Ingeniería de software 1

Clase 1

Introducción

El ***software*** son instrucciones, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

Se caracteriza por ser un elemento lógica el cual se desarrolla y no se desgasta.

Tipos de productos de software

- Genéricos: sistemas aislados producidos por organizaciones desarrolladoras de software y que se venden en un mercado abierto. Hay cada vez más sistemas construidos por un producto genérico como base que luego se adapta según lo necesario para el cliente.
- Personalizados: sistemas requeridos por un cliente particular.

Clasificación del software

1. De sistemas: aquel que sirve como plataforma o interfaz entre el *hardware* de la computadora y los programas de aplicación. Su función principal es gestionar y controlar los recursos del sistema (memoria, procesador, dispositivos I/O).
2. De aplicación: programas diseñados para realizar tareas específicas y directamente útiles para el usuario final.
3. Científico y de ingeniería: diseñado para resolver problemas complejos que requieren un alto nivel de cálculo numérico y modelado. Se utiliza principalmente en investigación, desarrollo y diseño.
4. Empotrado: reside en la memoria de un producto (a menudo un sistema basado en microprocesador) cuyo propósito principal no es necesariamente el cálculo informático. Controla funciones y proporciona interfaces para el producto.
5. De línea de productos: sistemas diseñados para proporcionar una capacidad específica para su uso por parte de muchos clientes diferentes, en lugar de para un solo cliente o proyecto. Están diseñados para ser escalables y parametrizables.
6. Aplicaciones web: sistemas que utilizan **navegadores** como interfaz de usuario y que residen en un servidor remoto. Ofrecen servicios o funcionalidades a través de Internet.
7. De inteligencia artificial: usa algoritmos no numéricos para abordar problemas complejos que no son fácilmente resolubles con la computación tradicional, como la inferencia y el reconocimiento de patrones. Imita las capacidades cognitivas humanas.

Ingeniería de software

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

- *Disciplina de la ingeniería:* hace que las cosas funcionen, aplicando teorías, métodos y herramientas.
- *Todos los aspectos de la producción:* no sólo comprende los procesos técnicos del desarrollo de software, sino también se realizan actividades como la gestión de proyectos y el desarrollo de herramientas, métodos y teorías de apoyo a la producción de software.

Según la IEEE define a la ingeniería de software como:

> El uso de métodos sistemáticos, disciplinados y cuantificables para el desarrollo, operación y mantenimiento de software y el estudio de dichas técnicas.

- Usa métodos sistemáticos cuantificables: la cuantificación rigurosa 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 de medir, estimar, planificar y administrar proyectos.
- Para el desarrollo, operación y mantenimiento: la ingeniería 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.

Historia de la IS

La OTAM dio espacio a la IS con el objeto de identificar los problemas de la industria del software y sentar bases de procesos, de esta forma nació el concepto.

- Era funcional (60s): estudia cómo explorar la tecnología para hacer frente a las necesidades funcionales de las organizaciones.
- Era de control (70s): aparece la necesidad de desarrollar software en tiempo, planeado y controlado. Introduce el modelo de ciclo de vida en fases.
- Era de costos (80s): la importancia de la productividad en el desarrollo de software se incrementa sustancialmente. Se ponen en práctica varios modelos de costos.
- Era de calidad (90s y actual): se identifica la cantidad de que el producto tenga atributos que satisfagan las necesidades explícitas e implícitas del usuario: mantenibilidad, confiabilidad, eficiencia y usabilidad.

Conocimientos de un IS

El ingeniero debe conocer las tecnologías y productos de sistemas operativos, lenguajes de programación, bases de datos, bibliotecas, etc.

Además también conocer técnicas de administración de proyectos como planificación, análisis de riesgos, control de calidad, seguimiento de proyectos, etc.

Ya que la IS se desarrolla en un marco económico, social y legal, deberá haber cierta ***responsabilidad profesional y ética*** y no usar su capacidad y habilidades deshonestamente o deshonrando la profesión.

Responsabilidad profesional y ética

- Confidencialidad: respetar la confidencialidad de sus empleados y clientes.
- Competencia: no falsificar el nivel de competencia y aceptar responsabilidades fuera de su capacidad.
- Derecho de la propiedad intelectual: conocer las leyes vigentes sobre las patentes y copyright.
- Uso inapropiado de computadoras: no usar habilidades técnicas para actividades inapropiadas.

Técnicas de comunicación

A la hora de iniciar un proyecto habrá que saber lo que el usuario *quiere*, *cómo lo quiere, cuándo* y *por qué,* por lo tanto será necesario la **comunicación.**

La cual es la base para la obtención de las necesidades del cliente donde es el lugar con mayores errores.

Cuando hablamos de necesidades, nos referimos a **requerimientos.**

Requerimientos

Característica del sistema o descripción de algo que es capaz de hacer para satisfacer el propósito del sistema.

Hay 3 principales fuentes para obtener los requerimientos

- Documentación.
- Stakeholders.
- Especificación de sistemas similares.

Stakeholder

Se refiere a cualquier persona o grupo que se verá afectado por el sistema, directa o indirectamente.

Hay varios stakeholders; usuarios finales, ingenieros, gerentes, expertos del dominio, diferentes visiones.

Hay 3 tipos de puntos de vista los cuales son

- **Interactuadores**: personas o sistemas que interactúan directamente son el sistema y que pueden influir en los requerimientos del sistema de algún modo.
- **Indirecto**: stakeholders que no usan el sistema pero influyen en los requerimientos.
- **Dominio**: características y restricciones del dominio que influyen en los requerimientos.

A veces la identificación puede ser difícil, hay algunas formas de obtenerlos de manera más específica:

- Proveedores o receptores de servicios de sistema.
- Regulaciones y estándares a aplicar.
- Fuentes de requerimiento.
- Puntos de vista de personas que lo desarrollan, administran y mantienen.
- Puntos de vista de marketing y otros que generarán requerimientos.

Elicitación de requerimientos

Un requerimiento es una característica del sistema que es capaz de hacer para satisfacer su propósito.

La ***elicitación de requerimientos*** es el proceso de adquirir todo el conocimiento relevante para producir un modelo de requerimientos de un dominio de problema.

Es necesario conocer el dominio del problema para comunicarse con clientes y usuarios, además de conocer el sistema actual, identificar necesidades y las expectativas sobre el sistema a desarrollar.

Podrán surgir problemas en la comunicación los cuales serán

- 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.
- Conflictos personales o políticos.

- Limitaciones cognitivas
  - No conocer el dominio del problema.
  - Hacer suposiciones sobre el dominio del problema.
  - Hacer suposiciones sobre aspectos tecnológicos.
  - Hacer simplificaciones excesivas.
- Conducta humana
  - Conficlos y ambigüedades en los roles de los participantes.
  - Pasividad de cliente, 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.

Recopilación de información

Métodos discretos

Los métodos discretos son aquellos los cuales son menos perturbadores que otras formas de averiguar requerimientos.

Se consideran insuficientes para recopilar información cuando se utilizan por sí solos, por lo que deben utilizarse junto con uno o varios de los métodos.

Utilizar diferentes métodos para acercarse a la organización es una práctica inteligente mediante la cual podrá formarse un panorama más completo de los requerimientos, estos métodos serán:

- Muestreo de la documentación, formularios y datos existentes: se observarán organigramas, notas, minutas, registros contables, documentación de proyectos anteriores.
- Investigación y visitas al lugar: se investigará el dominio, se observará patrones de soluciones, consulta a 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.

  Habrá ciertos lineamientos para la observación como determinar quién y cuándo será observado, obtener permiso, mantener bajo perfil, tomar notas de lo observado, no interrumpir a las personas trabajando.

  Lo positivo es que habrá datos confiables ya que se observa exactamente lo que se hace, hay un análisis de las disposiciones del lugar y resulta ser más económica a comparación de otras técnicas.

  Por otro lado, las desventajas principales será que las personas se sienten incómodas siendo observadas, algunas actividades del sistema pueden ser realizadas en horarios incómodos, las tareas podrían interrumpirse.

Métodos interactivos

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 y lo principal es hablar y escuchar para comprender a las personas de la organización.

- Cuestionarios: documento que permite al analista recabar información y opiniones de los encuestados. Recolecta hechos de un gran número de personas para detectar sentimientos generalizados y problemas.

  Tendrá respuesta rápida, es económico, anónimo y estructurado para fácil análisis pero se obtienen bajo número de respuestas, no responden a todas las preguntas, son rígidas, no hay un análisis corporal y las respuestas pueden ser incompletas.

  Habrá dos tipos de preguntas

  - Abiertas: aquellas que dejan abiertas todas las posibles opciones de respuestas
  - Cerradas: limitan o cierran las opciones de respuestas disponibles.

  A su vez habrá distintos tipos de información obtenida

  - Actitud: lo que las personas dicen qué quieren.
  - Creencias: lo que las personas creen que es verdad.
  - Comportamiento: lo que realmente hacen.
  - Características: de las personas o cosas.

  Es recomendable usar cuestionarios cuando las personas están dispersas geográficamente, hay muchas involucradas, quiero opiniones generales o identificar un problema general.

- 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.

  Se obtendrá información respecto a opiniones, objetivos, procedimientos informales y sentimientos.

  Las ventajas son que el entrevistado se siente inluído en el proyecto además de obtener una retroalimentación del encuestado, se pueden adaptar las preguntas según el entrevistado y se obtiene información no verbal del observado mediante acciones y expresiones.

  Las contras es que son costosas, conlleva tiempo y RRHH, la eficiencia de las entrevistas dependen de las habilidades del entrevistador y no pueden ser aplicadas a distancia.

  Dentro de las entrevistas hay dos tipos

  - Estructuradas/ cerradas: el encuestador tiene un conjunto específico de preguntas para hacérselas al entrevistado, se dirige al usuario por un requerimiento puntal y no permite adquirir un amplio conocimiento del dominio.
  - No estructuradas/ abiertas: el encuestador lleva a un tema en general, no hay preparación de preguntas específicas y inicia con preguntas que no dependen del contexto.

  Además hay distintos tipos de preguntas

  - Abiertas: permite al encuestado responder de cualquier manera. Lo positivo es que revelan una nueva línea de preguntas volviendo la entrevista más interesante y permite espontaneidad pero puede haber muchos detalles irrelevantes perdiendo el control de la entrevista y haciendo parecer que el entrevistador no tiene los objetivos claros.
  - Cerradas: las respuestas son directas, cortas o de sección específica. Lo positivo es que ahorran tiempo manteniendo fácil el control de la entrevista y brinda datos relevantes pero lo malo es que puede aburrir al encuestado o no brindar detalles.
  - Sondeo: permite obtener más detalle sobre un tema puntual.

  La entrevista además podrá tener distintas organizaciones

  - Piramidal: inicia con preguntas cerradas y se convierten en abiertas.
  - Embudo: inicia con preguntas abiertas y se dirige a preguntas cerradas.

- Diamante: inicia con preguntas cerradas, pasa a abiertas y vuelve a cerradas.

Hay ciertos lineamientos a seguir para una entrevista eficiente, llamado ***preparación previa (Kendall)***

Se recomienda leer los antecedentes para poner atención al lenguaje y buscar vocabulario en común ya que es imprescindible para poder entender al entrevistado. Además se debe establecer los objetivos de la entrevista.

Habrá que seleccionar a los entrevistados para minimizar el número de entrevistas y se deben elegir aquellos que sepan el objetivo de la entrevista y las preguntas que se le harán.

Es necesario también establecer fecha, hora, lugar y duración de cada entrevista según el entrevistado como también elegir el tipo de preguntas a usar y su estructura.

Whitten propone cómo conducir la entrevista:

Según el requerimiento a analizar, hay que conocer sus fortalezas, prejuicios y motivaciones, hacer una cita, respetar el horario de trabajo, establecer la duración de la entrevista, obtener el permiso del supervisor o jefe y llevarse a cabo en un lugar privado.

Para preparar la entrevista hay que informar al entrevistado del tema a tratar antes de la reunión, definir un guion de entrevista, evitando preguntas sesgadas o con intención, amenazantes o críticas, usar lenguaje claro y conciso, no incluir opinión como parte de la pregunta y evitar preguntas largas y complejas.

En el momento de la entrevista, hay que respetar el horario y tiempos definidos, iniciar la entrevista saludando y presentándose, mencionar el propósito de la misma, escuchar con atención y observando y finalmente, concluya la entrevista con una conclusión y agradecimiento.

Debe	Evite
Vestirse adecuadamente	Suponer que una respuesta no lleva a ningún lado
Ser cortés	Revelar pistas
Escuchar cuidadosamente	Usar jerga
Mantener el control	Revelar sesgos personales
Observar los gestos	Hablar en lugar de escuchar
Ser paciente	Suponer cualquier cosa acerca del tema o del entrevistado
Mantener al entrevistado en calma	Usar grabadores
Mantener el autocontrol	
Terminar a tiempo	

Se sugiere enviarle al entrevistado un resumen de la entrevista, permitiendo aclarar cualquier cosa que no se haya entendido.

Es clave saber estuchar, ya que es la parte más importante del proceso.

- Planeación conjunta de requerimientos (JRP): es un proceso mediante el cual se conducen reuniones de grupo altamente estructurados con el propósito de analizar problemas y definir requerimientos.

  Es necesario un extenso entrenamiento, lo positivo es que redue el tiempo de exploración de requerimientos además de existir una amplia participación de los integrantes y se va trabajando sobre lo que se va generando.

  Lo positivo es que se ahorra tiempo en la etapa de requerimientos, hay usuarios involucrados y desarrollos creativos pero por otro lado es difícil organizar los horarios y es complejo encontrar un grupo de participantes integrados y organizados.

  A la hora de planear una JRP hay que seleccionar un lugar para llevar a cabo las sesiones como también a los integrantes y la agenda

- Lluvia de ideas/ brainstorming: es una 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 hacen reuniones del equipo involucrado en la resolución del problema, las cuales son conducidas por un director.

Hay una serie de etapas las cuales son descubrir hechos, producir ideas y descubrir soluciones.

Será clave para resolver la falta de consenso entre usuarios y es útil combinarlo con la toma de decisiones además de ayudar a entender el dominio del problema y enfrenta la dificultad del usuario para transmitir ideas.

Clase 2

Proceso de software

En este proceso las actividades fundamentales será la especificación del software, desarrollarlo, validarlo y evolucionarlo, lo cual todo es tarea de los IS.

Modelo de software

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.

La mayoría de los modelos de proceso de software se basan en uno de los siguientes modelos o paradigmas

- Modelo en cascada: las etapas se representan cayendo en cascada. Cada etapa de desarrollo se debe completar antes que comience la siguiente.
- Desarrollo interactivo: un sistema inicial se desarrolla rápidamente a partir de una especificación abstracta. Éste se refina basándose en las peticiones del cliente.
- Desarrollo basado en componentes: esta técnica supone que las partes ya existen por lo tanto el proceso se enfoca en integrarlas.

Impacto de los errores en la etapa de requerimientos

El software resultante puede no satisfacer a los usuario debido a las múltiples interpretaciones de los requerimientos lo cual genera desacuerdos entre clientes y desarrolladores.

Podrá gastarse tiempo y dinero construyendo el sistema erróneo.

Tipos de requerimientos

- **Funcionales:** describen una interacción entre el sistema y su ambiente, describiendo cómo debe comportarse el sistema ante determinado estímulo. Además de describir lo que el sistema *debe hacer* o no.
- **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.
    - Requerimientos organizacionales: se derivan de las políticas y procedimientos existentes en la organización del cliente y en la del desarrollador.
    - Requerimientos externos: interoperabilidad, legales, privacidad, seguridad, éticos.
    - Requerimientos del dominio: reflejan las características, reglas y restricciones propias del área de negocio o el sector de aplicación para el cual se está desarrollando el sistema. Son el conocimiento inherente de ese campo particular. Se derivan de las leyes de la física, las normas contables, los procedimientos médicos, las regulaciones aeronáuticas, o cualquier otro estándar o conocimiento especializado del dominio (por ejemplo, banca, salud, logística, educación).

- Requerimientos por prioridad: se utiliza para gestionar el desarrollo en entornos con recursos limitados (tiempo, presupuesto), ayudando a los equipos a decidir qué implementar primero.
- 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 o conjunción de requerimientos.
- Requerimientos del sistema: establecen con detalle los servicios y restricciones del sistema. Es difícil excluir toda la información de diseño.

Ingeniería de requerimientos

Disciplina donde se transforma requerimientos declarados por los clientes a especificaciones precisas, no ambiguas, consistentes y completas del comportamiento del sistema, incluyendo funciones, interfaces, rendimiento y limitaciones.

Será utilizado para acuerdos comunes entre todas las partes involucradas y en donde se describen las funciones que realizará el sistema.

También es el proceso mediante el cual se intercambian diferentes puntos de vista para recopilar y modelar lo que el sistema hará. Combina métodos, herramientas y actores.

Es útil porque permite gestionar las necesidades del proyecto estructuradamente, mejora la capacidad de predecir cronogramas de proyectos, disminuye costos y retrasos, mejora la calidad del software como también la comunicación y evita rechazos de usuarios finales.

Estudio de viabilidad

Se usa en sistemas nuevos donde a partir de una descripción del sistema se elabora un informe que recomienda la conveniencia o no de hacer el desarrollo.

Una vez que se ha compilado toda la información necesaria, se debe hablar con las fuentes de información para responder nuevas preguntas y luego se redacta el informe.

Especificación de requerimientos

Las *propiedades* de los requerimientos principalmente será

- Necesario: 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.

Los *objetivos* son

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

Habrá ciertos *documentos* a completar

- Documento de definición de requerimientos: listado completo de todas las cosas que el cliente espera que haga el sistema propuesto.
- Documento de especificación de requerimientos: definición de términos técnicos.

- Documento de especificación de requerimientos de software: para brindar una colección de buenas prácticas para escribir especificaciones de requerimientos de software.

Aspectos básicos de una especificación de requerimientos

- Funcionalidad: qué debe hacer el software.
- Interfaces externas: cómo interactúa el software con el medio externo.
- Rendimiento: velocidad, disponibilidad, tiempo de respuesta, etc.
- Atributos: portabilidad, seguridad, mantenibilidad, eficiencia.
- Restricciones de diseño: estándares requeridos, lenguaje, límite de recursos.

Validación de requerimientos

Proceso de certificar la corrección del modelo de requerimientos contra las intenciones del usuario.

Trata de demostrar que los requerimientos definidos son los que estipula el sistema. Se describe el ambiente en el que debe operar el sistema. Es importante, porque los errores en los requerimientos pueden conducir a grandes costos si se descubren más tarde.

Validación: al final del desarrollo evaluar el software para asegurar que el software cumple los requerimientos. Solamente podrá hacerse con la activa participación del usuario

Verificación: el software cumple los requerimientos correctamente.

Se evidenció que no es suficiente solamente con validar después del desarrollo de software ya que cuanto más se tarde, más costará en corregir generando una bola de nieve de defectos. Por lo tanto, es necesario validar en la fase de especificación de requerimientos, ayudando a evitar costosas correcciones luego del desarrollo.

A la hora de verificar los requerimientos es necesario

- Verificaciones de validez.
- Verificaciones de consistencia, para que no haya contradicciones.
- Verificaciones de completitud de todos los requerimientos.
- Verificaciones de realismo para estar seguros que se pueden implementar.
- Verificabilidad de que pueden diseñarse conjunto de pruebas.

Las verificaciones podrán ser manuales o automatizadas

- Revisiones de requerimientos
  - Informales: los desarrolladores deben tratar los requerimientos con tantos stakeholders como sea posible.
  - Formal: el equipo de desarrollo debe conducir al cliente, explicándole las implicaciones de cada requerimiento. Es conveniente hacerla luego de la revisión informal.
- Construcción de prototipos.
- Generación de casos de pruebas.

Técnicas de especificación de requerimientos

- Estáticas: se describe el sistema a través de las entidades u objetos, sus atributos y relaciones con otros. No describe cómo las relaciones cambian con el tiempo.
- Dinámicas: se considera un sistema en función de los cambios que ocurren a lo largo del tiempo. Considera que el sistema está en un estado particular hasta que un estímulo lo obliga a cambiar su estado.

Historias de usuario

Representación de un requisito de software escrito en una o dos frases usando el lenguaje común del usuario.

Usadas en las metodologías ágiles como SCRUM o XP para especificar requerimientos y se acompaña con discusiones de los usuarios y pruebas de validación.

- Deberá ser **limitada**, por lo que debería poder escribirse sobre una nota adhesiva pequeña.
- **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 los requisitos cambiantes.**
- Al momento de implementar las historias, los desarrolladores deben tener la posibilidad de **discutirlo con los clientes.**
- Generalmente se espera que la **estimación de tiempo** de cada historia de usuario se sitúe entre unas **10 horas y un par de semanas.** Estimaciones mayores a 2 semanas es indicativo de que la historia es muy compleja y debe ser dividida en más.

Si bien el estilo puede ser libre, la historia debe responder a tres preguntas

1. A quién beneficia
2. Qué se quiere.
3. Cuál es el beneficio.

> *Como **usuario registrado** deseo l**oguearme** para **poder empezar a utilizar la aplicación.***

Características

- Ser independientes unas de las otras: de ser necesario, combinar las historias dependientes o buscar la forma que resulten independientes.
- Negociables: la historia en sí misma no es suficientemente explítica como para considerarse un contrato por lo que la discusión de los usuarios debe permitir esclarecer su alcance y éste debe dejarse explícito bajo la forma de pruebas 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 H.U es la estimación de tiempo que tomará 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 H.U 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 al expectativa del PM o PO.

Deben ser definidos durante la etapa inicial antes de la codificación, acompañan a la H.U porque complementan la historia de usuario y ayudan al equipo a entender mejor cómo se espera que el producto se comporte.

Los criterios de aceptación son usados para expresar el resultado de las conversaciones del cliente con el desarrollador.

Si una H.U tiene más de 4 criterios de aceptación, debe evaluarse subdividir la historia además de poder añadirse un número de escenario para identificar al criterio asociado.

Beneficios

- Al ser muy corta, representa requisitos del modelo de negocio que pueden implementarse rápidamente.
- Necesitan poco mantenimiento.
- Mantienen una relación cercada con el cliente.
- Permite dividir los proyectos en pequeñas entregas.
- Permite estimar fácilmente el esfuerzo de desarrollo.
- Ideal para proyectos con requisitos volátiles o poco claros.

Limitaciones

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

Épicas

Conjunto de historias de usuario que se agrupan por algún denominador común.

Clase 3

Casos de uso

Procesos de modelado de las funcionalidades del sistema en término de los eventos que interactúan entre los usuarios y el sistema.

Tienen sus orígenes en el modelado orientado a objetos 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 C.U 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 requisitos.

Componentes

- Diagrama de casos de uso: ilustra las interacciones entre el sistema y los actores. Representa un objetivo 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.

- Escenarios: descripción de la interacción entre el actor y el sistema para realizar la funcionalidad.
- Actores: inicia una actividad en el sistema. Representa a un usuario que interactúa. Puede ser una persona, sistema externo o dispositivo externo que dispare un evento.
- Relaciones
  - Asociaciones: relación entre un actor y un CU en el que interactúan entre sí.
  - Extensiones: un CU extiende la funcionalidad de otro CU. Un CU puede tener muchos CU extensiones pero los CU extensiones sólo 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 funcionalidad de uno o varios actores.

Proceso de modelado

1. Identificar a los actores: se buscarán los potenciales en documentación, minutas de reuniones o documentos de requerimientos. El objetivo es saber quién o qué proporciona las entradas al sistema, sus salidas, interfaces requeridas, la información.

   Se nombran mediante sustantivos o frase sustantiva.
2. Identificar los CU para los requerimientos: las principales preguntas a responder es cuáles son las tareas del actor, información necesaria, qué información proporciona el actor del sistema, necesita el sistema informar eventos o cambios y viceversa.
3. Construir el diagrama.
4. Realizar los escenarios.

Clase 4

Diagramas de transición de estados

// agregar otra info, clase incompleta

Clase 5

Redes de Petri

Inventadas por Carl Petri con el objeto de 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 o procesos, en varios procesadores o intercalados en un solo procesador.

Las tareas concurrentes deben estar sincronizadas para permitir la comunicación entre ellas. Pueden realizarse varias en paralelo, pero son ejecutadas en un orden impredecible.

- Los arcos indican a través de una flecha la relación entre sitios y transiciones y viceversa.
- A los lugares se les asignan tokens/ fichas que se representan mediante un número o puntos dentro del sitio.
- 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 pueden provocar que los tokens “viajen” cuando se cumplen las condiciones adecuadas.
- La ejecución es controlada por el número y distribución de los tokens.

La ejecución de una red de Petri se realiza disparando transiciones habilitadas, que lo estará cuando cada lugar de entrada tiene al menos tantos tokens como arcos hacia la transición.

Disparar una transición habilitada implica remover tokens desde la entrada hacia la salida.

Sincronización

Para que varios procesos colaboren en la solución de un problema es necesario que compartan información y recursos pero esto debe ser controlado para asegurar la integridad y correcta operación del sistema.

Características

- Es importante desarrollar modelos de los sistemas de eventos discretos para estudiarlos y comprender su comportamiento.
- Existe 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 redes 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.

Clase 6

Tabla de decisión

Técnica de especificación de requerimientos *dinámica* la cual es una herramienta que permite presentar de forma concisa las reglas lógicas que hay que utilizar para decidir acciones a ejecutar en función de las condiciones y la lógica de decisión de un problema específico.

Describe al sistema como un conjunto de posibles *condiciones* satisfechas por el sistema en un momento dado junto con *reglas* para reaccionar ante los estímulos que ocurren cuando se reúnen determinados conjuntos de condiciones y *acciones* a ser tomadas como un resultado.

A la hora de construir la tabla, se harán condiciones y acciones simples donde solo toman V o F. Hay 2^N reglas donde N es la cantidad de condiciones.

A partir del enunciado se debe

1. Identificar las condiciones y las acciones
2. Completar la tabla teniendo en cuenta
    a. Si hay condiciones opuestas, debe colocarse una de ellas porque por la negativa se “obtendrá” la otra.
    b. Las condiciones deben ser atómicas.
3. Se construyen las reglas

Ejemplo

Modelizar el problema de remisión de mercadería con las siguientes consideraciones

- Si el comprador no es cliente se imprime un mensaje de aviso y no se remite.
- Si no hay stock y el comprador es cliente, no se remite.

- Si hay stock y el comprador es cliente, se remite.

1. Identificación de condiciones y acciones

   Condiciones

Es cliente	V	V	F	F
Hay stock	V	F	V	F

   Acciones

Imprime mensaje de aviso			X	X
Se remite	X			
No se remite		X	X	X

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

Es esencial que verifique la integridad y precisión de sus tablas de decisión.

Clase 7

Análisis estructurado

Para entender los requerimientos, se debe poder reconocer además cómo se mueven los datos, los procesos o transformaciones que sufren dichos datos y sus resultados.

La elicitación proporciona una descripción verbal del sistema, una descripción visual puede consolidar la información.

La técnica de análisis estructurado permite lograr una representación gráfica que permite lograr una comprensión más profunda del sistema a construir y comunicar a los usuarios lo comprendido.

La notación no especifica aspectos físicos de implementación.

Hace énfasis en el procesamiento o la transformación de datos conforme estos pasan por distintos procesos.

Modelado de funciones del sistema

Diagrama de flujo de estados (DFD)

Herramienta que permite visualizar un sistema como una red de procesos funcionales, conectados entre sí por “conductos” y almacenamiento de datos.

Representa la transformación de entradas a salidas y es también llamado *diagrama de burbujas*.

Comúnmente usado por sistemas operacionales en los cuales las funciones del sistema son de gran importancia y son más complejas por los datos que éste maneja.

- Se usa un rectángulo para representar una **entidad externa**, esto es por ejemplo un elemento hardware, una persona u otro programa.
- Un círculo representa un **proceso** o **transformación** que es aplicado a los datos.
- Una flecha representa uno o más **elementos de datos**.
- Un rectángulo abierto representa un **almacén de datos.**

Para desarrollar un DFDs se debe visualizar desde una perspectiva jerárquica de arriba hacia abajo

1. Redactar la lista de actividades de la organización para determinar
    a. Entidades externas.
    b. Flujos de datos.
    c. Procesos.
    d. Almacenes de datos.
2. Crear un diagrama de contexto que muestre las entidades externas y los flujos de datos desde y hacia el sistema.
3. Dibujar el diagrama, con procesos generales y los almacenes correspondientes.
4. Dibujar un diagrama hijo por cada uno de los procesos del diagrama 0.

Modelos de 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.

Un **proceso de software** es un conjunto de actividades y resultados asociados que producen un producto de software.

Modelo de proceso de software: marco de referencia que contiene los procesos, actividades y tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso.

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 de procesos de software: representación abstracta de un proceso del software.

- **Modelos prescriptivos:** prescriben un conjunto de elementos del proceso: actividades del marco de trabajo, acciones de la ingeniería de 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.

  Ambos modelos deberían ser iguales.

- **Modelos tradicionales:** formados por un conjunto de fases o actividades en las que no tienen en cuenta la naturaleza evolutiva del software

- Clásico, lineal o en cascada: las etapas se representan cayendo en cascada donde cada una se debe completar antes que comience la siguiente.

  Es una técnica que resulta útil para diagramar lo necesario a hacer y su simplicidad hace que sea fácil explicarlo a clientes.

  Aparecen ciertas dificultades en este modelo las cuales son

  1. No existen resultados concretos hasta que esté todo terminado.
  2. Las fallas más triviales se encuentran al comienzo del período de prueba y las más graves al final.
  3. La eliminación de fallas puede ser extremadamente difícil durante las últimas etapas de prueba del sistema.
  4. Deriva del mundo del hardware y presenta una visión de manufactura sobre el desarrollo de software.
  5. La necesidad de pruebas aumenta exponencialmente durante las etapas finales.
  6. *Congelar* una fase es poco realista.
  7. Hay errores, cambios de parecer o del ambiente.
- 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.
- Basado en 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 para el producto terminado.

  Esta es una alternativa de especificación para tratar mejor la incertidumbre y ambigüedad de los proyectos reales.

- **Modelos 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 propone como solución. Resultan ser modelos descartables ya que no tienen funcionalidad.
  - En espiral: combina las actividades de desarrollo con la gestión del riesgo tratando de mejorar los ciclos de vida clásicos y prototipos incorporando objetivos de calidad y eliminando errores y alternativas no atractivas al comienzo.

    Permite iteraciones, vuelta atrás y finalizaciones rápidas.

    Cada ciclo empieza identificando los objetivos de la porción correspondiente y cada ciclo se completa con una
  - Evolutivo.
  - Incremental.
- **Modelo de 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 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.

Clase 8

Metodologías ágiles

El éxito de un desarrollo está dado por la metodología empleada la cual no 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 riesgos.

En este contexto, las metodologías ágiles emergen como una posible solución.

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

- *Enfoque iterativo:* el desarrollo iterativo es una estrategia de reproceso en la que el tiempo se separa para revisar y mejorar partes del sistemas.
- *Incremental:* 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.

El objetivo de las **metodologías ágiles** es producir software de alta calidad con un costo efectivo y en el tiempo apropiado.

Como también 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.

Además de 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.

Valores

- Más 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

- La mayor prioridad es satisfacer al cliente a través de fáciles y continuas entregas de software valuable.
- Los cambios de requerimientos son bienvenidos pero hace que el desarrollo sea tardío. Por lo tanto, los procesos ágiles 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 diálogo 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 del progreso.

- Los procesos ágiles promueven un desarrollo sostenible. Los stakeholders, desarrolladores y usuarios deberían ser capaces de mantener un paso constante indefinidamente.
- La atención continua a la excelencia técnica y buen diseño incrementa la agilidad.
- Simplicidad es esencial.
- las mejores arquitecturas, requerimientos y diseños surgen de la propia organización de los equipos.
- A intervalos regulares, el equipo reflexiona sobre cómo volverse más efectivo, entonces refina y ajusta su comportamiento en consecuencia.

Ágil vs no ágil

Ágil	No ágil
Pocos artefactos	Más artefactos
Pocos roles	Más roles
No existe un contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (< 10 integrantes) y trabajando en el mismo sitio	Grupos grandes
Menos énfasis en la arquitectura	La arquitectura es esencial

Desventajas

En la práctica, los principios que subyacen a los métodos ágiles son a veces difícil 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 de 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 de ingeniería de software se usa en el mantenimiento y la evolución de los sistemas de software existentes. Al no existir documentación se complejizaría.

Principales metodologías ágiles

- XP extreme programming.
- SCRUM.

- DSDM (Dynamic Systems development method).
- Crystal methods (Cockburn's crystal family methodologies).
- ASD adaptative software development.
- FDD feature-driven development.

Extreme programming

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.

Las características esenciales serán

- Historias de usuario.
- Roles.
- Proceso.
- Prácticas.

Y los roles

- Programador: es el responsable de las decisiones técnicas y de construir el sistema donde no hay distinción entre analistas, diseñadores y codificadores. En XP los programadores hacen todas las tareas.
- Jefe de proyecto: organiza y guía las reuniones asegurando condiciones adecuadas para el proyecto.
- Cliente: parte del equipo quien determina qué construir y cuándo, estableciendo las pruebas funcionales. Las pruebas funcionales son un tipo de prueba de software que se centra en verificar que cada función del sistema de software se comporta de acuerdo con las especificaciones de requerimientos del cliente.
- Entrenador: responsable del proceso que tiende que estar a segundo plano a medida que el equipo madura.
- Encargado de pruebas: ayuda al cliente con las pruebas funcionales. y asegura que las pruebas se superen.
- Rastreador: metric man que observa sin molestar y conserva datos históricos.

El ciclo de vida en XP se basa en

1. Exploración: los clientes plantean las H.U que son de interés para la primera entrega del producto, el equipo se familiariza con las herramientas, tecnologías y prácticas que se usarán en el proyecto y así se construirá un prototipo.

   Esta fase toma pocas semanas o meses, según tamaño y familiaridad de los desarrolladores.

2. Planificación: el cliente establece la prioridad de cada H.U, se estima el esfuerzo de los programadores y se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente.

   Fase que dura un par de días

3. Iteraciones: el plan de entrega está compuesto por iteraciones de no más de 3 semanas donde el cliente es quien decide qué historias se implementarán en cada iteración y al final de la última iteración el sistema estará listo para entrar en producción.
4. Producción: fase que requiere pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Además, se toman decisiones sobre la inclusión de nuevas características a la versión actual.

5. Mantenimiento: mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que se desarrollan nuevas iteraciones. Esta fase puede llegar a requerir nuevo personal dentro del equipo y cambios de estructura.
6. Muerte: 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 de 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.

Además, las prácticas a seguir en XP serán

- Testing: los programadores continuamente escriben pruebas unitarias, las cuales deben correr sin problemas para que el desarrollo continúe.
- Refactoring: actividad constante de reestructuración del código con el objetivo de remover duplicación de código, legibilidad, simplificarlo y hacerlo más flexible para facilitar 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. Esto 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 40 horas por semana ya que el trabajo extra desmotiva al equipo. Los proyectos que requieren trabajo extra para intentar cumplir con los platos 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 lugar de desarrollo: el cliente quiere estar presente y disponible todo el tiempo para el equipi.
- Estándares de codificación: los programadores escriben todo el código de acuerdo con reglas que enfatizan la comunicación a través del mismo.

SCRUM

Scrum es un proceso al 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 elección tiene origen en un estudio de la manera de trabajar en 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.

Tiene sus principios como

- **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 ya que a medida que va pasando el tiempo se obtiene más información.
- **Entregar rápido:** debe ser una de las ventajas competitivas.
- **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.

Además tiene los siguientes roles

- **Product owner**: conoce y marca las prioridades del proyecto o producto.
- **Scrum master:** 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.
- **Scrum team:** son las personas responsables de implementar la funcionalidad o funcionalidades elegidas por el P.O.
- **Usuarios/ clientes:** son los beneficiarios finales del producto, y son quienes viendo los progresos, pueden aportar ideas, sugerencias o necesidades.
- **Product backlog:** lista maestra que contiene toda la funcionalidad deseada en el producto. La característica más importante es que la funcionalidad tiene un orden de prioridad.
- **Sprint backlog:** lista que contiene toda la funcionalidad que el equipo se comprometió a desarrollar durante un sprint determinado.
- **Burndown chart:** muestra un acumulativo de trabajo hecho.

Sigue el siguiente proceso

El ***product backlog*** corresponde con todas las tareas, funcionalidades o requerimientos a realizar. El ***sprint backlog*** corresponde con una o más tareas que provienen del product backlog. Es decir, del product backlog se saca una o más tareas que van a formar parte del sprint backlog.

La ***daily scrum meeting*** es una tarea iterativa que se realiza todos los días que dure el sprint backlog con el equipo de desarrollo.

Scrum resulta ser ***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 en cascada por cada iteración, si no que tenemos todas estas etapas juntas que se ejecutan una y otra vez, hasta que se crea suficiente.

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

Desarrollo de software basado en modelos

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.

Desarrollo de software dirigido por modelos.

El adjetivo *modelo* en MDD, a diferencia de *basado,* enfatiza que ese 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.

En el paradigma MDD, los modelos (como los diagramas UML) ya no son solo dibujos o documentación, sino que se convierten en el **motor principal y tangible** de la construcción del *software.*

MDD es la evolución natural de la ingeniería de software basada en modelos enriquecida mediante el agregado de transformaciones automáticas entre modelos.

Si bien MDD define un nuevo paradigma para el desarrollo de software, sus principios fundamentales no constituyen realmente nuevas ideas, sino que son reformulaciones y asociaciones de ideas anteriores.

La técnica de transformación se asemeja al proceso de abstracción y refinamiento presentado por Dijkstra.

MDD tiene los siguientes beneficios

- Incremento en la productividad.
- Adaptación a los cambios tecnológicos.
- Adaptación a los cambios de requisitos.
- Consistencia.
- Re-uso.
- Mejoras en la comunicación con los usuarios y la comunicación entre los desarrolladores.
- Captura de la experiencia.
- Los modelos son productos de larga duración.
- Posibilidad de remorar decisiones tecnológicas.

Clase 9

La calidad es un concepto manejando con bastante frecuencia en la actualidad, pero a su vez, su significado es percibido de distintas maneras.

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, por lo cual es importante comenzar a unificar su definición.

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

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

La *Organización Internacional de Normalización* también conocida como *ISO* es una organización para la creación de estándares internacionales compuesta por diversas organizaciones nacionales de normalización.

Sistemas de información

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

Un sistema de información abarca más que el aspecto meramente computacional, pues no sólo hemos de tener en cuenta estas herramientas, sino también el modo de organizar dichas herramientas y de obtener la información necesaria para el correcto funcionamiento de la empresa.

La importancia de los sistemas de información en la actualidad hace necesario que las empresas de tecnología hagan mucho hincapié en los estándares de calidad.

Plantean que se debe apreciar la calidad desde un todo, donde cada parte que la componen debe tener su análisis de calidad.

Calidad de software

La calidad de software es el grado en el que un sistema, componente o proceso cumple con los requerimientos especificados y las necesidades o expectativas del cliente/ usuario. La industria ha evolucionado al reconocer que la calidad no se logra solo probando el producto, sino integrándola en todo el ciclo de desarrollo.

La calidad del software se divide y depende de 2 grandes pilares interconectados:

1. Calidad del producto obtenido: se centra en las propiedades intrínsecas del software resultante. Respondiendo a qué tan bien funciona. Tiene como estándar de referencia a ISO/ IEC 25000 que define las características y métricas de la calidad del producto.
2. Calidad del proceso de desarrollo: se centra en la manera en que se construye el software (métodos, herramientas, prácticas). Respondiendo a la pregunta de qué tan bien construimos el software. Se basa en ISO/ IEC 122207/ 15504 que establecen modelos para la gestión y evaluación de los procesos del ciclo de vida.

Es casi imposible obtener un buen producto sin un buen proceso de desarrollo por lo tanto están relacionados ambos.

SQuaRE (Software Quality Requeriments and Evaluation) es un conjunto de normas internacionales para la evaluación de la calidad de los productos de software. Su propósito es unificar y organizar los diferentes modelos y conceptos de calidad en un marco coherente.

Estándares y modelos de gestión de calidad de software

Abarca las herramientas claves que permiten a una organización asegurar la calidad tanto del proceso como de la gestión del proyecto.

- **PMBOOK (Project management body of knowledge):** guía exhaustiva que documenta las mejores prácticas para la gestión de proyectos, incluyendo la inicialización, planificación, ejecución, seguimiento y cierre.
- **SWEBOK (software engineering body of knowledge):** guía que define el conjunto de conocimientos esenciales que se espera que posea un ingeniero de software.
- **SIX SIGMA:** metodología de mejora de procesos centrada en la reducción de la variación y la eliminación de defectos.
- **MPS-BR:** mejora de procesos de software brasileño. Es un modelo de madurez desarrollado en Brasil con el objetivo de evaluar y mejorar la calidad de los procesos de software en organizaciones brasileñas, a menudo visto como una alternativa a CMMMI más adaptada al contexto local.
- **MOPROSOFT:** modelo de procesos para la industria del software mexicano. Fomenta la homologación y estandarización de la gestión de procesos de software en las organizaciones de IT en México.
- COMPETISOFT: iniciativa para la mejora de procesos de software en Iberoamérica, especialmente en pymes.
- **Métrica V3:** metodología de planificación y desarrollo de sistemas de información (España). Es una metodología integral desarrollada y utilizada en la Administración Pública Española para planificar, desarrollar y mantener los sistemas de información.

ISO/ IEC 25000 (SQuaRE)

La norma ISO/IEC 25000 define un modelo de calidad del producto, clasificando sus atributos en varias características primarias

- Funcionalidad: el grado en que el software proporciona funciones que satisfacen necesidades.
- Confiabilidad: la capacidad del software de mantener su nivel de rendimiento bajo condiciones especificadas.
- Eficiencia: el rendimiento relativo a la cantidad de recursos utilizados (tiempo de respuesta, uso de memoria, etc).
- Facilidad de uso: la facilidad con la que los usuarios pueden aprender a usar el software y operarlo.
- Mantenibilidad: la facilidad con la que el software puede ser modificado (corregir defectos, adaptarse a nuevos entornos, mejorar el rendimiento).
- Portabilidad: la capacidad del software de ser transferido de un entrono de hardware o software a otro.
- Seguridad: la capacidad de proteger la información y los datos para que personas o sistemas no autorizados no puedan leerlos o modificarlos.
- Compatibilidad: la capacidad de dos o más sistemas o componentes para intercambiar información y utilizar las mismas funciones.

ISO/IEC 25000

Enseña que la calidad del software es una evaluación holística que debe considerar

1. Calidad de producto de software: este es el aspecto más conocido y se centra en las propiedades estáticas y dinámicas del software cuando está en funcionamiento.
2. Calidad de uso: se refiere al impacto del producto de software cuando es usado por usuarios específicos en un entorno y contexto de uso particular. Mide la satisfacción del usuario y la efectividad del sistema para lograr objetivos.
3. Calidad de datos: reconoce que, para muchas aplicaciones, la calidad de la información es tan importante como la del código.

ISO/IEC 20000- Calidad de servicios

Especifica los requisitos para que una organización establezca, implemente, mantenga y mejore continuamente un sistema de gestión de servicios (SGS). Es el equivalente a la ISO 9001, pero enfocada en la gestión de servicios de tecnologías de la información.

El objetivo es asegurar que los servicios de IT sean gestionados efectiva, eficiente y consistentemente para satisfacer los requerimientos del negocio y las necesidades del cliente.

ISO/ IEC 12207- Modelo del ciclo de vida del software

Proporciona un conjunto de procesos comunes para el ciclo de vida del software (adquisición, suministro, desarrollo, operación y mantenimiento),

ISO/IEC 15504

*SPICE (*software process improvement and capability determination) es un estándar para la evaluación de la capacidad y madurez de los procesos de desarrollo.

ISO/IEC 90003

Es una guía para al aplicación de ISO 9001 al software. Proporciona directrices para interpretar y aplicar los requisitos de la norma general ISO 9001 al software.

ISO/ IEC 29110

Estándar para ciclos de vida de software para pequeñas entidades, ofrece perfiles y guías adaptadas para equipos de desarrollo pequeños menores a 25 personas.

ISO/ IEC 27001- Seguridad de la información

El objetivo principal de esta norma es ayudar a las organizaciones a proteger la **confidencialidad, integridad y disponibilidad (CIA)** de su información, sin importar el formato en que se encuentre (digital, papel, oral, etc.).

- Exige que los procesos de desarrollo (de la **ISO/IEC 12207**) incorporen **seguridad desde el diseño** (*Security by Design*).
- Garantiza que el **entorno de desarrollo** (servidores, repositorios de código, acceso a bases de datos) esté protegido.
- Asegura que las políticas de la organización cubran la protección de la **propiedad intelectual** (código fuente) y los **datos del cliente**.

ISO/IEC 25000

Calidad del producto

- Portabilidad.
- Seguridad.
- Facilidad de mantenimiento.
- Compatibilidad.
- Funcionalidad.
- Confiabilidad.
- Facilidad de uso.
- Eficiencia.

Mejora la calidad de los procesos software

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

Modelo de evaluación: ISO/ IEC 15505: 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)

Estos modelos ayudan a las organizaciones a evaluar su proceso y definir una hoja de ruta para la mejora continua.

El CMMI (Capability Maturity Model Integration) es un marco de referencia que ayuda a las organizaciones a mejorar sus procesos

- CMM (modelo de madurez de capacidad): enfocado originalmente en proceso de desarrollo de software. Introdujo el concepto de niveles de madurez para evaluar la organización.
- CMMI (integración del modelo de madurez de capacidad): evoluciona para integrar diferentes disciplinas (desarrollo, servicios, adquisiciones, etc). Posee dos representaciones: escalonada y continua.

CMMI

Posee dos vistas que permiten un enfoque diferentes según las necesidades de quien vaya a implementarlo.

- Escalonado: centra su foco en la madurez de la organización, igual que CMM. Los niveles de madurez del 1 al 5 se deben alcanzar secuencialmente.
- Continuo: se enfoca en los procesos individuales y hay 6 niveles del 0 a 5 de capacidad, que indican qué tan bien se desempeña la organización en un área de proceso específica, permitiendo mejoras selectivas.

1. Inicial: proceso impredecible, poco controlado y reactivo.
2. Gestionado: los procesos individuales están caracterizados y planificados.
3. Definido: los procesos están estandarizados a nivel de la organización y son proactivos.
4. Gestionado cuantitativamente: el proceso es medido y controlado utilizando métricas y técnicas estadísticas.
5. Optimizado: enfoque en la mejora continua del proceso a través de la innovación tecnológica.

ISO- International Organization for Standardization

La Internacional Organization of Standardization (ISO) proporciona estándares de gestión, no solo de software, que buscan consistencia y mejora.

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.

Familia de las ISO 9000

- ISO- 9001:2015: es la norma central de la familia. Especifica los requisitos para un sistema de gestión de la calidad que puede ser aplicado a cualquier organización. Su objetivo es asegurar la consistencia y la mejora continua del negocio.
- IRAM- ISO 9001:2015: sistema de gestión de la calidad- requisitos. Norma publicada por ISO y traducida por IRAM.
- ISO 90003:2004: directrices para el software. Es una guía que ayuda a interpretar los requisitos de la ISO 9001 en el contexto específico de los procesos de desarrollo, suministro y mantenimiento del software.

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

Implementar un SGC (ISO 9001) ayuda a la organización a cumplir con requisitos legales y del cliente, aumentar el rendimiento operacional, optimizar operaciones y mejorar la satisfacción del cliente.

SGC- IRAM- ISO 90001

Combina la implementación local de un estándar de calidad con el principio fundamental de toda la gestión de calidad.

- ISO 9001 es la norma global que define los requisitos para cualquier SGC.
- IRAM (Instituto Argentino de Normalización y Certificación): es la entidad argentina que traduce, adopta y certifica las normas ISO en el país.

La ISO 9001, implementada a través de IRAM, obliga a la organización a tener un **SGC** que no solo funcione bien hoy, sino que tenga mecanismos formales y permanentes para **mejorar continuamente** su desempeño.