3. METODOLOGÍAS DE GESTIÓN DE PROYECTOS

3.1. Introducción

En este capítulo describiremos las metodologías más importantes y difundidas en la gestión de proyecto, siempre con un enfoque hacia la gestión de los proyectos software.

La Metodología, (del griego metà "más allá", odòs "camino" y logos "estudio"), hace referencia al conjunto de procedimientos basados en principios lógicos, utilizados para alcanzar una gama de objetivos que rigen en una investigación científica o en una exposición doctrinal.

En el ámbito de la gestión de proyectos, podemos definir una metodología como un conjunto de técnicas, recomendaciones y verificaciones, que permitan sistematizar los procesos en los que se descompone la gestión de un proyecto.

El uso de una metodología puede aportar muchas ventajas a la gestión de un proyecto, como pueden ser:

- Facilitar la tarea de planificación.
- Facilitar la tarea del control y seguimiento de un proyecto.
- Mejorar la relación coste/beneficio.
- Optimizar el uso de recursos disponibles.
- Facilitar la evaluación de resultados y el cumplimiento de los objetivos.
- Facilitar la comunicación efectiva entre los interesados del proyecto.
- Optimizar las fases del proceso de desarrollo.
- Facilitar el mantenimiento del producto final.
- Permitir la reutilización de partes del producto.
- Garantía de un nivel de calidad en el producto final.
- Ayudar en el cumplimiento de los plazos de tiempo fijados en la definición del proyecto.
- Definir el ciclo de vida que más se ajuste a las condiciones y características del desarrollo.

Según la filosofía de desarrollo, las metodologías se pueden clasificar en dos grupos, las metodologías tradicionales, que se basan en una fuerte planificación durante todo el desarrollo y un ciclo de vida más tradicional, y las metodologías ágiles, en las que el desarrollo de software es incremental, cooperativo, sencillo y adaptado.

Las metodologías tradicionales son denominadas, como metodologías pesadas. Centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto, definido en la fase inicial del desarrollo del proyecto. Otra de las características más importantes dentro de este enfoque, son los altos costes al implementar un cambio y la falta de flexibilidad. Las metodologías tradicionales se focalizan en la documentación, planificación y procesos (plantillas, técnicas de administración, revisiones, etc.).

Las metodologías ágiles nacen como respuesta a los problemas que puedan ocasionar las metodologías tradicionales y se basa en dos aspectos fundamentales, retrasar las decisiones y la planificación adaptativa. Destacan en la adaptabilidad de los procesos de desarrollo. Estas metodologías destacan que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan.

A continuación en este capítulo, repasaremos las siguientes metodologías:

METODOLOGÍAS TRADICIONALES	METODOLOGÍAS ÁGILES
MÉTRICA V3	Extreme-Programming XP
PRINCE2	Scrum
SSADM	Crystal Methodologies
MERISE	Adaptive Software Development
	Feature-Driven Development (FDD)
	Dynamic Systems Development Method (DSDM)
	Lean software development

Tabla 4 Metodologías

3.2. Metodologías tradicionales

3.2.1. MÉTRICA V3

3.2.1.1. Introducción

MÉTRICA [35] es una metodología de planificación, desarrollo y mantenimiento de sistemas de información, promovida por el Ministerio de Administraciones Públicas del Gobierno de España para la sistematización de actividades del ciclo de vida de los proyectos software en el ámbito de las administraciones públicas.

Esta metodología propia está basada en el modelo de procesos del ciclo de vida de desarrollo ISO/IEC 12207 (Information Technology - Software Life Cycle Processes) así como en la norma ISO/IEC 15504 SPICE (Software Process Improvement And Assurance Standards Capability Determination).

Se han ido desarrollando diferentes versiones, siendo la V3 la más reciente:

- En 1989 apareció la primera versión de Métrica V.1: Guías metodológicas que explicaba paso a paso todas las actividades a realizar para lograr el producto deseado, en un tiempo y costes aceptados y con una definición de las personas participantes.
- En 1993, se publicó Métrica V.2: Guía técnica, de referencia y de usuario, que rápidamente fue sustituida por la V.2.1 en julio de 1995 que incluía 5 fases bien estructuradas que permitía una mayor facilidad para el soporte de los cambios futuros del sistema.

- En julio del 2001 se liberó la V.3 en la que se han tenido en cuenta las diferentes tecnologías actuales (cliente/servidor, orientación a objetos, reutilización, etc.). Esta versión aporta un conjunto de procesos (definidos en la metodología como interfaces) orientados a la organización y como apoyo al propio proceso de desarrollo. Entre los interfaces destaca uno específico para la gestión de los proyectos y que se estudia más en detalle.

Las razones más relevantes para hacer uso de Métrica son:

- Métrica favorece la implantación de una orientación a procesos en las organizaciones.
- De esa forma se identifican claramente las actividades a realizar, los elementos necesarios como inputs o entradas a esa actividad y los resultados o salidas a obtener
- Todo ello facilita la normalización de procesos en la organización
- Y la puesta en marcha de acciones de mejora continua.

En definitiva, ayuda a mejorar la productividad y la calidad de los servicios.

3.2.1.2. Descripción

La metodología MÉTRICA V3 tiene un enfoque orientado al proceso y descompone cada uno de los mismos en actividades, y éstas a su vez en tareas. Para cada tarea se describe su contenido haciendo referencia a sus principales acciones, productos, técnicas, prácticas y participantes.

Así los procesos de la estructura principal de MÉTRICA V3 son los siguientes:

- Planificación de Sistemas de Información (PSI)
- Desarrollo de Sistemas de Información (EVS, ASI, DSI, CSI, IAS)
- Mantenimiento de Sistemas de Información (MSI)

También incluye un conjunto de interfaces que definen una serie de actividades de tipo organizativo o de soporte al proceso de desarrollo y a los productos, que se deberán aplicar como apoyo a la ejecución de los procesos principales de la metodología y para complementar y garantizar el éxito del proyecto

Las interfaces incluidas en la metodología son:

- Gestión de Proyectos (GP)
- Seguridad (SEG)
- Aseguramiento de la Calidad (CAL)
- Gestión de la Configuración (GC)

Procesos		
Planificación de Sistemas de Información • PSI	Desarrollo de Sistemas de Información • EVS (Estudio de la Viabilidad del Sistema) • ASI (Análisis del Sistema de Información) • DSI (Diseño del Sistema de Información) • CSI (Construcción del Sistema de Información) • IAS (Implantación y Aceptación del Sistema)	Mantenimiento de Sistemas de Información • MSI

Tabla 5 Procesos Métrica V3

Interfaces			
Gestión de Proyectos • GP	Seguridad • SEG	Aseguramiento de la calidad • CAL	Gestión de la Configuración • GC

Tabla 6 Interfaces Métrica V3

A continuación se describirán cada uno de los procesos:

Figura 1 Procesos Métrica V3

- **PSI. Planificación de Sistemas de Información:**

  - Objetivo:
    - Tiene como objetivo proporcionar un marco estratégico de referencia para los Sistemas de Información de un determinado ámbito de la organización.
    - Desarrollo:
      - Estudio de las necesidades de información de los procesos de la organización afectados por el Plan de Sistemas, con el fin de definir los requisitos generales y obtener modelos conceptuales de información.

- Evaluación de las opciones tecnológicas y propuesta de un entorno.
- Análisis de las prioridades relacionadas con las distintas variables que afectan a los Sistemas de Información.
- Elaboración de un calendario de proyectos con una planificación lo más detallada posible de los más inmediatos.
- Desarrollo de una sistemática para mantener actualizado el Plan de Sistemas de Información para incluir en él todos los cambios necesarios.

- Resultados:
  - Orientar las actuaciones en materia de Desarrollo de Sistemas de Información con el objetivo básico de apoyar la estrategia corporativa, elaborando una arquitectura de información y un plan de proyectos informáticos para dar apoyo a los objetivos estratégicos.

- **EVS. Estudio de la Viabilidad del Sistema:**
  - Objetivo:
    - Análisis de un conjunto concreto de necesidades para proponer una solución a corto plazo, que tenga en cuenta restricciones económicas, técnicas, legales y operativas.
  - Desarrollo:
    - Se propondrán diferentes alternativas, que deben ser valoradas (se estudiará el impacto en la organización de cada una de ellas, la inversión y los riesgos asociados) para determinar una única solución.
    - Los criterios con los que se valoran las propuestas no serán estratégicos sino tácticos y relacionados con aspectos económicos, técnicos, legales y operativos.
  - Resultados:
    - En este proceso se decide si continuar con el desarrollo de la solución adoptada o abandonar, después de valorar su impacto en la organización, el presupuesto necesario y los riesgos que conlleva. Como productos relacionados con la solución adoptada se obtiene el contexto del sistema , el impacto en la organización, el coste/beneficio de la solución, la valoración de riesgos, el plan de trabajo, la planificación y la solución propuesta.

- **ASI. Análisis del Sistema de Información:**
  - Objetivo:
    - Conseguir la especificación detallada del Sistema de Información, a través de un catálogo de requisitos y una serie de modelos que cubran las necesidades de información de los usuarios para los que se desarrollará el sistema de información y que serán la entrada para el proceso de Diseño del Sistema de Información.

- Desarrollo:
  - En primer lugar se describe inicialmente el Sistema de Información. Se delimita su alcance, se genera un catálogo de requisitos generales y se describe el sistema mediante unos modelos iniciales de alto nivel.
  - Se recogen de forma detallada los requisitos funcionales que el Sistema de Información debe cubrir, catalogándolos. Además, se identifican los requisitos no funcionales del sistema (las facilidades que ha de proporcionar el sistema, y las restricciones en cuanto a rendimiento, frecuencia de tratamiento, seguridad, etc.).
  - Se identifican los subsistemas de análisis, y se elaboran los modelos de Casos de Uso y de Clases, en desarrollos orientados a objetos, y de Datos y Procesos en desarrollos estructurados.
  - Finalizados los modelos, se realiza un análisis de consistencia que puede forzar la modificación de algunos de los modelos obtenidos y, posteriormente, se elabora el producto Especificación de Requisitos Software.
  - Se inicia la especificación del Plan de Pruebas, que se completará en el proceso Diseño del Sistema de Información (DSI).
- Resultados:
  - Se obtiene el catálogo de requisitos del sistema a implementar, así como los estándares y normas a tener en cuenta, la descripción general del entorno tecnológico y la especificación de la interfaz de usuario.
  - Si el sistema a desarrollar sigue un enfoque Estructurado, además hay que añadir el plan de migración y carga inicial de datos, el contexto del sistema, la matriz de procesos/localización geográfica, la descripción de interfaz con otros sistemas, el modelo de procesos y modelo lógico de datos normalizado.
  - Si se sigue un enfoque Orientado a Objetos, hay que añadir la descripción de subsistemas de análisis, la descripción de interfaces entre subsistemas, el modelo de clases de análisis, el comportamiento de clases de análisis y el análisis de la realización de los casos de uso.

- **DSI. Diseño del Sistema de Información:**
  - Objetivo:
    - Definir la arquitectura del sistema y el entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del Sistema de Información.

- Desarrollo:
  - Las actividades de este proceso se agrupan en dos grandes bloques:
    - En un primer bloque se obtiene el diseño de detalle del Sistema de Información que comprende la partición física, independiente de un entorno tecnológico concreto, la organización en subsistemas de diseño, la especificación del entorno tecnológico sobre el que se despliegan dichos subsistemas y la definición de los requisitos de operación, administración del sistema, seguridad y control de acceso.
    - El segundo bloque complementa el diseño del Sistema de Información y en él se generan todas las especificaciones necesarias para la construcción del Sistema.
- Resultados:
  - Especificaciones de construcción relativas al propio sistema, la descripción técnica del plan de pruebas, la definición de los requisitos de implantación y el diseño de los procedimientos de migración y carga inicial, si procede.

- **CSI. Construcción del Sistema de Información:**
  - Objetivo:
    - Construcción y prueba de los distintos componentes del Sistema de Información, a partir del conjunto de especificaciones lógicas y físicas del mismo, obtenido en el Proceso de Diseño del Sistema de Información (DSI). Se desarrollan los procedimientos de operación y seguridad, y se elaboran los manuales del usuario final y de explotación.
  - Desarrollo:
    - En este proceso se genera el código de los componentes del Sistema de Información, se desarrollan todos los procedimientos de operación y seguridad y se elaboran todos los manuales de usuario final y de explotación.
    - Además se realizan los siguientes conjuntos de pruebas:
    - Pruebas unitarias.
    - Pruebas de integración de los subsistemas.
    - Pruebas del sistema.
    - Asimismo, se define la formación de usuario final y, si procede, se construyen los procedimientos de migración y carga inicial de datos.
  - Resultados:
    - Código fuente de los componentes desarrollados, así como los procedimientos de operación y administración del sistema y los procedimientos de seguridad y control de acceso.
    - Por otra parte se elaboran los manuales de usuario y la especificación de la formación a usuarios finales, además de los productos correspondientes al proceso de migración y carga inicial de datos.

- **IAS. Implantación y Aceptación del Sistema:**
  - Objetivo:
    - Este proceso tiene como objetivo principal la entrega y aceptación del sistema en su totalidad, y la realización de todas las actividades necesarias para su paso a producción.
  - Desarrollo:
    - Revisión de la estrategia de implantación determinada en el Estudio de Viabilidad del Sistema (EVS).
    - Preparación de la infraestructura necesaria, la instalación de los componentes, la activación de los procedimientos manuales y automáticos asociados y la migración o carga inicial de datos.
    - Realización de las pruebas de implantación por parte del usuario de operación y de las pruebas de aceptación por parte del usuario final.
    - Determinar los servicios que requiere el sistema que se va a implantar una vez que se inicie la puesta en producción y al nivel con el que se prestarán dichos servicios.
  - Resultados:
    - Plan de implantación del sistema en su totalidad, el equipo de implantación, el plan de formación del equipo de implantación (esquema, materiales, recursos necesarios, planificación y especificación de la formación de usuarios finales), la evaluación de las pruebas de implantación del sistema y de las pruebas de aceptación del sistema, el plan de mantenimiento previo al paso a producción, el acuerdo de nivel de servicio del sistema y el Sistema en producción.

- **MSI. Mantenimiento del Sistema de Información:**
  - Objetivo:
    - Obtención de una nueva versión del sistema de información, a partir de las peticiones de mantenimiento que los usuarios realizan con motivo de un problema detectado en el sistema o por la necesidad de una mejora del mismo.
  - Desarrollo:
    - Ante una petición de cambio de un sistema de información ya en producción, se realiza un registro de las peticiones, se diagnostica el tipo de mantenimiento y se decide si se le da respuesta o no, en función del plan de mantenimiento asociado al sistema afectado por la petición, y se establece con qué prioridad.
    - La definición de la solución al problema o necesidad planteada por el usuario que realiza el responsable de mantenimiento, incluye un estudio del impacto, la valoración del esfuerzo y coste, las actividades y tareas del proceso de desarrollo a realizar y el plan de pruebas de regresión.

- Resultados:
  - Catálogo de peticiones de cambio, el resultado del estudio de la petición, la propuesta de solución, el análisis de impacto de los cambios, el plan de acción para la modificación, el plan de pruebas de regresión, la evaluación del cambio y la evaluación del resultado de las pruebas de regresión.

MÉTRICA V3 proporciona cuatro interfaces que definen actividades orientadas a la mejora y perfeccionamiento de los procesos principales con el fin de garantizar la consecución del objetivo del desarrollo

Contempla el siguiente conjunto de interfaces:

- **GP. Gestión de Proyectos**
  - La Gestión de Proyectos tiene como finalidad principal la planificación, el seguimiento y control de las actividades y de los recursos humanos y materiales que intervienen en el desarrollo de un Sistema de Información.
  - Se estructura en 3 grupos de actividades fundamentales:
    - Inicio del proyecto (GPI): Una vez concluido el esfuerzo del estudio de viabilidad del sistema, se realizarán las actividades de estimación del esfuerzo y planificación del proyecto
    - Seguimiento y Control del Proyecto (GPS): Comprende la asignación de tareas, la gestión de las incidencias, los cambios de requisitos que puedan presentarse y afectar a la planificación del proyecto. El seguimiento y Control se realiza durante los procesos de desarrollo del proyecto (análisis, diseño, construcción, implantación y aceptación del sistema de información) con el objeto de vigilar el correcto desarrollo de las tareas planificadas.
    - Finalización del proyecto (GPF): Realiza las tareas propias de cierre del proyecto.
- **SEG. Seguridad**
  - El objetivo de la interfaz de Seguridad es incorporar en los sistemas de información mecanismos de seguridad adicionales a los que se proponen en la propia metodología, asegurando el desarrollo de cualquier tipo de sistema a lo largo de los procesos que se realicen para su obtención
  - La interfaz de Seguridad hace posible incorporar durante la fase de desarrollo las funciones y mecanismos que refuerzan la seguridad del nuevo sistema y del propio proceso de desarrollo, asegurando su consistencia y seguridad
- **CAL. Aseguramiento de la Calidad**
  - El objetivo es proporcionar un marco común de referencia para la definición y puesta en marcha de planes específicos de aseguramiento de calidad aplicables a proyectos concretos.

        - Las actividades están orientadas a verificar la calidad de los procesos y productos. Son actividades que evalúan la calidad y que son realizadas por un grupo de Garantía de la Calidad independiente de los responsables de la obtención de los productos.
- **GC. Gestión de la Configuración**
    - El objetivo de la gestión de la configuración es mantener la integridad de los productos que se obtienen a lo largo del desarrollo de los sistemas de información, garantizando que no se realizan cambios incontrolados y que todos los participantes en el desarrollo del sistema disponen de la versión adecuada de los productos que manejan. Así, entre los elementos de configuración software, se encuentran no únicamente ejecutables y código fuente, sino también los modelos de datos, modelos de procesos, especificaciones de requisitos, pruebas, etc.
    - Se realiza durante todas las actividades asociadas al desarrollo del sistema, y continúa registrando los cambios hasta que éste deja de utilizarse
    - Facilita el mantenimiento del sistema, aportando información para valorar el impacto de los cambios solicitados y reduciendo el tiempo de implementación de un cambio, tanto evolutivo como correctivo. Asimismo, permite controlar el sistema como producto global a lo largo de su desarrollo, obtener informes sobre el estado de desarrollo en que se encuentra y reducir el número de errores de adaptación del sistema, lo que se traduce en un aumento de calidad del producto, de la satisfacción del cliente y, en consecuencia, de mejora de la organización.

Para el desarrollo de los proyectos, METRICA V3 utiliza técnicas ampliamente difundidas y usadas, como son:

- Análisis coste beneficio
- Diagramas UML
- Diagramas de Flujo de Datos
- Diagrama de Interacción
- Diagramas de paquetes
- Diagramas de transición de estados
- Modelado de procesos de la organización
- Modelo entidad/relación
- Normalización y optimización
- Técnicas matriciales

Para el interfaz específico de gestión de los proyectos, Métrica v.3 añade otras técnicas como son:

- **Estimación**: Usa los métodos Albrecht y MARKII que es un método evolucionado de Albrecht, e inventado por Charles Symons. Ambos sirven para medir el tamaño funcional de cualquier aplicación. MarkII supera los problemas de Albrecht contemplando el sistema como una colección de transacciones lógicas compuestas por componentes de entrada, de proceso y de salida y que se corresponden con las funciones del sistema.
- **WBS (Work Breakdown Structure):** Esta técnica de descomposición permite estructurar las actividades sirviendo de lista de comprobación y de herramienta de contabilidad analítica del proyecto software. Permite la descomposición de las actividades de un proyecto según su naturaleza en forma de árbol con agrupación de actividades en: desarrollo, calidad, gestión, etc. Posteriormente y mediante los diagramas PERT y GANTT que se deduzcan de dicho árbol, se puede obtener una planificación de una forma más fácil y clara.
- **PERT (Program Evaluation and Review Technique)**: Sirve para establecer las dependencias entre las distintas tareas del proyecto para saber de qué manera han de encontrarse dichas tareas en la planificación. Este método parte de la descomposición del proyecto en una serie de actividades y estas actividades están controlados por los sucesos entendidos éstos como acontecimientos o principio o fin de la actividad.
- **Diagrama de GANTT:** Esta técnica tiene como objetivo la representación del plan de trabajo, mostrando las tareas a realizar, el momento de comienzo o fin y la forma en que las distintas tareas están unidas entre sí. Es la forma clásica de representar el plan de proyecto.

3.2.2. PRINCE2

3.2.2.1. Introducción

PRINCE2 (PRojects IN Controlled Environments) [29], fue desarrollado para el gobierno del Reino Unido y se utiliza regularmente no solo en el gobierno británico sino también en el sector privado. Actualmente es el estándar de facto en el Reino Unido.

PRINCE2 es una metodología estructurada basada en procesos. Ofrece una guía de dominio público para la aplicación de las mejores prácticas en la gestión de los proyectos.

En 1989 la Agencia Central de Computación y Telecomunicaciones (CCTA) desarrolló la técnica PRINCE como un estándar para la gestión de los proyectos de telecomunicaciones del gobierno del Reino Unido.

En 1996 se liberó una versión de PRINCE2 como una metodología genérica para la gestión de los proyectos. La última versión es la edición del 2009.

3.2.2.2. Descripción

PRINCE2 es una metodología para la gestión de los proyectos basada en ocho procesos que a su vez se componen de 45 subprocesos. La metodología cubre todos los aspectos de organización, y gestión y control de los proyectos. Tiene como objetivo, lograr que los productos se entreguen en el tiempo establecido y con el presupuesto acordado.

La metodología se puede aplicar a cualquier tipo de proyecto, y permite la gestión de los riesgos, el control de la calidad y la eficiencia de los cambios.

Las principales características de PRINCE2 se centran en el establecimiento de un ciclo de vida claro, la definición y medición de productos de negocio, el suministro de un conjunto de actividades para conseguir los productos de negocio ,y el establecimiento de una estructura organizativa con responsabilidades bien definidas para poder gestionar el proyecto de forma óptima.

PRINCE2 no cubre todos los aspectos de la gestión de los proyectos. Hay ciertos aspectos propios de la gestión de los proyectos que no están contemplados en la metodología como pueden ser el liderazgo, las habilidades para la gestión de recursos, así como la cobertura detallada de técnicas y herramientas propias de la gestión.

PRINCE2 está constituido de Procesos, Componentes y Técnicas.

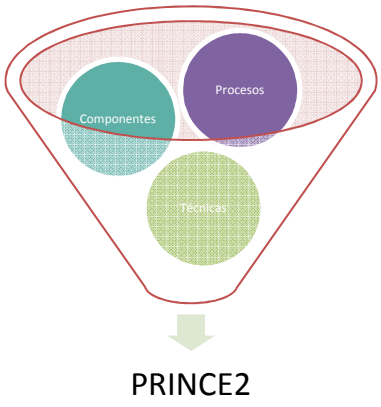


Figura 2 Elementos PRINCE2

PRINCE2 se centra en los componentes a los que considera fundamentales para la garantía de éxito y la finalización en plazos y tiempos de los proyectos. La estrategia consiste en construir procesos para vincular los componentes y reducir los riesgos de los proyectos, al mismo tiempo que proporciona las técnicas que los soportan y sugiere un modo efectivo de organizarlos.

Esta metodología es una combinación de ocho procesos, ocho componentes y de tres técnicas.

Los procesos que se describen son los siguientes:

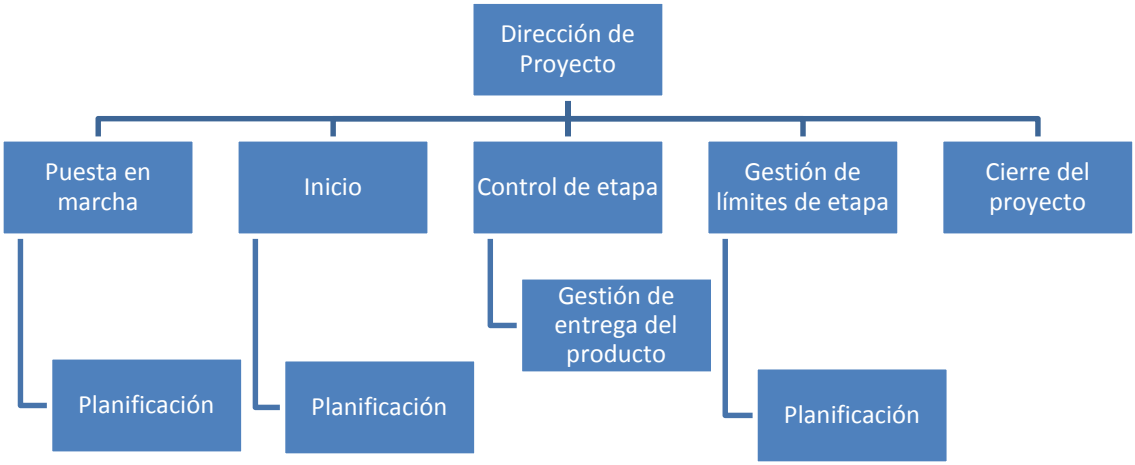


Figura 3 Procesos PRINCE2

- **Puesta en marcha del proyecto**: Permite un inicio controlado del proyecto. Solo se realiza al principio del ciclo de vida del proyecto y proporciona una preparación inicial para la gestión del resto del proyecto, así como para el control y viabilidad del proyecto. Este proceso crea la junta del proyecto, y garantiza el acuerdo de las necesidades de recursos.
- **Inicio del proyecto**: Es otro proceso que solo se realiza una vez durante el ciclo de vida del proyecto. Sirve para realizar un trazado de cómo se puede gestionar la totalidad del proyecto, y lo plasma en un "contrato", denominado documento de inicio del proyecto (PID Project Initiation Document). El objetivo de este documento es el establecimiento de un entendimiento común de los elementos críticos del proyecto, así como el acuerdo de la junta del proyecto para la primera etapa de desarrollo del proyecto.
- **Dirección del proyecto**: Dirige todo el proyecto y define las responsabilidades de la junta del proyecto en la supervisión del mismo. De acuerdo con su posición en el diagrama del modelo de procesos, está por encima de todos e interactúa con el resto de procesos. Proporciona los mecanismos para las autorizaciones de aprobación de continuidad al final de cada etapa y al cierre del proyecto. Este proceso es el marco de suministro de entradas, de recepción de requisitos y para la toma de decisiones. Es el único proceso en el que actúa la junta del proyecto, ya que el resto de procesos son conducidos por el director del proyecto y el resto del equipo de proyecto.

- **Gestión de los límites de la etapa:** Este proceso ayuda a realizar la transición de un estado finalizado al inicio del siguiente estado, al mismo tiempo que permite garantizar que el trabajo definido en el estado finalizado se ha realizado de acuerdo con los requisitos establecidos. También proporciona a la junta de proyecto, una ayuda para garantizar la viabilidad del proyecto, los planes de desarrollo, la autorización de la nueva etapa de trabajo y un archivado de lecciones aprendidas.
- **Control de etapa:** Suministra una guía para la gestión diaria del proyecto. Incluye: autorización y recepción de trabajos, gestión del cambio y de versiones, análisis e informes, consideraciones de viabilidad, acciones correctiva y escalado de incidencias a la junta de proyecto. Este proceso de control se realiza de forma iterativa por cada etapa de desarrollo del proyecto.
- **Gestión de entrega del producto:** Forma parte del sistema de autorización de Prince2. Es el mecanismo que sirve para que los ejecutores del trabajo técnico acuerden en los trabajos a realizar, los informes de progreso, etc. Se repite por cada paquete de trabajo autorizado.
- **Cierre del proyecto:** Es el mecanismo que permite la transición de entrega del proyecto a la organización. Puede finalizar por haber realizado el trabajo satisfactoriamente o por terminación prematura, aunque en cualquier caso, se almacenan las lecciones aprendidas. El proceso permite garantizar que si el cierre es por finalización del trabajo, ésta ha sido realizado a satisfacción del cliente y todos los productos han sido aceptados por el cliente, así como los acuerdos para el soporte de los productos del proyecto.
- **Planificación**: Es el proceso común para el resto de los procesos de Prince2. Los planes se producen identificando los entregables del proyecto, las actividades y recursos necesarios para crearlos, y todo ello, en una relación consistente con los requerimientos identificados en el PID.

Y los siguientes componentes:

Figura 4 Componentes PRINCE2

- **Proceso de Negocio:** La principal condición de control de un proyecto Prince2 es la existencia de un caso de negocio viable. El caso de negocio se verifica previamente pro el equipo de proyecto y es el punto principal de decisión del proyecto. El proyecto debería ser parado si el proyecto no es viable por alguna razón.
- **Organización:** Debido a la necesidad de informar desde el staff al resto de la estructura organizativa, se necesita una supervisión organizativa para asegurar la coordinación de todos esos recursos. Además, es necesario gestionar las decisiones de validación e inventariar las entregas a lo largo de la gestión del proyecto. En PRINCE2 esta supervisión es lo que Prince2 denomina Project Board.
- **Planes:** Los planes suponen la columna vertebral del sistema de información que gestiona el proyecto, y necesitan por lo tanto ser aprobados y aceptados por los niveles organizativos apropiados. El componente de planes resalta los conceptos fundamentales del proyecto resultando ser las tareas fundamentales del proceso de planificación del modelo de procesos.
- **Controles**: El control se refiere a la toma de decisiones: su propósito es garantizar que, por una parte el proyecto genera los productos necesarios definidos en los criterios de aceptación y por otra parte, que cumple la programación de acuerdo con los recursos y costes planificados. Además, debe garantizar la viabilidad del proceso de negocio.
- **Gestión del riesgo:** La gestión del riesgo es fundamental dentro de la gestión del proyecto, y debe realizarse de una manera disciplinada, ya que muchos de los trabajos de un proyecto no son previsibles.

- **Calidad en el entorno del proyecto**: La gestión de la calidad debe garantizar que se consigue la calidad esperada por el cliente mediante un sistema de calidad disciplinado. Los requerimientos de calidad de los entregables se basan en las descripciones del producto que a su vez son preparados por el director del proyecto y aprobados por la junta del proyecto.
- **Gestión de la configuración:** La gestión de la configuración proporciona al equipo de gestión del proyecto el control necesario para la validación del proyecto y es vital para el sistema de calidad. Este componente suministra los mecanismos para las cuestiones de trazabilidad del proyecto.
- **Control de cambios:** El control de los cambios del alcance calcula el impacto de los potenciales cambios, su importancia, costes, impacto en el proceso de negocio y la decisión de poder gestionar su inclusión o no.

PRINCE2 define las siguientes **técnicas**:

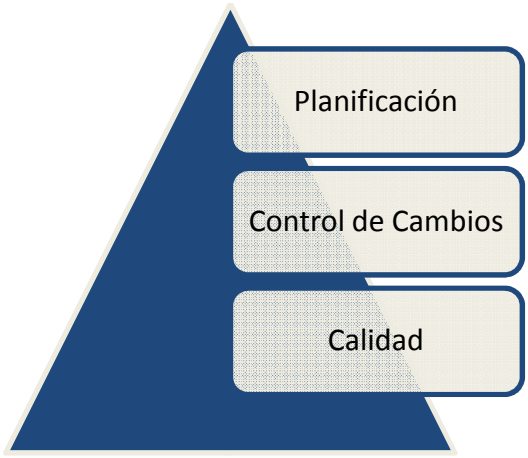


Figura 5 Técnicas PRINCE2

- **Planificación basada en el producto:** esta técnica involucra otros tres elementos que nos ayudan a la definición de los productos a entregar:
  - Product breakdown: diagrama de los productos.
  - Product description: descripción detallada de (los) producto(s).
  - Product Flor: descripción de la interrelación de productos.
- **Aproximación al control de cambios:** esta técnica nos garantiza someter a procesos toda la gerencia del proyecto basada en tener bajo control cualquier cambio que ocurra.
- **Revisiones de la calidad:** esta técnica nos ayuda a revisar los estándares ya existentes y también poder buscar nuevos que puedan ser aplicados. También nos ayuda a tener procedimientos exitosos así como tener un acercamiento a revisar cada uno de los elementos y productos a entregar. En esta técnica también involucra la correcta toma de decisiones del proyecto, la gestión de proveedores y el control de la información.

3.2.3. SSADM

3.2.3.1. Introducción

SSADM (Structured Systems Analysis and Design Method) [46] es una metodología de aproximación en cascada para el desarrollo de sistemas de información y que puede ser considerada como una de las más metodologías estructuradas más completas.

SSADM fue desarrollada inicialmente por Learmonth y Burchett Management Systems (LBMS) y continuada por el Central Computing and Telecommunications Agency (CCTA) mediante la adopción de un método de desarrollo de sistemas de información para el uso en los proyectos del gobierno del Reino Unido.

Se lanzó la primera versión en 1981 y en 1983 se convirtió en uso obligatorio para el desarrollo de todos los proyectos nuevos del gobierno de Reino Unido, y en 1988 fue promocionada como un Standard abierto. En el año 2000 CCTA renombró SSADM como "Business System Development"

Actualmente, es la metodología estándar de desarrollo de proyectos del gobierno del Reino Unido.

3.2.3.2. Descripción

SSADM se basa en 3 vistas fundamentales:

- **Modelo lógico de datos**: Es el proceso de identificar, modelar y documentar los requerimientos de información de un sistema de tecnología de la información. El modelo lógico de datos consiste en la estructura lógica de datos (LDS) y su documentación asociada. LDS representa las entidades y relaciones.
- **Modelo de flujo de datos**: Es el proceso de identificar, modelar y documentar como fluyen los datos a través del sistema de información. El Modelo de flujo de datos consiste en un conjunto de diagramas de flujo de datos (DFD) y su documentación asociada. Los DFD representan los procesos, entidades externas y flujos de datos.
- **Modelo de Eventos de entidad**: Es el proceso de identificar, modelar y documentar como los eventos de negocio que afectan a cada entidad y la secuencia en que ocurren. Un modelo entidad/evento consiste en un conjunto de historia de vida de las entidades y la correspondiente documentación. Es decir, este proceso representa el comportamiento de un sistema dinámico a lo largo del tiempo.

SSADM considera las estructuras de datos con mayor estabilidad que los procesos, y por lo tanto, los datos forman la columna vertebral de la metodología. De este modo, SSADM pertenece a la familia de los métodos "orientado a los datos".

SSADM es una aproximación top-down donde se representa el esquema de alto nivel y se va descomponiendo gradualmente en niveles inferiores. Además, aplica técnicas individuales así como el modo en que se pasa el control de una a otra una vez que la primera ha finalizado su actividad.

La gestión del proyecto se centra más en la monitorización de la calidad y la completitud de su producto que en la aplicación de las técnicas que lo crean.

SSADM consiste en una arquitectura de especificación de tres esquemas, considerando a su vez tres áreas del sistema de información:

- Capa externa mediante la que los usuarios interactúan con el sistema
- Diseño interno
- Modelo conceptual que representa los requerimientos de negocio y sobre los que se basa el diseño interno

SSADM cubre las tres fases fundamentales del ciclo de vida del desarrollo software: Estudio de viabilidad, Análisis y Diseño, pero no está diseñada para realizar la implementación y el mantenimiento de las aplicaciones, lo que hace es suministrar la documentación completa y precisa para poder mantener el sistema fácilmente.

Tabla 7 Fases ciclo de vida SSADM

Realizando ya un estudio más detallado, enunciamos a continuación las diferentes niveles en los que se descompone la metodología. En el siguiente gráfico se realiza un mapeo con las fases más típicas del ciclo de vida del desarrollo software.

Estudio de Viabilidad	• Nivel 0. Viabilidad
Análisis de requerimientos	• Nivel 1. Investigación del entorno actual • Nivel 2. Opciones del sistema de negocio
Especificación de requerimientos	• Nivel 3. Definición de requerimientos
Especificación del sistema lógico	• Nivel 4. Opciones técnicas del sistema • Nivel 5. Diseño Lógico
Diseño	• NIvel 6. Diseño físico

Tabla 8 Niveles SSADM

- **Nivel 0. Estudio de Viabilidad**: Durante esta fase se realizan las siguientes actividades:
  - Se define el alcance del sistema de información propuesto.
  - Se investigan las diversas opciones para el desarrollo del proyecto.
  - Se selecciona una de las opciones realizando un análisis coste-beneficio.
  - Búsqueda de riesgos que puedan poner en peligro la viabilidad del proyecto (el proyecto podría descartarse).
- **Nivel 1. Investigación del entorno actual**: Durante esta actividad se realizan las siguientes tareas:
  - Determinación si el sistema será uno nuevo o la ampliación o sustitución de uno existente.
  - Realización de un análisis completo de requerimientos mediante la modelización del sistema actual con el objeto de extraer los problemas existentes y las nuevas necesidades.
  - Detección los puntos débiles que se deben cubrir.
- **Nivel 2. Opciones de negocio del sistema**: Durante esta actividad se realizan las siguientes tareas:
  - Estudio del conjunto de requisitos obtenido del nivel anterior.
  - Definir las soluciones de negocio (no confundir con las tecnológicas), teniendo en cuenta los aspectos técnicos y físicos.
- **Nivel 3. Definición de requerimientos**: Durante esta tarea se realiza:
  - Transformación de los requerimientos en las especificaciones de lo que se quiere del sistema.
  - Aplicación de técnicas de modelado para pasar de análisis a diseño.

- **Nivel 4. Opciones técnicas del sistema:**
  - Se proponen las alternativas técnicas de implementación a partir de los resultados del nivel anterior.
  - Se especifican las distintas opciones hardware, software y de plataforma de desarrollo para poder seleccionar la más adecuada.
  - Esta tarea se realiza en paralelo con el diseño lógico.
- **Nivel 5. Diseño lógico:**
  - Se obtiene el proceso de diseño lógico independientemente del entorno técnico particular.
  - El diseño resultante podrá ser implementado en diversas plataformas.
  - Puede ser un modelo de cómo el sistema satisfará los requerimientos del usuario.
- **Nivel 6. Diseño físico**:
  - La información obtenida en la fase anterior se utiliza para trasladar al diseño físico, el entorno técnico seleccionado.
  - Se utilizan técnicas de selección del entorno, ya que SSADM simplemente se limita a suministrar unas directrices genéricas.

En relación a las técnicas de representación, las más usuales que utiliza SSADM serían las siguientes:

- **Diagramas de flujo de datos (DFD):** es una representación gráfica del "flujo" de datos a través de un sistema de información. Un diagrama de flujo de datos también se puede utilizar para la visualización de procesamiento de datos.

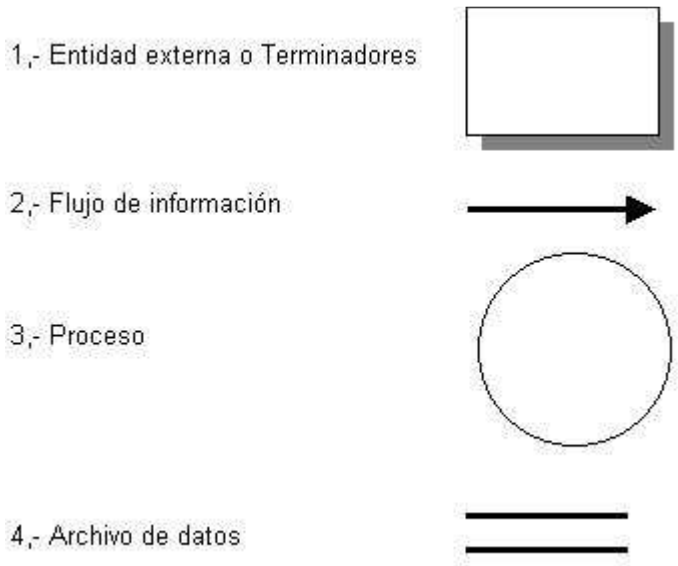


Figura 6 Diagramas de Flujo de Datos

- **Diagramas de datos lógicos:**
  - **Estructura lógica de datos (LDS):** Se basa en el modelo entidad/relación de CHEN, en el que se dibuja el diagrama inicial de E/R y se ajusta posteriormente convirtiendo las relaciones en entidades compuestas e identificando nuevas relaciones.
  - **Diagramas de entidad / relación**: Se utiliza para representar estructuras de datos del mundo real.
- **Diagramas de comportamiento de entidad**: se utilizan dos variantes:
  - **Diagramas Historia de vida de entidades (ELH):** Representan todo el proceso de creación, actualización y desaparición de una entidad a lo largo del tiempo.
  - **Diagramas de correspondencia (ECD):** Relacionan el número máximo de ocurrencias de una entidad que puede intervenir en una ocurrencia de relación.

3.2.4. MERISE

3.2.4.1. Introdución

MERISE [47] es una metodología de la Administración francesa, creada por iniciativa del Ministerio de Industria Francés y desarrollada por Tardieu, Rochfeld y Colleti.

Comienza a desarrollarse en 1972, y se publica la primera versión a finales de 1976 con el objetivo de crear una metodología para las necesidades de la administración francesa por parte del CTI (Centre Technique Informatique) del Ministerio de Industria Francés.

En 1977 surgió la metodología RACINES tratando la informatización como un acto estratégico que maneja un recurso de una manera reflexiva y ordenada.

A lo largo de 1979 MERISE se orientó hacia el análisis y diseño de sistemas de información, aportando un plan de trabajo y técnicas de modelado para la fabricación de aplicaciones coherentes para el área de gestión de empresas, suponiendo una intersección entre informática y organización e integrando los sistemas así diseñados en el marco común diseñado por RACINES.

En 1982, y también, bajo el patrocinio del Ministerio de Industria Francés se procedió a actualizar RACINES y a definir sus interfaces con MERISE formando un cuerpo metodológico completo bajo la denominación de MERISE.

3.2.4.2. Descripción

La metodología MERISE introduce dos ciclos complementarios:

- El ciclo de abstracción
- El ciclo de decisión.

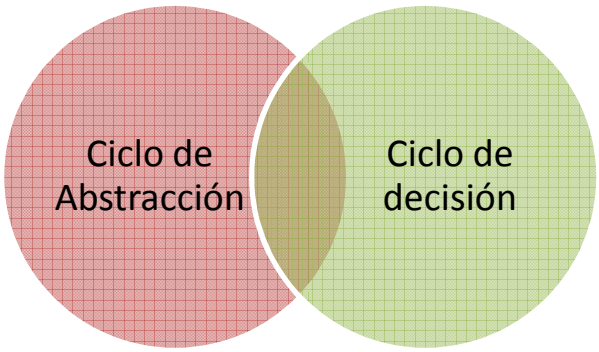


Figura 7 Ciclos MERISE

El ciclo de abstracción se basa en tres niveles:

- **Nivel conceptual**: Es donde se define el "qué" es decir, los objetivos y limitaciones. En este nivel se realiza un tratamiento de los datos según el modelo conceptual de datos y los procesos según el modelo conceptual de procesos.
- **Nivel Organizativo**: Es donde se define la organización adecuada que hay que implantar para alcanzar los objetivos asignados, y se realiza un tratamiento de los datos y el modelo organizativo de tratamientos para la realización de los procesos.
- **Nivel físico u operativo**: Es donde se realiza la integración de los medios técnicos necesarios para el proyecto. Utiliza el modelo físico de datos para los datos y el modelo operativo de tratamiento para los procesos.

MERISE cubre las cuatro fases fundamentales del ciclo de vida de desarrollo del software: Estudio de viabilidad (Estudio preliminar), Análisis, Diseño, e Implementación. Pero no cubre el mantenimiento de las aplicaciones, lo que hace es suministrar la documentación completa y precisa para poder mantener el sistema fácilmente.

Tabla 9 Ciclo de Vida MERISE

Cada una de estas fases se describe en el siguiente gráfico:

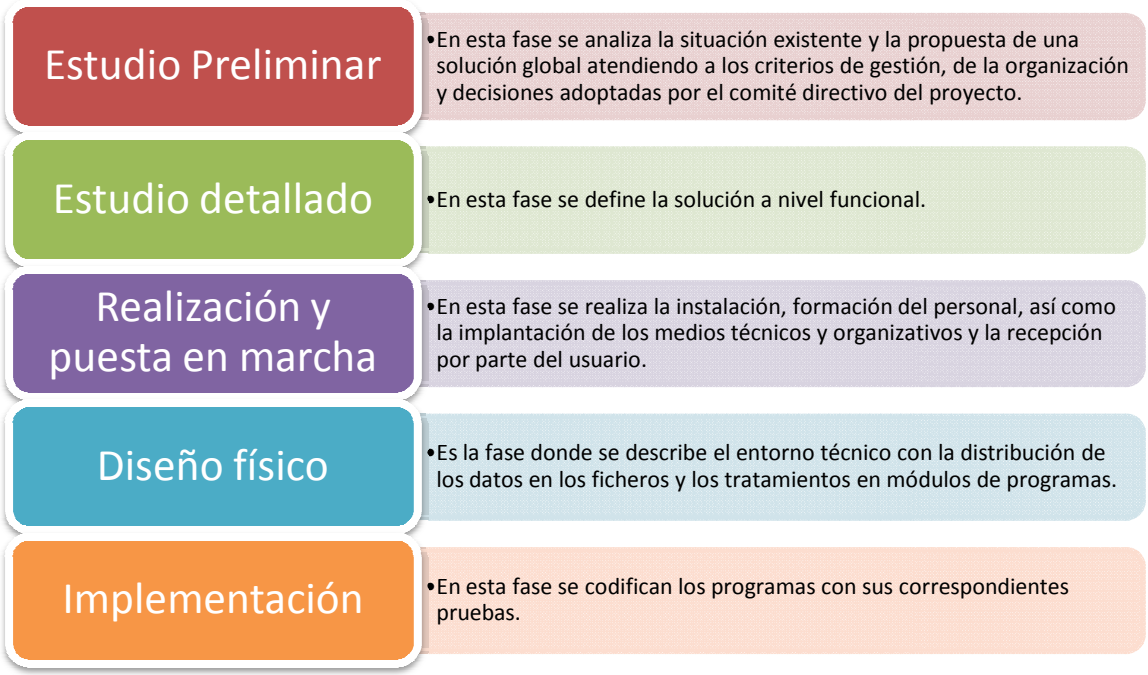


Tabla 10 Fases MERISE

Se definen tres equipos para el desarrollo de un proyecto:

- **Comité director**: Su función es fijar los objetivos y tomar las decisiones importantes en el desarrollo del mismo, interviniendo al final de cada etapa para realizar el control y seguimiento del proyecto y aprobando los informes de las mismas. Este equipo de trabajo estará formado por los directivos de cada área afectada, los responsables de los servicios implicados, el responsable del servicio de informática y el jefe del proyecto.
- **Comité de usuarios**: Su función es realizar un seguimiento sistematizado para la comprobación a lo largo del desarrollo de los cumplimientos de las actividades asignadas, mediante la comprobación de los diseños de las pantallas, validación de informes y de listados, resolución de problemas, etc. Estará compuesto por los responsables de los servicios afectados.
- **Equipo de desarrollo**: Su misión es elaborar los informes y documentación contemplados en cada una de las fases de desarrollo, así como la realización de los análisis, programación, pruebas, instalación y puesta en marcha. Este equipo estará formado por el jefe del proyecto y los analistas y programadores asignados al mismo, así como el representante del grupo de usuarios.

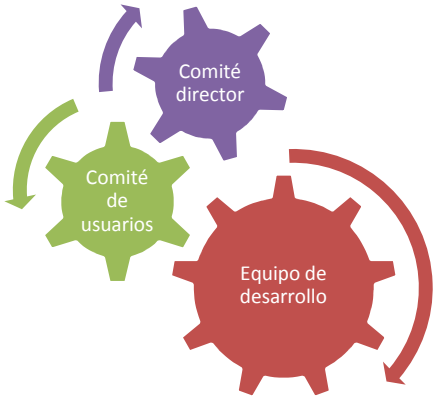


Figura 8 Equipos MERISE

La metodología MERISE utiliza las siguientes técnicas:

- **DFD (Diagramas de flujo de datos):** Para la representación gráfica de la organización identificando los flujos de información entre los diferentes actores.
- **Modelo conceptual de datos (Modelo entidad/relación):** Para la representación mediante estructuras del mundo real, pudiendo optar por cualquier tipo de base de datos.
- **Modelo lógico de datos:** Para adaptar el modelo conceptual de datos al sistema gestor de base de datos elegido.

- **Modelo conceptual de tratamientos**: Para representar las acciones a realizar sobre los datos con el objeto de obtener los resultados previstos. Para ello, utiliza las redes Petri.
- **Modelo organizativo de tratamientos:** Para describir cómo se va a organizar la ejecución de las mismas.
- **Modelo operacional de tratamientos:** Se parte de la descomposición de los procedimientos y fases anteriores y se obtienen los procedimientos manuales, las fases en tiempo real, las fases en tiempo diferido donde se incluyen las entradas, tratamientos y listados.

3.3. Metodologías Ágiles

3.3.1. Extreme-Programming (XP)

3.3.1.1. Introducción

La programación extrema o eXtreme Programming (XP) fue creada por Kent Beck en 1999, a través de su libro "Extreme Programming Explained: Embrace Change" [48]. Está especialmente diseñada para el desarrollo de software.

XP es una metodología ágil centrada en la potenciación de las relaciones interpersonales como clave para lograr el éxito. Promueve el trabajo en equipo, prestando atención al aprendizaje de los desarrolladores, y estableciendo un buen clima de trabajo.

XP utiliza la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y determinación frente a los cambios. Los principios y prácticas que describe pueden parecer de sentido común, pero se llevan al extremo, de ahí su nombre.

XP es especialmente adecuada para proyectos con requisitos imprecisos, cambiantes, y donde existe un alto riesgo técnico.

3.3.1.2. Descripción

A continuación se presentarán las características más relevantes de XP organizadas en los siguientes elementos:

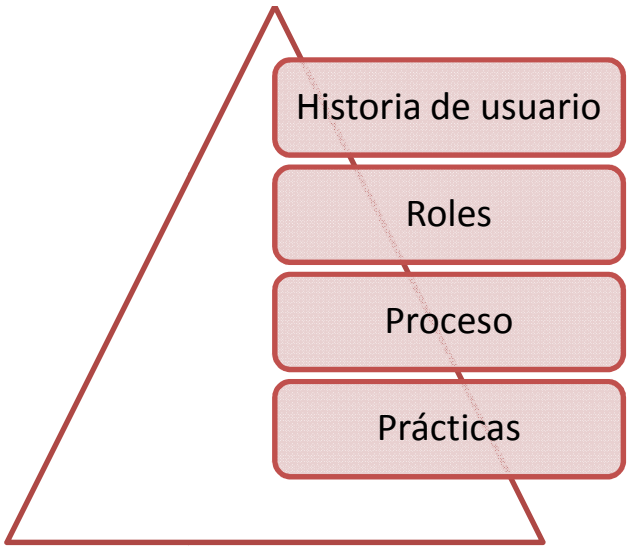


Figura 9 Características XP

- **Historia de usuario:** Esta técnica se utiliza para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe tener, sean requisitos funcionales o no funcionales.
  El tratamiento de las historias de usuario es muy dinámico y flexible.
  Cada historia de usuario debe ser entendible y suficientemente concreta para que los programadores puedan implementarla en unas semanas.

  Un ejemplo de ficha podría recoger los siguientes campos:

Customer Story and task card	
Fecha	Tipo de Actividad (nueva, corrección, mejora)
Prueba Funcional	Número de Historia
Prioridad	Referencia
Estimación técnica	Descripción
Notas	Lista de Seguimiento
Comentarios	Por Terminar
…	Etc.

Tabla 11 Ficha XP

A efectos de la planificación, las historias pueden durar de una a tres semanas de de programación (para no superar el tamaño de una iteración).
Las historias de usuario se descomponen en tareas de programación (task cards) y asignadas a los programadores para ser implementadas durante una única iteración.

- **Roles:** Los roles propuestos son:
  - Programador (Programmer). El programador escribe las pruebas unitarias y desarrolla el código del sistema.
  - Cliente (Customer). Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna prioridades a cada una de las historias de usuario,decidiendo cuáles se implementan en cada iteración según aporten mayor valor al negocio.
  - Encargado de pruebas (Tester). Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, di funde los resultados en el equipo y es el responsable de las herramientas que dan soporte a dichas pruebas.
  - Encargado de seguimiento (Tracker). Proporciona realimentación (feedback) al equipo. También verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada una de las iteraciones.
  - Entrenador (Coach). Responsable del proceso global. Provee de guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
  - Consultor. Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.
  - Gestor (Big boss). Es el vínculo entre clientes y programadores. Ayuda a que el equipo trabaje de forma efectiva creando las condiciones adecuadas. Su labor fundamental es de coordinación.

- **Proceso**: El ciclo de desarrollo se puede resumir en los siguientes pasos:

Figura 10 Ciclo de desarrollo XP

En todas las iteraciones, tanto el cliente como el programador, aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software y/o no se cumplirán los plazos.
De la misma forma el cliente tiene la obligación de gestionar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada una de las iteraciones.

El ciclo de vida ideal de XP consiste de seis fases:

- Exploración
- Planificación de la Entrega (Release)
- Iteraciones
- Producción
- Mantenimiento
- Muerte del Proyecto.

- **Prácticas**: Una de las grandes virtudes de XP es conseguir disminuir la curva exponencial, que supone la inclusión de cambios a lo largo de la vida del proyecto.
Esto se consigue gracias a las nuevas tecnologías disponibles para ayudar en el desarrollo de software y gracias a la aplicación de las siguientes prácticas definidas:

- El juego de la planificación. Existe una comunicación frecuente y fluida entre el cliente y el equipo de desarrollo. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas, y de cada iteración.
- Entregas pequeñas. Es necesario producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Cada versión ya constituye valor para el negocio. Una entrega no debería tardar más 3 meses.
- Metáfora. El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Podemos definir una metáfora como una historia compartida que describe cómo debería funcionar el sistema.
- Diseño simple. Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
- Pruebas. El desarrollo del código está dirigido por las pruebas unitarias. Éstas son establecidas por el cliente antes de desarrollarse el código, y son ejecutadas constantemente ante cada modificación del sistema.
- Refactorización (Refactoring). Es una actividad constante de reestructuración del código con el objetivo de eliminar código duplicado, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los futuros cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.
- Programación en parejas. Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores,.., etc.).
- Propiedad colectiva del código. Cualquier programador puede cambiar cualquier parte del código de la aplicación en cualquier momento.
- Integración continúa. 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.
- Evitar horas extras. No se deben trabajan horas extras en dos semanas seguidas. Este tipo de actuaciones normalmente encubre un problema de planificación que debe corregirse. El trabajo extra desmotiva al equipo.
- Cliente in-situ. El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de las principales clave del éxito del proyecto XP. El cliente dirige constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.
- Estándares de programación. XP enfatiza que la comunicación de los programadores sea a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

Para obtener el mayor beneficio posible de estas prácticas, se recomienda su aplicación conjunta, ya que unas se apoyan en otras. La mayoría de las prácticas propuestas por XP no son novedosas sino que de alguna forma ya habían sido propuestas en ingeniería del software, o incluso demostrado su valor en la práctica. El mérito de XP es integrarlas de una forma efectiva, y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

3.3.2. Scrum

3.3.2.1. Introducción

El concepto de Scrum [28] tiene su origen en un estudio de 1986, "The New New Product Developement Game (Hirotaka Takeuchi y Ikujiro Nonaka)". Este estudio versaba sobre los nuevos procesos de desarrollo utilizados en productos exitosos en Japón y los Estados Unidos (cámaras de fotos de Canon, fotocopiadoras de Xerox, automóviles de Honda, ordenadores de HP y otros).

Los equipos que desarrollaron esos productos partían de requisitos muy generales, así como muy novedosos, y debían salir al mercado en mucho menos del tiempo del que se tardó en lanzar productos anteriores. Estos equipos seguían patrones de ejecución de proyecto muy similares. En este estudio se comparaba la forma de trabajo de estos equipos altamente productivos y multidisciplinares con la colaboración entre los jugadores de Rugby y su formación de Scrum (melé en español).

Fue ya en 1993, cuando Jeff Sutherland, John Scumniotales y Jeff McKenna concibieron, ejecutaron y documentaron el primer Scrum para desarrollo ágil en desarrollo de software, utilizando el estudio de gestión de equipos de Takeuchi y Nonaka como base en Easel Corporation.

Scrum fue presentado formalmente en 1995 por Jeff Sutherland y Ken Schwaber en el OOPSLA'95. En 2001, Schwaber y Mike Beedle describieron la metodología en el libro Agile Software Development with Scrum.

Actualmente existe, la Guía de Scrum (The Scrum Guide), que representa la guía de conocimiento oficial de Scrum ("Scrum Body Of Knowledge"), escrita por Ken Schwaber and Jeff Sutherland. La última versión data de Febrero de 2010.

3.3.2.2. Descripción

Scrum es una de las metodologías ágiles más utilizadas en la actualidad. Está destinada principalmente al desarrollo y mantenimiento de proyectos software.

Utiliza un proceso iterativo que divide el desarrollo de un producto en ciclos de pocas semanas de duración (*sprints*). En cada uno de los ciclos, el equipo trabaja sobre una lista de requisitos priorizada, dando lugar, al acabar cada ciclo, a un producto entregable. Este enfoque gradual optimiza la previsibilidad y el control de riesgos.

Los principales beneficios que aporta Scrum, son:

- Entrega de resultados con periodicidad quincenal o mensual (los requisitos más prioritarios en ese momento, ya completados) lo cual proporciona grandes ventajas.
- Gestión regular de las expectativas del cliente y basada en resultados tangibles.
- Resultados anticipados (time to market).
- Flexibilidad y adaptación respecto a las necesidades del cliente, cambios en el mercado, etc.
- Gestión sistemática del Retorno de Inversión (ROI).
- Mitigación sistemática de los riesgos del proyecto.
- Incremento de la productividad y la calidad.
- Alineamiento entre el cliente y el equipo de desarrollo.
- Equipo motivado.

Scrum se basa en tres pilares:

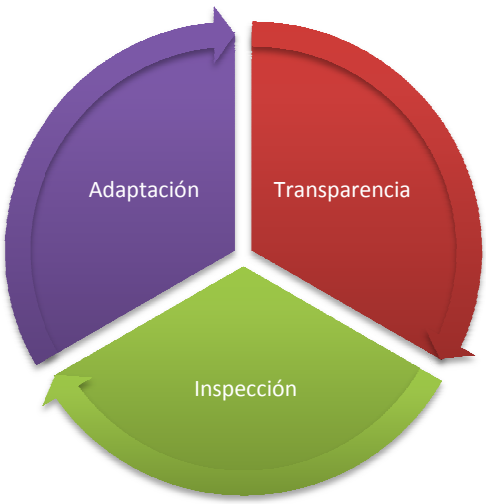


Figura 11 Pilares Scrum

A continuación describiremos los conceptos más importantes en los que se basa Scrum, y se describirá el proceso.

- **El Equipo y sus Roles**

Los equipos de Scrum son equipos multifuncionales y auto gestionados. No se tiene un equipo de desarrollo de bases de datos ni un equipo de testeo. En cada equipo de haber al menos un miembro capaz de realizar cada una de las tareas. Suelen estar compuestos por un número pequeño de integrantes que no suele superar los cinco.

Se dice que el equipo es auto gestionado porque son los miembros del equipo los que estiman las historias al principio de cada sprint y quienes se las van asignando a ellos mismos durante el mismo.

Los roles básicos de Scrum son tres:

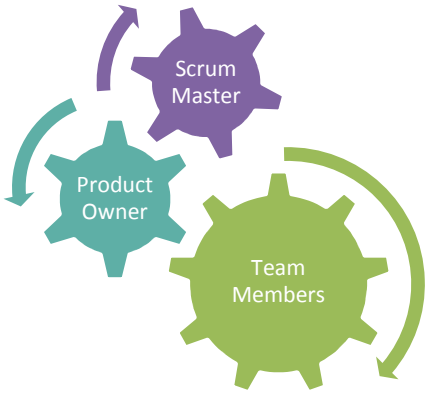


Figura 12 Roles Scrum

- **Scrum Master (facilitador)**
  - El Scrum Master es el responsable de vigilar que el proceso de Scrum se siga de forma correcta. Esto incluye eliminar impedimentos y distracciones que puedan afectar a los miembros del equipo. Se asegura que el equipo es completamente funcional y productivo.
  - Su figura se suele comparar con la de un Project Manager pero es importante comprender que no se trata de un jefe. Su misión es "facilitar" y no "dar órdenes".
- **Dueño del Producto (Product Owner)**
  - El dueño del producto es el cliente. Es el responsable de ir añadiendo requisitos y priorizarlos. Es el responsable de la rentabilidad del producto (ROI).
- **Miembros del Equipo:**
  - Son los desarrolladores, y tienen la misión de realizar el análisis, diseño, implementación y testeo de cada una de las historias de usuario (requerimientos) durante los sprints.
  - Son ellos los que se asignan las tareas a ellos mismos y también los que las estiman su complejidad.

- **La Pila de Producto (backlog)**

La pila de producto es una lista priorizada de las características o requisitos que el dueño del producto quiere que el producto tenga. A cada uno de dichos requisitos se les llama historias de usuario o simplemente historias.

Un ejemplo de pila de producto sería el siguiente:

1. Incluir sindicación de noticias por RSS.

2. Añadir un Web Service para la autenticación con sistemas externos.

3…., etc.

El único miembro con derecho a hacer cambios en la pila de producto es el dueño del producto. Él es el que tiene la responsabilidad de valorar que historias aportan más valor que otras. El resto de miembros del equipo puede realizar sugerencias, pero es el dueño de producto quien las aprueba.

El esfuerzo que requiere cada historia para ser implementada se estima por los miembros de equipo y se representa en puntos o story points.

Los puntos son una forma de indicar como de compleja es una historia y si se va a tardar mucho o poco en implementarla. También es posible estimarlos en horas de trabajo..

- **Indicadores**

Scrum utiliza fundamentalmente, dos indicadores para hacer un seguimiento de la productividad del equipo. Estos son:

- **Velocidad de equipo (team velocity)**
  La velocidad de equipo indica el número de story points u horas de trabajo que el equipo finaliza cada mes.
  Es un indicador directo de la productividad neta del equipo y resulta muy útil para estimar sprints futuros.

- **Factor de Foco (focus factor)**
  El factor de foco de un sprint es el resultado de dividir las horas que se esperaba durasen las tareas de dicho sprint, entre las horas reales que ha tardado el equipo.
  Aunque un miembro del equipo trabaje ocho horas al día, lo normal es que una tarea de ocho horas tarde más de un día en hacerse. Esas ocho horas al día no son 100% productivas.

- **Sprints**

Un sprint es un ciclo de desarrollo durante el cual, los miembros del equipo, trabajan sobre las historias situadas en la cima de la pila del producto.

Normalmente tiene una duración fija de unas dos o cuatro semanas. Es importante que dicha duración sea fija para así poder hacer un seguimiento de cómo de productivo es el equipo.

El sprint comienza con un sprint planning. Durante el mismo se realizan scrum meetings a diario, y al finalizar el sprint se realiza una demostración del producto y una retrospectiva.

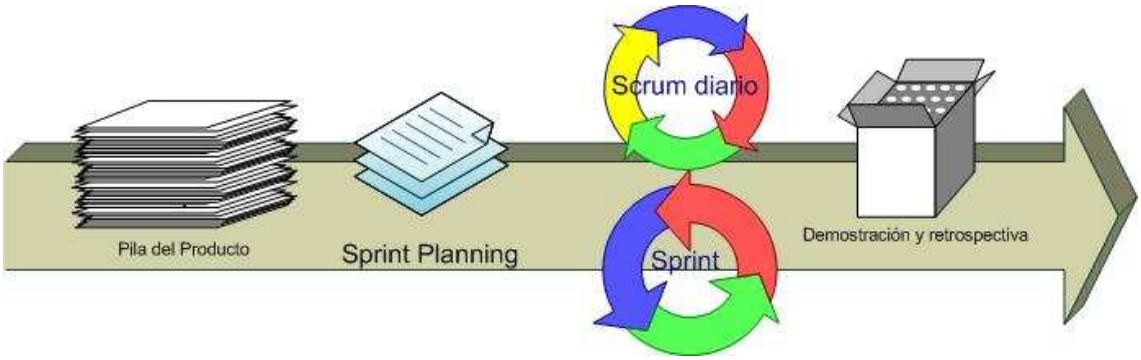


Figura 13 Flujo Scrum

Durante el sprint no se deben añadir tareas a la cima de la pila de producto ni alterar las prioridades.

Es posible, sin embargo, que durante un sprint aparezcan tareas que deban ser resueltas de forma inmediata. Estos son las historias inesperadas y hay que procurar que surjan con la menor frecuencia posible porque afectan muy negativamente a la productividad del equipo.

- **Sprint Planning**

  El sprint planning es una reunión en la cual el equipo se compromete a realizar las n primeras historias de la pila de producto durante el siguiente sprint.

  Para ello, el equipo hace una estimación en story points de dichas historias, y basándose en la velocidad de equipo (a través de sprints anteriores), se decide cuantas historias se va a comprometer el equipo a realizar.

  Dado que usamos la velocidad de equipo de sprints anteriores para estimar sprints futuros es importante que todos estos tengan la misma duración.

  También es posible decidir a cuantas historias se va a comprometer el equipo en base a las horas de trabajo en vez de story points. Para ello, se subdividen las historias en tareas y se estima cuantas horas se va a tardar en implementar cada una, posteriormente se calculan cuantas horas productivas tiene el equipo en ese sprint, multiplicando has horas laborables por el factor de foco y en base al resultado se cogen más o menos historias.

  A la hora de seleccionar el número de historias del sprint, es importante tener en cuenta que durante el desarrollo del sprint puede que surjan historias inesperadas. De forma que conviene dejar un determinado número de horas libres para imprevistos. Es posible que durante el sprint, el equipo acabe todas las historias antes de tiempo, en ese caso el equipo debe continuar trabajando sobre la pila de producto.

- **Burndown Chart**

El burndown chart es una gráfica que muestra la evolución diaria del número de horas o story points restantes para finalizar todas las tareas de ese sprint.

Es una herramienta esencial dentro de Scrum puesto que aporta mucha información de forma visual.

El burndown consta de dos líneas. La primera, es una recta que representa la evolución teórica de las horas o story points restantes para finalizar todas las historias de ese sprint. Esta línea estará más inclinada conforme mayor sea la velocidad de equipo.

La segunda línea, representa la evolución real. Para calcular esta segunda recta, es importante que cada miembro del equipo actualice las tareas en las que ha trabajado a diario, indicando cuantas horas quedan para que estén terminadas. Si, conforme avanza el sprint, observamos que la línea real avanza por encima de la teórica, significa que el equipo avanza más lentamente de lo esperado y tendrá que trabajar más si quiere finalizar el sprint satisfactoriamente .Si por el contrario, se encuentra por debajo, están produciendo más de lo esperado. Dado que es posible que el equipo siga trabajando sobre la pila de producto a pesar de haber finalizado todas las historias a las que se comprometieron para dicho sprint, es posible que para un determinado sprint, el burndown continúe por debajo de cero y finalice con un valor negativo.

Otra forma de expresar lo mismo es subiendo la línea de la evolución real para que siempre quede por encima de cero. En este caso, la diferencia entre las horas o story points a los que se ha comprometido el equipo, y las que ha producido nos lo da la diferencia entre la línea real y la ideal al inicio del sprint.

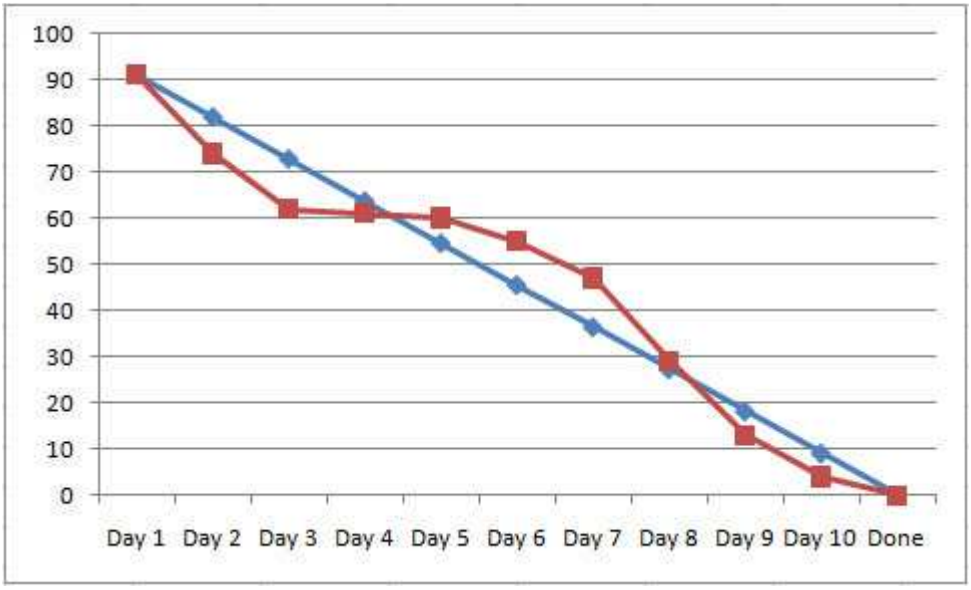


Figura 14 Burndown Chart

- **Scrum Diario**

El scrum diario es una reunión diaria en la que cada miembro del equipo responde a tres cuestiones.

- ¿Qué hiciste ayer?
- ¿Qué vas a hacer hoy?
- ¿Hay obstáculos en tu camino?

También se le conoce como scrum meeting o daily scrum. Se suele celebrar de pie y no debe durar más de 15 minutos.

Todo el mundo está invitado, pero sólo los miembros del equipo, Scrum Master y Dueño del Producto pueden hablar.

Esta reunión favorece mucho la visibilidad sobre el proyecto y permite detectar posibles problemas de inmediato.

- **Demostración (Sprint review)**

Al final de cada sprint se realiza una demostración de las nuevas funcionalidades que se han añadido ante el dueño del producto. De esta forma, el dueño del producto hace un seguimiento exhaustivo y puede añadir tareas al a pila de producto antes del siguiente sprint.

- **Retrospectiva (Sprint retrospective)**

Periódicamente, se revisa lo que funciona y lo que no. No suele durar más de 15 o 30 minutos y se realiza al finalizar el sprint. Participa todo el equipo (Scrum Master, Product Owner, y equipo).

3.3.3. Crystal Methodologies

3.3.3.1. Introducción

Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. Han sido desarrolladas por Alistair Cockburn.

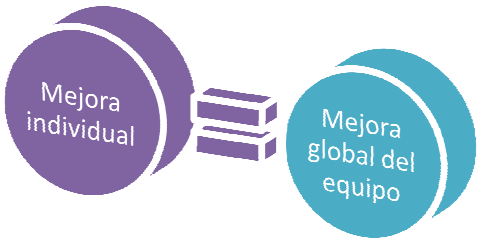
3.3.3.2. Descripción

Crystal Methodologies [5] da una importancia vital a las personas que componen el equipo de un proyecto, y por tanto sus puntos de estudio son:

- Aspecto humano del equipo
- Tamaño de un equipo
- Comunicación entre los componentes
- Distintas políticas a seguir
- Espacio físico de trabajo

Características del equipo

Aconseja que el tamaño del equipo sea reducido, este factor mejora la comunicación entre los miembros del equipo del proyecto. También destaca que un mismo lugar de trabajo disminuye el coste de la comunicación.



Equipo Crystal

Diferentes políticas de equipo

Se utilizarán políticas diferentes para equipos diferentes. Crystal Methodologies utiliza una codificación por colores Codificación por colores dependiendo del tamaño del equipo:

Figura 16 Codificación por colores Crystal Methodologies

Roles

Se describen los siguientes roles:

- Executive Sponsor (Patrocinador Ejecutivo)
- Project Manager (Jefe de Proyecto)
- Domain Expert (Experto en el Dominio)
- Usage Expert (Experto de uso)
- Designer-Programmer (Programador Diseñador)
- UI Designer (UI Diseñador)
- Tester (Realizador de Pruebas)
- Technical (Programador Técnico)

Metodologías Crystal más conocidas:

Crystal Clear	•Se corresponde con el color Blanco en la codificación de colores de Crystal •3-8 personas
Crystal Orange	•Se corresponde con el color Naranja en la codificación de colores de Crystal •25-50 personas

Tabla 12 Tipos Metodologías Crystal

3.3.4. Adaptive Software Development

3.3.4.1. Introducción

Es una metodología ágil orientada al desarrollo de software.

Fue creada por Jim Highsmith en su libro "Adaptive Software Development" [49].

Sus principales características son:

- Iterativo
- Orientado a los componentes software más que a las tareas
- Tolerante a los cambios

3.3.4.2. Descripción

El ciclo de vida que propone tiene tres fases esenciales:

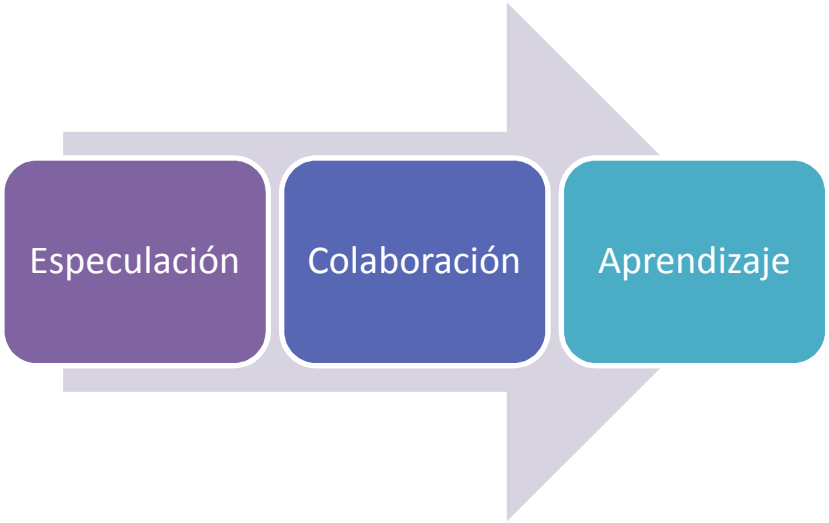


Figura 17 Fases ASD

- Especulación: En esta fase se inicia el proyecto y se planifican las características del software.
- Colaboración: Se desarrollan las características del producto.
- Aprendizaje: Se revisa la calidad, y se entrega al cliente. La revisión de los componentes nos sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo (lecciones aprendidas).

3.3.5. Feature-Driven Development (FDD)

3.3.5.1. Introducción

FDD es una metodología ágil creada inicialmente por Jeff De Luca para satisfacer las necesidades específicas de un proyecto de desarrollo software que debía realizarse en 15 meses con 50 personas para un gran banco de Singapur en 1997.

La descripción de FDD apareció por primera vez en el libro *"Java Modeling in Color with UML"* [50], en 1999. Posteriormente, en 2002, se publicó el libro *"A Practical Guide to Feature-Driven Development"* [51], donde se describía de forma más precisa la metodología.

Se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorizar.

Las iteraciones se deciden en base a "features" o funcionalidades, con significado para el cliente. A diferencia de otros procesos ágiles no cubre todo el ciclo de vida sino sólo las fases de diseño y construcción. No requiere un modelo específico de proceso y se complementa con otras metodologías. Enfatiza cuestiones de calidad y define claramente entregas tangibles y formas de evaluación del progreso.

3.3.5.2. Descripción

FDD describe los siguientes roles:

- **Gerente del proyecto**: es quien tiene la última palabra en materia de negocio, planificación y asignación del personal.
- **Arquitecto jefe**: este rol puede dividirse en arquitecto de dominio y arquitecto técnico.
- **Gerente de desarrollo**: puede combinarse con arquitecto jefe o gerente de proyecto, se encarga de resolver los conflictos dentro del equipo de desarrollo.
- **Programador jefe**: es la persona que participa en el análisis de requerimientos y selecciona los que se van a desarrollar en la siguiente iteración.
- **Propietarios de clases:** trabajan bajo la supervisión del programador jefe en diseño, codificación, prueba y documentación, repartidos por requerimientos (rasgos).
- **Experto de dominio**: que puede ser un cliente, patrocinador, analista de negocio o una mezcla de todos.
- **Administrador de entrega**: controla el progreso del proceso revisando los informes del programador jefe y manteniendo reuniones (breves) con él, informa al gerente de proyecto.
- **"Guru" de lenguaje**: conoce a la perfección el lenguaje de programación y la tecnología.
- **"Herramientista" (toolsmith):** construye pequeñas herramientas de desarrollo o mantiene las bases de datos y sitios web.
- **Administrador del sistema**: controla el ambiente de trabajo, servidores, red…etc.
- **"Tester":** verificador del sistema producido.
- **Escritores de documentos técnicos**: se dedican a la realización de la documentación técnica del proyecto

Un miembro del equipo puede tener otros roles a cargo, y un solo rol puede ser compartido por varias personas.

FDD consiste en cinco procesos secuenciales durante los cuales se diseña y construye el sistema.

La parte iterativa soporta desarrollo ágil con rápidas adaptaciones a cambios en requerimientos y necesidades del negocio.

Cada fase del proceso tiene un criterio de entrada, tareas, pruebas y un criterio de salida.

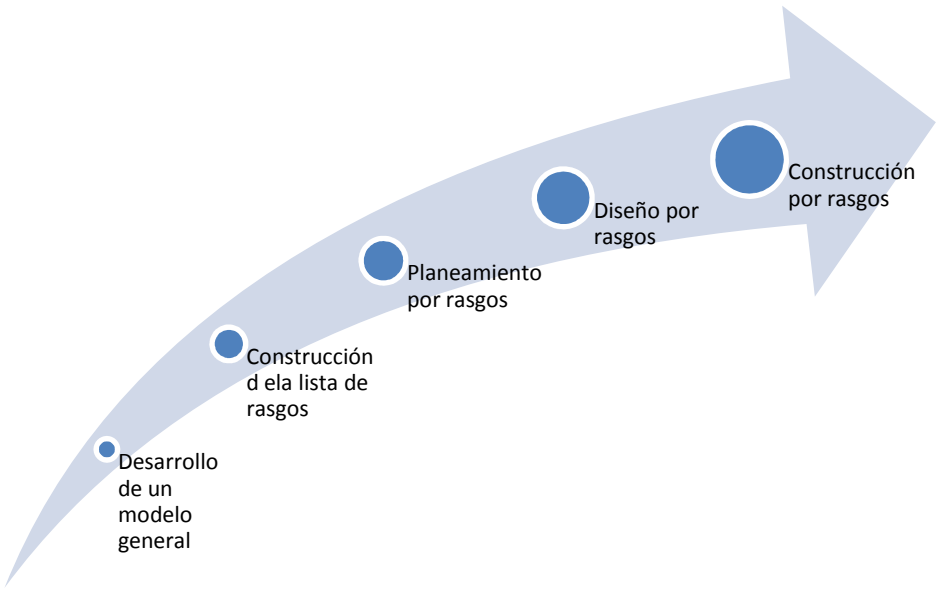


Figura 18 Fases FDD

- **Desarrollo de un modelo general:** Cuando comienza el desarrollo, los expertos de dominio están al tanto de la visión, el contexto y los requerimientos del sistema a construir. A esta altura se espera que existan requerimientos tales como casos de uso o especificaciones funcionales. Los expertos de dominio presentan un informe (ensayo) para que los miembros del equipo y el arquitecto jefe conozcan la descripción de alto nivel del sistema.

  El dominio general se subdivide en áreas más específicas y se define un ensayo más detallado para cada uno de los miembros del dominio. Luego, un equipo de desarrollo trabaja en pequeños grupos para producir modelos de objetos de cada área de dominio. Simultáneamente, se construye un gran modelo general para todo el sistema.

- **Construcción de la lista de rasgos:** Los ensayos, modelos de objeto y documentación de requerimientos proporcionan la base para construir una amplia lista de rasgos.
  Estos rasgos son pequeños ítems útiles a los ojos del cliente. La lista de rasgos es revisada por los usuarios y patrocinadores para asegurar su validez y exhaustividad, los rasgos que requieran de más de diez días se descomponen en otros más pequeños.

- **Planeamiento por rasgos:** Incluye la creación de un plan de alto nivel, en el que los conjuntos de rasgos se secuencian conforme a su prioridad y dependencia, y se asigna a los programadores jefes.

- **Diseño por rasgos y Construcción por rasgos:** Se selecciona un pequeño conjunto de rasgos del conjunto, y los propietarios de clases seleccionan los correspondientes equipos dispuestos por rasgos. Se procede luego iterativamente hasta que se codifican los rasgos seleccionados. Una iteración puede durar desde unos pocos días, hasta un máximo de dos semanas. El proceso iterativo incluye inspección de diseño, codificación, pruebas unitarias, integración e inspección de código.

Algunos precursores del desarrollo ágil sienten que FDD es demasiado jerárquico para ser un método ágil, porque demanda un programador jefe, quien dirige a los propietarios de clases, quienes dirigen equipos de rasgos.

3.3.6. Dynamic Systems Development Method (DSDM)

3.3.6.1. Introducción

La metodología ágil DSDM [5] fue desarrollada en el Reino Unido en los años 90 por un consorcio de proveedores y de expertos en la materia del desarrollo de sistemas de información liderado por Tony Mobbs, combinando sus experiencias de mejores prácticas.

Este consorcio es una organización sin ánimo de lucro y proveedor independiente, que posee y administra el framework.

En febrero de 1995 se publicó la primera versión, siendo la 4.2., la actual (liberada en 2006).

DSDM se centra en los proyectos TI que son caracterizados por presupuestos y agendas apretadas. DSDM trabaja sobre problemas que ocurren con frecuencia en el desarrollo de los sistemas de información, cómo tiempo, presupuesto, falta de implicación del usuario, de la gerencia, etc.

DSDM es un método que provee un framework para el desarrollo ágil de software, apoyado con la continua implicación del usuario en un desarrollo iterativo y creciente que sea sensible a los requerimientos cambiantes, para desarrollar un sistema que reúna las necesidades de la empresa en tiempo y presupuesto.

3.3.6.2. Descripción

Esta metodología tiene como principios fundamentales:

- Participación activa del usuario.
- El equipo toma decisiones.
- Frecuentes entregas del producto.
- Ajuste a los objetivos del negocio.
- Desarrollo iterativo e incremental.
- Cambios reversibles.
- Especificar requerimientos globales.
- Pruebas integradas durante todo el ciclo de vida.
- La cooperación entre el equipo, usuarios e interesados es esencial.

DSDM consiste en 3 fases:

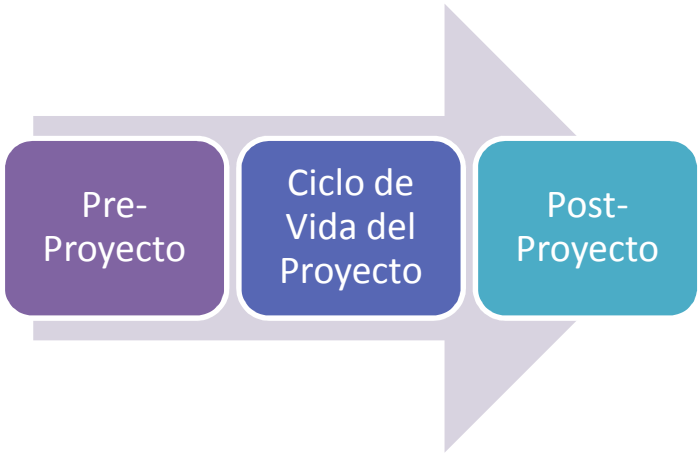


Figura 19 Fases DSDM

La fase del ciclo de vida del proyecto se subdivide en 5 etapas:

- Estudio de viabilidad
  - Calcular los costes
  - Ver si es técnicamente viable
  - Asegurarse de que DSDM sea el enfoque adecuado
- Estudio de negocio
  - Modelado del proceso del negocio
  - Fuerte colaboración cliente-equipo de desarrollo.
- Iteración del modelo funcional
  - Refinar aspectos funcionales del negocio

- Diseño e iteración de la estructura
  - Identificación
  - Planificación
  - Producción
  - Validación
- Implementación
  - Identificación
  - Planificación
  - Producción
  - Validación

DSDM utiliza las siguientes técnicas:

- **TimeBoxes (Cajas de tiempo):** La rapidez de DSDM se basa en seleccionar las funcionalidades más prioritarias para el negocio. El mecanismo para manejar esto en DSDM es el timebox.

- **MoSCoW Rules:** Para dar prioridades a los requisitos DSDM usa las MoSCoW rules.
  - Tenemos 4 clases de requisitos:
    M → "Must Have": vitales para el proyecto
    o
    S → "Should Have": para obtener el máximo beneficio
    C → "Could Have": deben implementarse si el tiempo lo permite
    o
    W → "Won't Have": pueden dejarse para otro momento

- **Prototipados:** El prototipado evolutivo es una de las técnicas en las que se basa DSDM. Encontramos los siguientes prototipos :
  - Bussines
  - Usability
  - Performance
  - Capability

3.3.7. Lean software development

3.3.7.1. Introducción

Lean Software Development [15] es una metodología ágil que tiene origen en el libro del mismo nombre, escrito por Mary Poppendieck y Tom Poppendieck en el año 2003. En él se presentan los principios de Lean aplicados, así como un conjunto de 22 instrumentos, herramientas y las comparaciones con otras prácticas ágiles

La metodología se basa en los principios de Lean Manufacturing. Lean es la forma de producir bienes mediante la eliminación de despilfarros y la implantación de un flujo, que surge como contraposición a la fabricación en cadena basada en un procesado y encolamiento masivo vigentes hasta la fecha, y es una derivación del método de producción de Toyota (TPS).

3.3.7.2. Descripción

Es una metodología ágil que se basa en un proceso iterativo para la construcción del producto.

Existen siete principios en Lean Software Development que vienen de los siete principios de "Lean Thinking". No son recetas especiales para desarrollo de software, pero sin dan unas guías muy interesantes para la forma de desarrollar software.

Estos siete principios son:

PRINCIPIOS
Eliminar los residuos
Ampliar el aprendizaje
Decidir lo más tarde posible
Entrega rápida
Potenciar el equipo
Crear integridad
Véase en conjunto

Figura 20 Principios Lean Thinking

1. **Eliminar los residuos**.

   Se considera residuo, todo aquello que no aporta valor desde la perspectiva del cliente. Y se busca como objetivo eliminar estos residuos.

Se pueden considerar como residuos:

- Inventario de trabajo a "medio hacer".
- Procesos que no aportan valor
- Funcionalidades adicionales que no ha pedido expresamente el cliente
- Tiempo de cambio entre tareas (cada miembro del equipo no debería hacer más de una tarea a la vez).
- Tiempos muertos
- Documentación o material escrito a mano.
- Defectos

2. **Ampliar el aprendizaje**.

   El desarrollo de software es un proceso continuo de aprendizaje. Los miembros del equipo deben poder experimentar y aprender de sus errores. La retroalimentación es una parte muy importante del proceso.

3. **Decidir lo más tarde posible**.

   El desarrollo de software está siempre asociado con cierto grado de incertidumbre, los mejores resultados se alcanzan con un enfoque basado en opciones, lo que retrasa las decisiones tanto como sea posible, hasta que estas se basen en hechos y no en suposiciones y pronósticos inciertos.
   Cuanto más complejo es un proyecto, más posibilidades de cambios existen.

   El enfoque iterativo promueve este principio, la capacidad de adaptarse a los cambios y corregir los errores, lo que podría ser muy costoso si se descubre después de la entrega completa del sistema.

4. **Entrega rápida**.

   Cuanto antes el producto final se entrega sin defectos considerables, más pronto se pueden recibir comentarios, y se incorporan en la siguiente iteración.

   Cuanto más cortas sean las iteraciones, mejor es el aprendizaje y la comunicación dentro del equipo. Sin velocidad, las decisiones no pueden ser postergadas.

   La velocidad puede asegurar que se cumplan las necesidades actuales del cliente y no lo que este requería para el día anterior. Esto les da la oportunidad de pararse a pensar lo que realmente necesitan, hasta que adquieran un mejor conocimiento. Los clientes valoran la entrega rápida de un producto de calidad.

5. **Potenciar el equipo**.

   El equipo necesita algo más que la lista de tareas y la seguridad de que no será alterada durante la realización de las tareas. Las personas necesitan motivación y un propósito para el cual trabajar.

   - El equipo debe poder elegir sus propios compromisos.
   - Los desarrolladores deben tener acceso a los clientes.
   - El jefe del equipo debe proporcionar apoyo y ayuda en situaciones complicadas

6. **Construir con integridad**.

   Existen dos tipos de integridad, la integridad que se percibe, y la integridad conceptual.

   La integridad que se percibe, es la integridad que percibe el cliente. Es lo que quiere el cliente aunque no sepa como pedirlo.

   En cambio, la integridad conceptual se refiere a las parte software del sistema, que tienen que funcionar de una forma integrada, armónica, etc. Es la integridad "técnica".

7. **Véase el conjunto**

   Los sistemas software no son simplemente la suma de sus partes, sino también son el producto de sus interacciones.
   Los defectos en el software tienden a acumularse durante el proceso de desarrollo por la descomposición de las grandes tareas en pequeñas tareas, y por la normalización de las diferentes etapas de desarrollo.

   Cuanto mayor sea el sistema, más serán las personas que participan en su desarrollo y más partes son las desarrolladas por diferentes equipos, por lo que mayor es la importancia de tener bien definidas las relaciones entre todos los actores, con el fin de producir una buena interacción entre los componentes del sistema.