Técnicas de especificación de requerimientos móviles

CASOS DE USO

Es el proceso de modelado de las "funcionalidades" del sistema en término de los eventos que interactúan e/ el usuario 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 esta técnica facilita y alienta la participación de los usuarios.

Beneficios: Herramienta para capturar requerimientos funcionales. Descompone el alcance del sistema en piezas manejables. Medio de comunicación con los usuarios. 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.

Elementos:

- Diagrama de Casos de Uso (ilustra las interacciones entre el sistema y los actores).
- Escenarios (descripción de la interacción entre el actor y el sistema para realizar la funcionalidad).
- Caso de uso (representa un objetivo (funcionalidad) individual del sistema y describe la secuencia de actividades y de interacciones para alcanzarlo. Para que el CU sea considerado un requerimiento debe estar acompañado de su respectivo escenario).
- 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 interactúan entre sí.
    - *Extensiones (extends):* un CU extiende una funcionalidad de otro CU. Un CU puede tener muchos CU extensiones. Los CU extensiones solo son iniciados por un CU.
    - *Uso o inclusión (uses):* reduce la redundancia entre dos o más CU al combinar los pasos comunes de los CU.
    - *Dependencia (depends):* relaciones 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 funcionalidades de uno o varios actores.

Proceso de modelado:

- Identificar los actores.
- Identificar los CU para los requerimientos.
- Construir el diagrama.
- Realizar los escenarios.

Características importantes

- Un CU debe representar una funcionalidad 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 de uso sin alteraciones.
- Las precondiciones no deben representarse en los cursos alternativos, ya que al ser una precondición no va a ocurrir.
- Los "uses" deben ser accedidos por lo menos desde dos CU.

HISTORIAS DE USUARIO

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

Son utilizadas en las metodologías de desarrollo agiles (ejemplo: XP, SCRUM) para la especificación de requisitos (acompañadas de las discusiones de los usuarios y las pruebas de validación).

Debe ser limitada. 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 más rápidamente 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 HU se situé entre unas 10 horas y un par de semanas.

Debe responder a tres preguntas:

¿Quién se beneficia?
¿Qué se quiere?
¿Cuál es el beneficio?

Esquema: **Como** (rol) **quiero** (algo) **para poder** (beneficio).
Ejemplo: Como usuario registrado quiero loguearme para poder empezar a utilizar la aplicación.
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 misma no es lo suficiente explicita para considerarse un contrato, la discusión con los usuarios debe permitir esclarecer su alcance y este debe dejarse explicito bajo las formas de prueba de validación.
- *Valoradas por los clientes o 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.
- *Estimables:* un resultado de la discusión de una HU es la estimación del tiempo que tomara completarla. Esto permite estimar el tiempo total del proyecto.
- *Pequeñas:* las historias muy largas son difíciles de estimar e imponen restricciones sobre la planificación de un desarrollo iterativo. Generalmente se recomienda la consolidación de historias muy cortas en una sola historia.
- *Verificables:* las HU cubren requerimientos funcionales, por lo que generalmente son verificables. Cuando sea posible, la verificación debe automatizarse, de manera que pueda ser verificable en cada entrega del proyecto.

Criterios 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 la complementan y ayudan al equipo de desarrollo a entender mejor como se espera que el producto se comporte.
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 inicia de la definición del cómo. No están diseñados para ser tan detallados como una especificación de diseño tradicional.
Si una HU 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 pruebas de validación 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.

DIAGRAMA de FLUJO de DATOS (DFD)

Es una herramienta que permite visualizar un sistema como una red de procesos funcionales, conectados entre si por "conductos" y almacenamiento de datos.
Representa 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.

- Procesos: se representan con círculos o burbujas y representan las funciones individuales que ejecuta el sistema. Las funciones transforman entradas en salidas.
- Flujos: se representan con flechas continuas, la información que los procesos necesitan como entrada o producen como salida.
- Almacenamientos: se representan con líneas dobles los datos permanentes del sistema de operación. Al concretarse el diseño dará origen a las bases de datos y archivos.
- Entidades externas o terminadores: muestran productores o consumidores de información que residen fuera de los límites del sistema.

Diccionario de datos:

Listado organizado de todos los datos pertinentes al sistema:

- Definición sin ambigüedad de los datos y elementos del sistema.
- Permite revisar consistencia.
- Representa el contenido de la información.
- Define el significado de los flujos y los almacenes.
- Un dato debe contener: Tipo, nombre y descripción.
- Notación:
    - = está compuesto de
    - + y (secuencia)
    - () optativo
    - {} iteración
    - [] selección de alternativas
    - ‘’ comentarios
    - @ campo clave de archivo
    - | separar opciones

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. Permite que sobreviva a cambios tecnológicos.

- Componentes:

1- *Modelo ambiental:* define las interfaces entre el sistema y el ambiente donde el mismo se ejecuta.

1.1- Declaración de propósito: en forma sintética 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 acontecimientos: 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.

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

Control (C)

La construcción del modelo ambiental es lo primero y más importante en la construcción del modelo de requerimientos del usuario para el nuevo sistema.

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.

2- Modelo de comportamiento:

a. El modelo preliminar de comportamiento contiene:
    i. Un diagrama preliminar de flujos de datos del sistema (DFD)
    ii. Un diagrama preliminar de entidad-relación (ER)
    iii. Una primer versión del diccionario de datos (DD)
    iv. Un diagrama de transición de estados (DTE)
b. 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.
c. Construcción:
    i. Una burbuja o proceso por cada acontecimiento de la lista
    ii. La burbuja se nombra identificando la respuesta del sistema al acontecimiento
    iii. Se dibujan las entradas-salidas y los almacenamientos apropiados para que la burbuja “funcione”.
    iv. Se chequea el borrador de DFD obtenido con el diagrama de contexto y la lista de acontecimientos.

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.

Nivelación de un DFD:

- A partir del DFD preliminar se realizan nivelaciones:
  - Ascendentes: agrupa las burbujas con algún criterio.
    - Tiene una utilidad de presentación al usuario
    - El DFD preliminar tiene un proceso por cada acontecimiento
    - Utilizando el principio de "ocultamiento de la información" agrupa los procesos que acceden al mismo almacenamiento
  - Descendentes: descompone las burbujas funcionalmente.
    - Las burbujas que no tiene más explosiones son las burbujas primitivas

SISTEMAS DE TIEMPO REAL

Características: responden a un mundo real en un tiempo prefijado, deben ser fiables, reinicializables y recuperables a fallas.
Ejemplos: control de procesos, investigación médica, comunicaciones, etc.
En aplicaciones de tiempo real, el sistema debe controlar la información continua en el tiempo generada por algún proceso del mundo real.
La notación del flujo de datos convencional no hace distenciones entre datos discretos y datos continuos en el tiempo.
Una ampliación de la notación básica del análisis estructurado proporciona un mecanismo para representar el flujo de datos continuo en el tiempo.
Para representar el flujo continuo en el tiempo se usa la flecha de dos cabezas, mientras que el flujo de datos discreto se representa con una flecha de una sola cabeza.

DFC

Muchas aplicaciones de software son dependientes del tiempo y procesan más información orientada al control que a los datos, por ej.: control de naves, procesos de fabricación, etc.
Las primeras ampliaciones que se hacen a este método están efectuadas por Ward y Mellor, y posteriormente lo hacen Hatley y Pirbhai y GoldSmith.
Estas ampliaciones permiten reflejar el flujo de control y el procesamiento de control, así como el procesamiento y el flujo de datos.

MODELOS DE PROCESO

Proceso: cuando proveemos un servicio o creamos un producto, siempre seguimos una secuencia de pasos para realizar un conjunto de tareas. Las tareas son realizadas usualmente en el mismo orden. Se puede pensar a un conjunto ordenado de tareas como un proceso.
Proceso de software: es un conjunto de actividades y resultados asociados que producen un producto de software.
Modelo de proceso 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.
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.

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 prescriptivos: prescriben un conjunto de elementos del proceso: actividades del marco de trabajo, acciones de la ing. de software, tareas, aseguramiento de la calidad y mecanismo de control. Cada modelo de proceso prescribe también un "flujo de trabajo", es decir de qué forma los elementos del proceso de interrelacionan entre sí.
Modelos descriptivos: descripción en la forma en que se realizan en la realidad.
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 periodo 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 pareceres, cambios en el ambiente.

Modelo en V: demuestran 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 e/ los lados izquierdo y derecho implica que, si se encuentran problemas durante la verificación y validación, entonces al 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 este 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 construya 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 examinaran los modelos abstractos. Usuarios que no determinaran 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. Debe desarrollarse rápidamente. Equipo de desarrollo reducido. Herramientas y lenguajes adecuados.

Desarrollo por fases: 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.

- Incremental: el sistema es particionado en subsistemas de acuerdo a 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.

Modelo espiral (Boehm): combina las actividades de desarrollo con la gestión del riesgo. Trata de mejorar los ciclos de vida clásico y prototipos. Incorpora objetivos de calidad. Elimina errores y alternativos no atractivas al comienzo. Permite iteraciones, vuelta atrás y finalizaciones rápidas. Cada ciclo se completa con una revisión que incluye todo el ciclo anterior y el plan para el siguiente.

METODOLOGIAS AGILES

"Es un enfoque iterativo e incremental (evolutivo) de desarrollo de software"

Iterativo: es una estrategia de reproceso en la que el tiempo se separa para revisar y mejorar partes del sistema.

Incremental: es una estrategia programada y en etapas, en la que las diferentes partes del sistema se desarrollan en diferentes momentos o a diferentes velocidades, y se integran a medida que se completan.

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.

Una metodología ágil es aquella en la que "se da prioridad a las tareas que dan resultados directos y reducen la burocracia tanto como sea posible" [Fowler], adaptándose además rápidamente al cambio de proyectos.
Ese enfoque ha sido utilizado desde hace más de dos décadas por un grupo de profesionales de software.

"The Agile Alliance" (AA) es una organización dedicada a promover los conceptos de desarrollo de software ágil, y de ayudar a las organizaciones a adoptar dichos conceptos. Estos están resumidos en el manifiesto (redactado por Kent Beck, el padre de XP) para el desarrollo ágil de software y consta de valores y principios.

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.

Principios:

- Nuestra mayor prioridad es satisfacer al cliente a través de fáciles y continuas entregas de software valuable.
- Los cambios de requerimientos son bienvenidos, aun tardíos, en el desarrollo. Los procesos agiles capturan los cambios para que el cliente obtenga ventajas competitivas.
- 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.
- Usuarios y desarrolladores deben trabajar juntos durante todo el proyecto.
- Construir proyectos alrededor de motivaciones individuales.
- Darles el ambiente y el soporte que ellos necesitan y confiar el trabajo dado. El dialogo cara a cara es el método más eficiente y efectivo de intercambiar información entre el equipo de desarrolladores.
- El software que funciona es la medida clave de progreso.
- Los procesos agiles promueven un desarrollo sostenible. Los stakeholders, desarrolladores y usuarios deberían ser capaces de mantener un paso constante indefinidamente.
- Atención continua a la excelencia técnica y buen diseño incrementa la agilidad.
- Simplicidad (el arte de maximizar la cantidad de trabajo no dado) es esencial.
- Las mejores arquitecturas, requerimientos y diseños surgen de la propia organización de los equipos.
- A intervalos regulares, el equipo reflexiona sobre como volverse más efectivo, entonces afina y ajusta su comportamiento en consecuencia.

Principales AMs

XP Extreme Programming:

Los principios básicos de esta programación son simplicidad, comunicación, retroalimentación y coraje.
Las características esenciales son historias de usuario, roles, procesos, practicas.

- Roles:
    - Programador (programmer): responsable de decisiones técnicas y de construir el sistema, sin distinción entre analistas, diseñadores o codificadores. En XP, los programadores diseñan, programan y realizan las pruebas.
    - Jefe de proyecto (manager): organiza y guía las reuniones, asegura condiciones adecuadas para el proyecto.
    - Cliente (customer): es parte del equipo, determina que construir y cuando, estable las pruebas funcionales.
    - Entrenador (coach): responsable del proceso. Tiene a estar en un 2do 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): metric man. Observa sin molestar. Conserva datos históricos.
- Proceso:

  El ciclo de vida consiste en:

    1- Exploración:
        a. Los clientes plantean las historias de usuario que son de interés para la primera entrega del producto.
        b. El equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizaran el proyecto.
        c. Se construye un prototipo.

    2- Planificación:
        a. El cliente estable la prioridad de cada historia de usuario.

b. Los programadores realizan una estimación del esfuerzo.
c. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente.

3- Iteraciones:
   a. El plan de entrega está compuesto por iteraciones de no más de 3 semanas.
   b. El cliente es quien decide que historias se implementaran en cada iteración.
   c. Al final de la última iteración el sistema estará listo para entrar en producción.
4- Producción:
   a. Esta fase requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente.
   b. 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:
   a. Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones.
   b. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.
6- Muerte:
   a. Es cuando el cliente no tiene más historias para ser incluidas en el sistema.
   b. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura.
   c. 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:
  - 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.
  - 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.
  - Programación de a pares: todo el código de producción es escrito por dos programadores en una máquina.
  - 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.
  - 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 o compartido.
  - Semana de 40 horas: se debe trabajar un máximo de 40hs por semana. El trabajo extra desmotiva al equipo. Los proyectos que requieran 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.
  - Cliente en el lugar de desarrollo: el cliente tiene que estar presenta y disponible todo el tiempo para el equipo.
  - 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 conocimientos: 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 mas 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.

Artefactos:

- Product Backlog: es la lista maestra que contiene toda la funcionalidad deseada en el producto. La característica más importante es la 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.

Procesos:

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 iteración, si no que tenemos todas esas etapas juntas que se ejecutan una y otra vez, hasta que se crea suficiente.

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

DESARROLLO DE SOFTWARE BASADO EN MODELOS (MDB)

Hacia fines de los 70’ De Marco introdujo el concepto de desarrollo de software basado en modelos. Destaco 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 actua como una especificación precisa de los requerimientos que el sistema de software debe satisfacer.

Modelo de software:

Un modelo es la descripción de un sistema (o de una parte) en un lenguaje bien definido. Un lenguaje bien definido es un lenguaje con una forma definida (sintaxis) y significado (semántica) que sea apropiado para ser interpretado automáticamente por un computador.

Un modelo se presenta con frecuencia como una combinación de dibujos y texto.

Ciclo de vida:

Requisitos. Análisis. Diseño. Codificación. Testeo. Emplazamiento.

Problemas:

Persisten para asegurar calidad y corrección durante el desarrollo del software.

El problema de la productividad, el mantenimiento y la documentación.
El problema de la flexibilidad a los cambios tecnológicos.

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.
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 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 para convertirse en entidades productivas a partir de las cuales se deriva la implementación en forma automática.

Modelos de MDD (PIMs y PSMs)
Platform Independent Model (PIM): "Un modelo de un sistema que no contiene información acerca de la plataforma o la tecnología que es usada para implementarlo".
Platform Specific Model (PSM): "Un modelo de un sistema que incluye información acerca de la tecnología especifica que se usara para su implementación sobre una plataforma específica".
Transformación de modelos: "Especifica el proceso de conversión de un modelo en otro modelo del mismo sistema".
Cada transformación incluye (al menos): - Un PIM. - Un modelo de plataforma. – Una transformación. – Un PSM.
Transformación: 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).
El patrón MDD es normalmente utilizado sucesivas veces para producir una sucesión de transformaciones.
Orígenes de MDD: MDD es la evolución natural de la ingeniería de software basada en modelos enriquecida mediante el agregado de transformaciones automáticas entre modelos. La técnica de transformación se asemeja al proceso de abstracción y refinamiento presentado por Dijkstra.
Beneficios de MDD: Incremento en la productividad (modelos y transformaciones). Adaptación a los cambios tecnológicos y 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

Es la capacidad de un producto o servicio para servir satisfactoriamente a los propósitos del usuario mediante su utilización, la conformidad con los requisitos explícitos e implícitos de un cliente y la ausencia de defectos e imperfecciones.
Es un concepto manejado con bastante frecuencia en la actualidad. Al hablar de bienes y/o servicios de calidad, la gente se refiere normalmente a bienes de lujo o excelentes con precios elevados.
Su significado sigue siendo ambiguo y muchas veces su uso depende de lo que cada uno entiende por calidad, lo cual es importante comenzar a unificar su definición.
Definiciones de algunos "filósofos de la calidad":

- Crosby: "La calidad es la conformidad de los requerimientos"
- W. Edwards Deming: "Calidad en términos de la satisfacción del cliente"
- Armand V. Feingenbaum: "Calidad como una relación directa entre los productos y servicios, y las necesidades del cliente"
- Kaoru Ishikawa: "Estable que los requerimientos y necesidades van cambiando lo que conlleva a una definición cambiante"

Las normas internaciones definen a 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 la confieren aptitud para satisfacer unas necesidades expresadas o implícitas" (ISO 8402)

Historia de la calidad

Se indica que el control de calidad tuvo el inicio en 1916, cuando las empresas Western Electric y la Bell Telephone unieron esfuerzos e investigaciones para la fabricación de teléfonos que pudiesen resistir con gran fiabilidad el duro uso del público.

En 1946 se creó la American Society for Quality, una centra de divulgación de la información en los temas de control de la calidad.

En 1950 con la ayuda de W. Edwars Deming, Japon adopto una metodología dirigida hacia el usuario fabricando productos de calidad, es decir, "haciendo bien las cosas a la primera". Esta metodología se iniciaba en la alta dirección y luego extendiéndose a todos los niveles de la empresa (Total Quality Management).

Calidad de los sistemas de información

Stylianou y Kumar plantean que se debe apreciar la calidad desde un todo, donde cada parte que la componen debe tener su análisis de calidad.

- Componentes:
    - Calidad de la infraestructura: incluye, por ej, la calidad de las redes y sist. de software.
    - Calidad de software: de las aplicaciones de software construidas, o mantenidas, con el apoyo de IS.
    - Calidad de datos: que ingresan en el sistema de información.
    - Calidad de información: está relacionado con la calidad de los datos.
    - Calidad de gestión: incluye el presupuesto, planificación y programación.
    - Calidad de servicio: incluye los procesos de atención al cliente.

Calidad de software

Definición según Pressman (2002): "la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente"

Se divide en calidad del producto obtenido y la calidad del proceso de desarrollo (son dependientes)

Calidad del producto y proceso

- Producto (Hatton, 1995): un producto es de buena calidad si le sirve a quien lo adquiere y si este lo usa para realizar las tareas para lo que fue concebido. Aunque el software tiene aspectos intangibles, un producto software es sin embargo un bien en sí mismo e incluye sus documentos asociados.
- Proceso: un proceso malo, mal concebido e implementado generara producto de mala calidad. Un proceso bueno, bien concebido e implementado generara la mayor cantidad de veces productos de buena calidad.

Modelos de calidad

Tienen en cuenta criterios para satisfacer las necesidades de los desarrolladores, mantenedores, adquisidores y usuarios finales. Los modelos de calidad pueden ser utilizados para construir mejores productos y asegurar su calidad.

Se han desarrollado varios modelos de calidad para diferentes productos y procesos de software.

- Calidad del producto: ISO/IEC 9126: tecnologías de la información – calidad de los productos de software es un estándar internacional el cual clasifica la calidad de software en un conjunto estructurado de características de la siguiente manera.

  Más recientemente ha aparecido la norma ISO25000 software Engineering – Software Product Quality Requirements and Evaluation (SQuaRE)

  SQuaRE cubre tres procesos de calidad complementarios: especificación de requisitos, medidas y evaluación. Se creó para satisfacer necesidades que existían con la ISO 9126 y la ISO/IEC 14598, (primera generación de estándares de calidad de un producto de software). Por consiguiente, SQuaRE pertenece a la segunda generación de normas de calidad de un producto de software.
- Calidad de Proceso:
    - CMM (Capability Maturity Model)/CMMI:
        - Desarrollado por SEI (software Engineering Institute), organismo creado por el DoD de USA.
        - Fuerte impacto en mejora del proceso.
        - Estipula un camino para la mejora.
        - Establece Áreas Clave que se deben atacar.
    - ISO 12207 – Modelos de Ciclos de Vida del Software

        - Actividades que debe incluir.
    - ISO 15504 – Modelo de Evaluación
    - ISO 90003 guía para la aplicación de la norma ISO 9001 al desarrollo, suministro y mantenimiento de software.

CMM – CMMI

- Historia: 1985 SEI empieza a trabajar en un marco de madurez de procesos que permita evaluar a las empresas productoras de software. La investigación evoluciona hacia el "Modelo de madurez de las capacidades (CMM)"
  1991 en agosto SEI publica la versión 1.0 del modelo de madurez de las capacidades para el software (SW – CMM, Capability, Maturity Model for Software).
  1993 SEI publica la versión 1.1 de SW – CMM.
  1997 publicación de la versión 1.2.

En diciembre de 2000, el SEI publico un nuevo modelo, el CMMI o "Modelo de Capacidad y Madurez – Integracion", con el objetivo de realizar algunas mejoras respecto al SW-CMM (e integrarlo con SE-CMM y el IPD-CMM, que pasaron a ser considerados como "obsoletos").
Incluye cuatro disciplinas, software, Ingenieria de Sistemas, Desarrollo integrado de procesos y productos y Gestion de proveedores.
A su vez incorpora una nueva representación, "Continua", la que permita evaluar el nivel en cada área independientemente.
El SEI ha desarrollado también un nuevo método de evaluación de las organizaciones según CMMI denominado SCAMPI.

CMM

Posee una sola forma de aplicación (escalonada) que permite un enfoque dirigido por niveles de madurez que indican como se desempeña una organización en base a la madurez en el área de proceso. Presenta 5 (cinco) niveles de madurez.

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

CMM – CMMI Escalonado

- Nivel 1 – Inicial: desempeño basado en la competencia personal.
    - Frecuentemente la organización vive "apagando incendios".
    - Aparecen héroes.
    - Dificultad para encarar mejoras a largo plazo.
    - La organización actúa esencialmente por reacción.

  Entran los requerimientos y otras entradas y salen los productos.
- Nivel 2 – Repetible: la organización.
    - Estableció la gestión de los proyectos de software, y está documentado.
    - Usa políticas organizacionales.
    - Repite prácticas exitosas desarrolladas en proyectos previos.
    - Información (recursos, tiempo, esfuerzo) compartida por medios informales.

  Existen riesgos al presentarse nuevos desafíos.
- Nivel 3 – Definido:
  El proceso para la gestión y las actividades de ingeniería está documentado e integrado en un proceso estándar para la organización. Todos los proyectos usan una versión documentada y aprobada del proceso estándar de la organización. La información del proceso se halla estandarizada y es compartida a través de la BDD de proceso de software.
- Nivel 4 – Gestionado:

La organización aplica los principios de la gestión estadística de procesos para controlar el proceso del software. La dirección tiene bases objetivas para tomar decisiones. Puede predecir el desempeño en un entorno cuantificado realista. Usa los datos como base para decisiones, objetivos y mejoras.

- Nivel 5 – Optimizado: la organización
    - Identifica y elimina causas del desempeño pobre
    - Mejora continua del proceso en base a gestión del cambio del proceso y de la tecnología.

CMMI Representación continúa

Nivel 0 – Incompleto. Nivel 1 – Ejecutado. Nivel 2 – Administrado. Nivel 3 – Definido. Nivel 4 – Administrado Cuantitativamente. Nivel 5 – Optimizado.

ISO (International Organization for Standardization)

La familia ISO 9000 es un conjunto de normas de "gestión de la calidad" aplicables a cualquier tipo de organización (empresa de producción, empresa de servicios, administración pública, etc.) 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.

La familia de normas apareció por primera vez en 1987 teniendo como base una norma estándar británica (BS), y se extendió principalmente a partir de su versión de 1994, estando actualmente en su versión 2008.

Las normas ISO 9000 de 1994 estaban principalmente pensadas para organizaciones que realizaban proceso productivo, y por tanto su implantación en empresas de servicios era muy dura y por eso se sigue en la creencia de que es un sistema bastante burocrático.

Con la última revisión se ha conseguido una norma bastante menos burocrática para organizaciones de todo tipo, y además se puede aplicar sin problemas en empresas de servicios e incluso en la Administración Publica.

ISO 90003

- Objetivo:
    - Proveer las especificaciones de cómo aplicar la ISO 9001 al desarrollo de software, implementación y mantenimiento.
    - Se incluyen temas como administración de la configuración y planeamiento de proyectos.

Es requerida por las empresas desarrolladoras de software para:

- Poder incursionar en la competencia del mercado europeo
- Cubrir expectativas del cliente
- Obtener ventajas competitivas
- Reducir costos

Otras normas

- Moprosoft (Norma Mexicana)
- MPS (Norma Brasilera)
- Competisoft
- Metrica V3
- ISO 29110