brunocascio / **requerimientos-tecnicas-agiles.md**

Last active Jul 17, 2020

Requerimientos, Técnicas de Especificación y metodologías Ágiles

Ingeniería de Software I (Resúmen)

- Requerimientos
  - Introducción
  - Ingeniería de requerimientos
    - Introducción
    - Estudio de viabilidad
    - Obtención y análisis de requerimientos
    - Especificación de requerimientos
    - Validación de requerimientos
  - Gestión de Requerimientos
- Técnicas de especificación de requerimientos.
  - Introducción
  - Estáticas
  - Dinámicas
    - Tablas de decisión (TDD)
    - Diagrama de Transición de Estados (DTE)
    - Redes de Petri (RP)
    - Casos de Uso (CU)
    - Historia de Usuario
    - Diagrama de Flujo de Datos (DFD)
      - Diccionario de datos (DD)
    - Diagrama de Flujo de Control (DFC)
- Metodologías Ágiles
  - Extreme Programming (XP)
  - SCRUM
  - Desarrollo de Software Basado en Modelos (MBD)
  - Desarrollo de Software Dirigido por Modelos (MDD)

--> Requerimientos

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

Al iniciar un proyecto, ¿cuál es la primera actividad? Tenemos que comunicarnos.

- Saber lo *que* el **usuario** quiere, *cómo lo quiere, cuándo y porqué.*

Al hablar de necesidades, en términos más técnicos, estamos hablando de **requerimientos**.

La comunicación es la base para la obtención de los requerimientos, pero es también la principal fuente de error:

- Falta de procedimientos y guías formales.
- Falta de participación del usuario.

- Mala interpretación de las necesidades.
- Falta de comunicación.

Ingeniería de Requerimientos

La **ingeniería de requerimientos** es el proceso de transformación de los requerimientos de los clientes, ya sean hablados o escritos, en especificaciones precisas, no ambiguas consistentes y completas. También se encarga de establecer y mantener los cambios de requerimientos entre clientes y el equipo. La importancia de esta disciplina:

- Permite mantener una estructura en las necesidades del proyecto.
- Disminuyes costos y retrasos.
- Mejora la calidad del software.
- Mejora la comunicación.
- Evita rechazos de usuarios finales.

Estudio de viabilidad

A partir de una descripción resumida del sistema se elabora un informe que recomienda la conveniencia o no de realizar el proceso de desarrollo. Responde a las siguientes preguntas:

- ¿El sistema contribuye a los objetivos generales de la organización?
- ¿El sistema se puede implementar con la tecnología actual?
- ¿El sistema se puede implementar con las restricciones de costo y tiempo?
- ¿El sistema puede integrarse a otros que existen en la organización?

Una vez que se ha recopilado toda la información necesaria para contestar las preguntas anteriores se debería hablar con las fuentes de información para responder nuevas preguntas y luego se redacta el informe, donde debería hacerse una recomendación sobre si debe continuar o no el desarrollo.

Obtención y análisis de requerimientos

- Tipos de requerimientos:
  - **Requerimientos funcionales:**
    - Describen una interacción entre el sistema y su ambiente. Cómo debe comportarse el sistema ante determinado estímulo.
    - Describen lo que el sistema debe hacer, o incluso cómo NO debe comportarse.
    - Describen con detalle la funcionalidad del mismo.
    - Son independientes de la implementación de la solución.
    - Se pueden expresar de distintas formas
  - **Requerimientos no funcionales:**
    - Describen una restricción sobre el sistema que limita nuestras elecciones en la construcción de una solución al problema.
    - *Requerimientos del producto*: Especifican el comportamiento del producto (usabilidad, eficiencia, rendimiento, espacio, fiabilidad, portabilidad).
    - *Requerimientos organizacionales*: Se derivan de las políticas y procedimientos existentes en la organización del cliente y en la del desarrollador (entrega, implementación, estándares).
    - *Requerimientos externos*: Interoperabilidad, legales, privacidad, seguridad, éticos.
  - **Otras Clasificaciones**:
    - *Requerimientos del dominio:*
      - Reflejan las características y restricciones del dominio de la aplicación del sistema.
      - Pueden ser funcionales o no funcionales y pueden restringir a los anteriores.
      - Como se especializan en el dominio son complicados de interpretar.
    - *Requerimientos por Prioridad:*
      - Que deben ser absolutamente satisfechos
      - Que son deseables pero no indispensables
      - Que son posibles, pero que podrían eliminarse
    - *Requerimientos del Usuario:*
      - Son declaraciones en lenguaje natural y en diagramas de los servicios que se espera que el sistema provea y

de las restricciones bajo las cuales debe operar.
        - Pueden surgir problemas por falta de claridad, confusión de requerimientos, conjunción de requerimientos.
    - *Requerimientos del Sistema:*
        - Establecen con detalle los servicios y restricciones del sistema.
        - Es difícil excluir toda la información de diseño (arquitectura inicial, interoperabilidad con sistemas existentes, etc.)

Especificación de requerimientos

En la espeicificación de requerimientos se permite a los desarrolladores que expliquen como han entendido lo que el cliente pretende del sistema, asi como también indicar a los diseñadores las funcionalides y características del sistema resultante. Finalmente mostrar al equipo de testing que demostraciones llevar a cabo para convencer al cliente de que el sistema es lo que había pedido.

Para docuemntar todo esto, se utliza el SRS, en donde se describen los contenidos y cualidades de una buena especificación de requerimientos.

Los aspectos básicos de una especificación de requerimientos son:

- Funcionalidad: Lo que debe hacer
- Intefaces Externas: Como interactúa el sistema con el usuario, la arquitectura y otros sistemas.
- Rendimiento: Velocidad, disponibilidad, etc
- Atributos: Portabilidad, seguridad, mantenibilidad, eficiencia.
- Restricciones de diseño: Estándares, lenguaje, límite de recursos, etc.

Valicación de requerimientos

Es el proceso que certifica que los requerimientos definidos son los que hace el sistema.

- **Validación**:
    - Al final del desarrollo evaluar el software para asegurar que el software cumple los requerimientos.
    - La validación sólo se puede hacer con la activa participación del usuario.
    - La validación implica hacer el software correcto.
- **Verificación**:
    - Determinar si un producto de software de una fase cumple los requerimientos de la fase anterior.
    - Verificación de validez: Para todos los usuarios
    - Verificación de consistencia: Sin contradicciones.
    - Verificación de completitud: Todos los requerimientos.
    - Verificación de realismo: Se pueden implementar.
    - Verificabilidad: Se puede diseñar un conjunto de pruebas.
    - La verificación implica hacer el software correctamente.

Técnicas de valicación

Pueden ser manuales o automatizadas.

- Revisiones de requerimientos (formales o informales).
- Construcción de prototipos.
- Generación de casos de prueba.

Revisión de Requerimientos

Es un proceso manual que involucra a distintas personas. Ellos verifican el documento de requerimientos en cuanto a anomalías y omisiones.

- 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
    - Antes de una revisión formal, es conveniente realizar una revisión informal.

¿Es suficiente validar después del desarrollo del software?

- La evidencia estadística dice que NO
- Cuanto más tarde se detecta, más cuesta corregir (Boehm)
- Bola de nieve de defectos
- Validar en la fase de especificación de requerimientos puede ayudar a evitar costosas correcciones después del desarrollo

¿Contra qué se verifican los requerimientos?

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

Gestión de Requerimientos

¿Por qué cambian los requerimientos?

- Porque al analizar el problema, no se hacen las preguntas correctas a las personas correctas (En sistemas grandes hay una comunidad diversa de usuarios)
- Porque los clientes y los usuarios son distintos
- Porque cambió el problema que se estaba resolviendo
- Porque los usuarios cambiaron su forma de pensar o sus percepciones
- Porque cambió el ambiente de negocios (mercado, etc.)
- etc

Los requerimientos pueden ser:

- **Durareros:** Son relativamente estables, se derivan de la actividad principal de la organización.
- **Volátiles:** Cambian durante el desarrollo del sistema o después que se puso en operación (EJ: cambios gubernamentales)
  - Requerimientos Cambiantes: Cambian porque cambia el entorno.
  - Requerimientos Emergentes: Surgen como ampliación.
  - Requerimientos Consencuentes: Surgen por la introducción del sistema. Pueden implicar cambios en la forma de trabajo.
  - Requerimientos de Compatibilidad: Cambian porque interactúan con otros sistemas que cambian.

Para la gestión de los requerimientos se debe hacer:

1. identificación de requerimientos
2. Gestión del cambio * Análisis del problema y especificación del cambio. * Análisis del cambio y cálculo de costos. * Implementación del cambio.
3. Políticas de rastreo * Fuente - Requerimiento - Diseño
4. Ayuda de herramientas CASE * Almacenar requerimientos * Gestionar el cambio * Gestionar el rastreo

--> Técnicas de especificación de requerimientos.

Las técnicas de requerimientos se dividen en *estáticas* y *dinámicas*.

Estáticas

Se describe el sistema a través de las entidades, sus atributos y sus relaciones con otros. NO describe como las relaciones cambian con el tiempo. Esta descripción es útil y adecuada cuando el timepo no es un factor relevante. Por ejemplo:

- Referencia indirecta: Se define QUË se hace, no CÓMO.
- Definición axiomática: Se generan teoremas a partir del comportamiento del sistema y se demuestran.
- Expresiones regulares: Se define un alfabeto y las combinaciones permitadas. El sistema permite aceptar o no una cadena.
- Abstracciones de datos: Para aquellos sistemas en los que los datos determinan las clases de acciones que se realizan (importa para qué son)
- Etc.

Dinámicas

Se considdera un sistema en función de los cambios que ocurren a lo largo del tiempo. Se considera que un sistema está en un estado particular, hasta que un estímulo lo obliga a cambiar su estado. Por ejemplo:

- Tablas de decisión (TDD)
- Diagramas de transición de estados (DTE)
- Redes de Petri (RP)
- Etc

Tablas de decisión

Es un herramienta que mediante reglas lógicas nos ayuda a decidir que acciones a ejecutar en función de las condiciones.

Describe el sistema como un conjunto de:

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

	Reglas				
Es Cliente		V	V	F	F
Hay Stock		V	F	V	F
Imprime mensaje		-	-	X	X
Se remite		X	-	-	-
No Se remite		-	X	X	X

Condiciones | *Acciones*

El resultado de la tabla pueede dar *especificaciones* de tipo:

- **Completas:** Aquellas que determinan una o varias acciones para todas las reglas
- **Contradictorias:** Aquellas que para reglas que determinan mismas condiciones, dan acciones distintas.
- **Redundantes:** Aquellas que para reglas que determinan mismas condiciones, dan acciones iguales.
    - Para solucionar esto se combinan las reglas en donde sea evitdente que un alternativa no representa una diferencia en el resultado.

      En el ejemplo de arriba:

	Reglas			
Es Cliente		V	V	F
Hay Stock		V	F	_
Imprime mensaje		-	-	X
Se remite		X	-	-
No Se remite		-	X	X

Diagrama de Transición de Estados (DTE)

El diagrama de transición de estados se realiza con *Máquinas de estados finito*, esto es, máquinas que describen un sistema como un conjunto de estados donde el sistema reacciones a ciertos eventos ya sean internos o externos.

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

- Σ es un alfabeto;
- S un conjunto de estados;
- T es la función de transición;

- s es el estado inicial;
- A es un conjunto de estados de aceptación o finales.

Para construir este diagrama se siguen los siguientes pasos:

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

Redes de Petri (RP)

Utilizadas para especificar sistemas de tiempo real en los que son necesarios representar aspectos de concurrencia. Los sistemas concurrentes se diseñan para permitir la ejecución simultánea de componentes de programación, llamadas tareas o procesos, en varios procesadores o intercalados en un solo procesador. Las tareas concurrentes deben estar **sincronizadas** para permitir la comunicación entre ellas (por el hecho de que no todas se realizan a la misma velocidad). Deben prevenir bloqueos y/o modificación de datos compartidos.

Pueden realizarse varias tareas en *paralelo*, pero son ejecutados en un orden impredecible. Éstas **NO** son secuenciales, las RP son asincrónicas y el orden en que ocurren los eventos es uno de los permitidos. La ejecución es NO DETERMINÍSTICA. Se acepta que el disparo de una transición es instantáneo.

Los **eventos (o acciones)** se representan como **transiciones (T)**.

Los **estados (o condiciones)** se representan como **lugares o sitios (P)**.

Los arcos indican, a través de una flecha, la relación entre sitios y transiciones, y viceversa. A los sitios se les asignan tokens que se representan mediante un número o puntos dentro del sitio. Esta asignación de tokens a sitios constituye la *marcación*. Luego de una marcación inicial se puede simular la ejecución de la red. El número de tokens asignados a un sitio es ilimitado.

El conjunto de tokens asociado a cada estado sirve para manejar la coordinación de eventos y estados. Una vez que ocurre un evento, un token puede “viajar” de uno de los estados a otro. Las reglas de disparo provocan que los tokens “viajen” de un lugar a otro cuando se cumplen las condiciones adecuadas. La ejecución es controlada por el número y distribución de los tokens.

La ejecución de una Red de Petri se realiza disparando transiciones habilitadas. Una transición está habilitada cuando cada lugar de entrada tiene al menos tantos tokens como arcos hacia la transición.

Disparar una transición habilitada implica remover tokens de los lugares de entrada y distribuir tokens en los lugares de salida (teniendo en cuenta la cantidad de arcos que llegan y la cantidad de arcos que salen de la transición).

Casos de Uso (CU)

Proceso de modelado de las “funcionalidades” del sistema en término de los eventos que interactúan entre los usuarios y el sistema. El uso de CU facilita y alienta la participación de los usuarios. Tiene diferentes beneficios, entre ellos:

- Herramienta para capturar requerimientos **funcionales**.
- Descompone el alcance del sistema en piezas más manejables.
- Medio de comunicación con los usuarios (de fácil lenguaje).
- 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.

Los *diagramas* de CU, permiten ilustrar las interacciones entre el sistema y los actores. Los *escenarios* que forman parte de la narración del CU, permiten describir la intereacción entre el actor y el sistema para realizar la funcionalidad.

Algunos conceptos importantes son:

- 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 sin alteraciones.
- Las pre-condiciones no deben representarse en los cursos alternativos, ya que al ser una pre-condición no va a ocurrir.
- Los “uses” deben ser accedidos por lo menos desde dos CU.

Los CU tienen diferentes elementos que lo conforman, entre ellos:

- **Caso de uso**
  - Representa un objetivo (functionalidad) individual del sistema y describe la secuencia de actividades y de interacciones para alcanzarlo.
  - Paara que el mismo sea considerado un requerimiento debe estar acompañado de su escenario.
- **Actores**
  - Es el que inicia la actividad (CU) en el sistema.
  - 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:
    - Un CU extiende la funcionalidad de otro CU
    - Un CU puede tener muchos CU extensiones
    - 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 funcionalidades de uno o varios actores
- **Escenario** (narración del CU)
  - Conceptos generales
  - Descripción de la interacción del Escenario
  - Descripción de eventos alternativos

El proceso de modelado de un CU se da de la siguiente forma:

1. Identificar los actores. * Deben nombrarse con un sustantivo o frase sustantiva. * Responden a:
   - ¿Quién o qué proporciona las entradas al sistema?
   - ¿Quién o qué recibe las salidas del sistema?
   - ¿Se requieren interfaces con otros sistemas?
   - ¿Quien mantendrá la información en el sistema? Dónde buscar actores potenciales: * Documentación o manuales existentes
   - Minutas de reunión
   - Documentos de requerimientos
2. Identificar los CU para los requerimientos. * Responder a
   - ¿Cuáles son las principales tareas del actor?
   - ¿Qué información necesita el actor del sistema?
   - ¿Qué información proporciona el actor al sistema?
   - ¿Necesita el sistema informar al actor de eventos o cambios ocurridos?
   - ¿Necesita el actor informar al sistema de eventos o cambios ocurridos?
3. Construir el diagrama.
4. Realizar los escenarios.

Historias de Usuario

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

- Utilizadas en las metodologías de desarrollo ágiles (XP, SCRUM, etc)

- Se espera que la estimación de cada historia se sitúe entre unas 10 hroas y un par de semanas.

El esquema de una historia de usuario es:

Como **rol** quiero **algo** para poder **beneficio**.

Ejemplo:

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

Las historias de usuario son:

- Independientes una de Otras
  - Dividir y conquistar
- Negociables
  - La discución con los usuarios debe permitir esclarecer su alcance y este quedar explícito bajo las pruebas de validación.
- Valoradas por clientes o usuarios
  - Cada historia debe ser importante para alguno de ellos más que para el desarrollador.
- Estimables
  - Un resultado de la discusión de una historia de usuario es la estimación del tiempo que tomará completarla. Esto permite estimar el tiempo total del proyecto.
- Pequeñas
  - Generalmente se recomienda la consolidación de historias muy cortas en una sola historia.
- Verificables
  - Las historias de usuario cubren requerimientos funcionales, por lo que generalmente son verificables. Cuando sea posible, la verificación debe automatizarse, de manera que pueda ser verificada en cada entrega del proyecto.

Los beneficios son:

- Permite dividir los proyectos en pequeñas entregas.
- Es ideal para proyectos con requisitos volátiles o no muy claros.

Las limitaciones son:

- 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 sí por “conductos” y almacenamientos de datos. Representa la transformación de entradas a salidas. Es una herramienta comúnmente utilizada donde las funciones del sistema son de gran importancia y son más complejas que los datos que éste maneja.

Los procesos** se representan con *círculos* y son las funciones que convierten las entradas en salidas. Los **flujos** se represenan con *flechas continuas* la información que necesitan como entrada para producir la salida. Los **almacenamientos** se representan con *líneas dobles* los datos permanenetes del sistema en operación. Esto da el origen de la base de datos. Las **entidades externes o terminadores** muestran productores o consumidores de información que residen fuera de los límites del sistema. Se representan en formas de cajas.

Diccionario de datos (DD)

Es un listado organizado de todos los datos pertinentes al sistema. Representan el contenido de la información y defne el significado de los flujos y los almacenes. Cada datos contiene:

- Tipos
- Nombre
- Descripción

Modelo Esencial

Debe indicarse lo que el sistema debe hacer para satisfacer los requerimientos del usuario, con una mínima (en lo posible nula) explicación de cómo lo hace. Evitar el detalle de cualquier restricción o aspecto derivado de la implementación. Pensar el modelo esencial "suponiendo que se dispone de tecnología perfecta".

Componentes:

1. **Modelo Ambiental**

```
Define las interfaces entre el sistema y el ambiente donde el mismo se ejecuta.
La construcción de un modelo ambiental es lo primero y más importante en la construcción del modelo de
requerimientos del usuario para el nuevo sistema.
A medida que encaramos un proyecto mayor, hay cientos de flujos, decenas de terminadores y la lista
de acontecimientos crece y es difícil de manejarla.

  * *DECLARACIÓN DE PROPÓSITO*
    * En forma sintéitca debe indicarse el objetivo del sistema, de que es responsable el sistema
  * *DIAGRAMA DE CONTEXTO*
    * Es un caso especial de DFD, donde se representa todo en una sola burbuja, vinculando almacenes y
entidades externas.
  * *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
        * Temporales (T): comienzan con la llegada de un momento dado en el tiempo.
        * Control (C).
```

2. **Modelo de comportamiento**

```
Los modelos de comportamiento se utilizan para describir el comportamiento del sistema en su totalidad.

* DFD
* DER
* DD
* DTE

La construcción conlleva a:
  1. Una burbuja o proceso por cada acontecimiento de la lista.
  2. La burbuja se nombra identificando la respuesta del sistema al acontecimiento.
  3. Se dibujan las entradas-salidas y los almacenamientos apropiados para que la burbuja “funcione”.
  4. Se chequea el borrador de DFD obtenido con el diagrama de contexto y la lista de acontecimientos.
  5. Una vez validado se hace una nivelación:
    * Ascendentes:
        Agrupa burbujas con algún criterio. Agrupa las burbujas que acceden al mismo almacenamiento, con
el fin de "ocultamiento de información".
    * Descendentes:
        Descompone burbujas funcionalmente.
```

Diagrama de Flujo de Control (DFC)

La notación del flujo de datos convencional (DFD) no hace distinciones 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.

Estas ampliaciones permiten reflejar el flujo de control y el procesamiento de control, así como el procesamiento y el flujo de datos.

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.

--> Metodologías Ágiles

Antes en general se realizaban sistemas críticos, desarrollados por grandes equipos, a menudo dispersos geográficamente que seguían un contrato muy riguroso y poco flexible. Sin embargo, cuando este enfoque fue aplicado a sistemas de negocio pequeños y de tamaño medio, el esfuerzo invertido era grande, y cuando cambiaban los requerimientos, se hacia esencial rehacer el trabajo. Del descontento nacieron las metodologías agiles.

Las metodologías ágiles son un enfoque iterativo e incremental (evolutivo) de desarrollo de software, cuyos son objetivos entre otros son la *producción de software de calidad con un costo y tiempo apropiado* como tambien *responder a los cambios que surjan* a lo largo del proyecto, brindando así una mayor flexibilidad a los procesos de software tradicionales.

Las metodologías ágles presentan algunos 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, aún tardíos, en el desarrollo. 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 de progreso.
  - Los procesos ágiles 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 cómo volverse más efectivo, entonces afina y ajusta su comportamiento en consecuencia.

Ágil	**No Ágil**
Pocos artefactos	Muchos artefactos
Pocos roles	Muchos roles
Contratos flexibles o inexistentes	Contratos rigurosos
El cliente forma parte del equipo	El cliente interactua con reuniones
Grupos pequeños in-situ	Grupos grandes
Menos énfacis en la arquitectura	La arquitectura es esencial

EXtreme Programming (XP)

Las características esenciales :

- **Historias de usuario**
  - El cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales.
  - Le da tiempo suficiente al programador para implementarla en poco tiempo.
  - Muy flexibles y dinámicas. Pueden cambiarse o reemplazarse en cualquier momento.
- **Roles**
  - Programador:
    - Responsbale de construir el sistema
    - En XP, diseñan, programan y realizan las pruebas
  - Jefe de Proyecto (Manager):
    - Organiza y guia las pruebas

        - Asegura las condiciones para el desarrollo del proyecto
    - Cliente:
        - Es parte del equipo
        - Determina qué y cuándo construir
        - Establece las pruebas funcionales
    - Encargado de pruebas:
        - Ayuda al cliente con las pruebas funcionales
        - Se encarga de asegurarse que las pruebas funcionales se superen
    - Entrenador:
        - Responsable del proceso
        - Tiende a estar en 2do plano, a medida que el equipo madura
    - Rastreador:
        - Observa sin molesstar
        - Conserva datos históricos
- **Proceso**
    i. Exploración
    - Los clientes plantean historias de usuario
    - Programadores se familiariza con tecnologías
    - Se construye un prototipo

    ii. Planificación
    - El cliente establece prioridad en las historias de usuario
    - Los programadores realizan la estimación de tiempo
    - Se acuerda primera entrega y se determina el cronograma con el cliente

    iii. Iteraciones
    - El plan de entrega esta compuesto por iteraciones de no mas de 3 semanas
    - El cliente decide que historias implementar
    - En la última iteración el sistema está listo para entrar en producción

    iv. Producción
    - Esta fase requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente.
    - Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

    v. Mantenimiento
    - Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones.
    - La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

    vi. Muerte
    - Es cuando el cliente no tiene más historias para ser incluidas en el sistema.
    - Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura.
    - La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.
- **Prácticas**
    - Testing:
        - Los programadores escriben pruebas unitarias, que deben pasar.
        - Los clientes escriben pruebas demostrando que el sistema funciona hasta el momento
    - Refactoring:
        - Los realizan los programadores, para mejorar la calidad del código del sistema.
    - Programación a Pares
    - Propiedad colectiva del código:
        - Todos pueden cambiar el código del sistema. Evitando programadores indispensables.
    - Integración Continua:
        - Cada pieza del sistema es integrada una vez que está lista.
    - Semana de 40-horas:
        - Máximo de 40 horas semanales, de otra forma desmotiva al equipo
    - Cliente en el equipo de Desarrollo

        - El cliente tiene que estar presente y disponible todo el tiempo para el equipo
    - Estándares de codificación

SCRUM

Es un proceso *iterativo e incremental* donde se busca poder atacar todos los problemas que surgen durante el desarrollo. El nombre se debe a que durante los *sprints* se solapan lo que serían las fases de desarrollo repitiendonse todas en cada iteración, y no en cascada. Se realizan entregas parciales y regulares del resultado final del proyecto, priorizadas por el beneficio que aportan al receptor del proyecto. **SCRUM** se recomienda ser aplicado donde el "caos" es una constante. Requermiento dinámicos y donde tenemos que implementar tecnología de punta. También donde son equipos altamente competitivos.

Los principios del SCRUM son:

- Elimiar el desperdicio: No perder tiempo haciendo cosas sin valor para el cliente.
- Construir la calidad con el producto: Inyectar la calidad directamente desde el principio.
- Crear conocimiento: No se puede tener conocimiento desde antes de empezar el proyecto.
- Difereir decisiones: Dejar la decisiones para el momento que se necesiten tomarlas, puesto que se puede tener más información en ese momento.
- Entregar rápido: Da competencia.
- Respetar a las personas: Respetar a todo el equipo, permite motivar.
- Optimizar el todo: Optimizar cada proceso para mejorar continuamente.

El SCRUM cuenta con los siguientes roles:

- **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.
- **Scrum Team (Equipo)** son las personas responsables de implementar la funcionalidad o funcionalidades elegidas por el Product Owner.
- **Usuarios o Cliente** son los beneficiarios finales del producto, y son quienes viendo los progresos, pueden aportar ideas, sugerencias o necesidades.

Los **artefactos** del SCRUM son:

- **Product Backlog**: es la lista maestra que contiene toda la funcionalidad deseada en el producto. La característica más importante es que la funcionalidad se encuentra ordenada por un orden de prioridad.
- **Sprint Backlog**: es la lista que contiene toda la funcionalidad que el equipo se comprometió a desarrollar durante un Sprint determinado.
- **Burndown Chart**: muestra un acumulativo del trabajo hecho, día-a-día.
- Etc

Desarrollo de Software Basado en Modelos (MBD)

Es este proceso, se hace énfasis en la creación de un modelo predecesor a la creación del software, tal como en los 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 debe satisfacer. Un **modelo** es la descripción de un sistema (o parte) con un lenguaje bien definido. Un **lenguaje bien definido** es un lenguaje con sintaxis y semántica que sea apropiado para que un computador lo entienda. Generalmente se representa como una combinación de textos y diagramas.

Esto trae consigo algunos problemas como son:

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

  ..... | Requisitos | .....

    - Basicamente textos

  ..... | Análisis | .....

    - Textos y digramas

..... | Diseño | .....

- Textos y digramas

..... | Codificación | .....

- Código

..... | Testeo | .....

- Código

..... | emplazamiento | .....

Desarrollo de Software Dirigido por Modelos (MDD)

Enfatiza que este paradigma asigna a los modelos un rol central y activo: *son al menos tan importantes como el código fuente.*

MDD es la evolución natural de **MBD** enriquecida mediante el **agregado de transformaciones automáticas entre modelos.**

Promueve enfatizar los siguientes puntos clave:

- Mayor **abstracción** en la espeicificación del problema y de la solución
- Aumento de confianza en la **automatización**
- Uso de **estándares**
- Los modelos son los conductores primarios en todo el desarrollo.

..... | Requisitos | ..... * Basicamente textos

..... | Análisis | ..... * **PIM**

..... | Diseño | ..... * **PSM**

..... | Codificación | ..... * Código

..... | Testeo | ..... * Código

..... | emplazamiento | .....

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 específica que se usará para su implementación sobre una plataforma específica

Transformación de modelos:

Especifica el proceso de conversión de un modelo en otro modelo del mismo sistema.
Cada transformación incluye (al menos): * un PIM, * un Modelo de la Plataforma, * una Transformación, y * un PSM

Beneficios frente a MBD:

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