**

* Hacia fines de los 70’ De Marco introdujo el concepto de desarrollo de software basado en modelos. Destacó que la construcción de un sistema de software debe ser precedida por la construcción de un modelo, tal como se realiza en otros sistemas ingenieriles.
* 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)

Construcción de un Sistema de Software:

* Construcción de un Sistema de Software

**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.
* *Promueve enfatizar los siguientes puntos claves:*

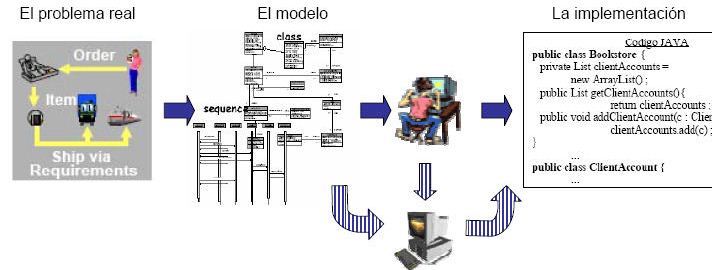
Mayor nivel de abstracción en la especificación tanto del problema a resolver como de la solución correspondiente.

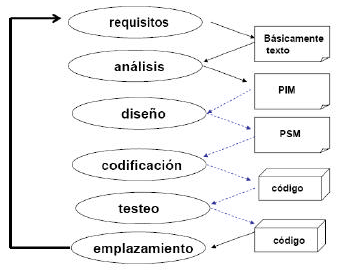
Aumento de confianza en la automatización asistida por computadora para soportar el análisis, el diseño y la ejecución.

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.

* 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.



Ciclo de vida del Software Dirigido por Modelos: 

Los tres pasos principales en el proceso de  
desarrollo MDD.

* PIM: “Un modelo de un sistema que no contiene información acerca de la plataforma o la tecnología que es usada para implementarlo”
* PSM: “Un modelo de un sistema que incluye información acerca de la tecnología específica que se usará para su implementación sobre una plataforma específica”
* Transformación de modelos: “Especifica el proceso de conversión de un modelo en otro modelo del mismo sistema.” En general, se puede decir que una definición de transformación consiste en una colección de reglas, las cuales son 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

Beneficios de MDD.

* 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.

**CALIDAD**

**¿Qué es?**

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

Las principales normas internacionales definen la calidad como:

* “El grado en el que un conjunto de características inherentes cumple con los requisitos “(ISO 9000)
* “Conjunto de propiedades o características de un producto o servicio que le confieren aptitud para satisfacer unas necesidades expresadas o implícitas” (ISO 8402)

**Criterios erróneos comunes sobre la calidad**

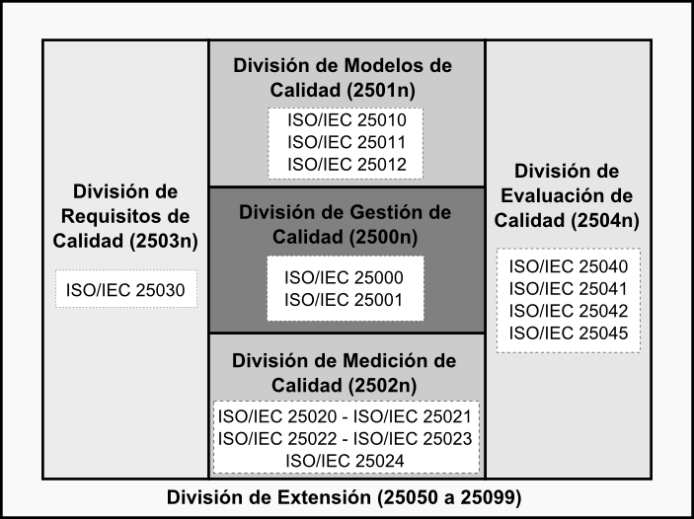
* Un producto de calidad es un producto de lujo.
* La calidad es intangible y por lo tanto no mensurable.
* Los problemas son originados por los trabajadores de producción.
* La calidad se origina en el Depto de calidad

**CALIDAD DE LOS SISTEMAS DE INFORMACION (O DE SOFTWARE)**

“Conjunto de propiedades o características de un producto o servicio que le confieren aptitud para satisfacer unas necesidades expresadas o implícitas” (ISO 8402)

1. Se divide en: (SON DEPENDIENTES)
   1. Calidad del producto obtenido: La estandarización del producto define las propiedades que debe satisfacer el producto software resultante.
   2. Calidad del proceso de desarrollo: La estandarización del proceso define la manera de desarrollar el producto software.

Norma/Modelo de Calidad SQuaRE ISO/IEC 25000



“División de requisitos de calidad (2503n)”: Ayuda a especificar los requisitos de calidad que pueden ser usados en el proceso de elicitación.

“División de medición de calidad (2502n)”: Modelo de referencia de la medición de la calidad del producto, definiciones de ***medidas de calidad*** y guías prácticas de uso

“División de gestión de calidad (2500n)”: Las normas que forman este apartado definen todos los modelos, términos y definiciones comunes referenciados por toda la serie SQuaRE

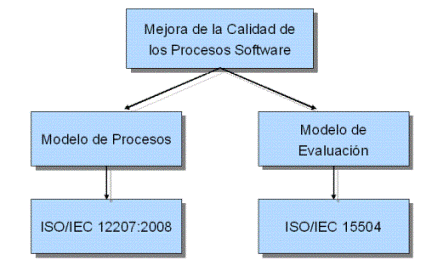
“División de modelos de calidad (2501n)”: Modelo de calidad detallado incluyendo ***características para calidad interna y externa y la calidad de datos***.

“División de evaluación de calidad (2504n)”: Requisitos, recomendaciones y guias para la evaluación de producto.

Norma/Modelo de Calidad SQuaRE ISO/IEC 25010- Características ▼



Norma/Modelo de Calidad Software:



ISO/IEC 12207 establece un modelo de procesos para el ciclo de vida del software

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

**CMM (1993) – CMMI (2000)**

* Modelo de evaluación de los procesos de una organización.
* Fue desarrollado inicialmente para los procesos de desarrollo e implementación de software por la Universidad Carnegie-Mellon para el Software Engineering Institute (SEI).
* Marco de referencia para desarrollar procesos efectivos
* Proporciona un marco estructurado para evaluar los procesos actuales de la organización, establecer prioridades de mejora, e implementar esas mejoras

Posee dos vistas que permiten un enfoque diferente según las necesidades de quien vaya a implementarlo (CMMI):

* 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 (seis) niveles de capacidad. Los niveles de **capacidad** indican qué tan bien se desempeña la organización en un área de proceso individual.

Niveles de madurez

1. Proceso impredecible, poco controlado y reactivo (Inicial)
2. Proceso caracterizado por **proyectos** y frecuentemente reactivo (Gestionado)
3. Proceso caracterizado por la **organización** y proactivo (Definido)
4. El proceso es controlado cuantitativamente (Gestionado Cuantitativamente)
5. Enfoque en la mejora del proceso (Optimizado)

ISO - lnternational Organization for Standardization

La familia **ISO 9000** es un conjunto de normas de “gestión de la calidad” aplicables a cualquier tipo de organizacisón

con el objetivo de obtener mejoras en la organización y, eventualmente arribar a una certificación, punto importante a la hora de competir en los mercados globales.

* **ISO – 9001:2015** - Quality management system – Requirements

Norma publicada por ISO en el año 2015.

* **IRAM – ISO 9001:2015** – Sistema de gestión de la calidad – Requisitos

Norma publicada por ISO y traducida por IRAM.

* + **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 la evidencia 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 Sistema de Gestión de la Calidad, ayuda a implementar procesos simplificados y mejorar la eficiencia operacional.
* Asegura la toma de decisiones y mejore la satisfacción del cliente.
* Optimiza sus operaciones para así cumplir y superar los requisitos de sus clientes.
* Mejora su rendimiento financiero.