Ingeniería del Software

Resumen de Nico Matovelle

Índice

Tema 1 – El producto

Definiciones

El software, según **Pressman**, se define en tres grandes conjuntos:

- El **conjunto de instrucciones** de ordenador que cuando se ejecutan proporcionan la función y el rendimiento deseado.
- Las **estructuras de datos** que facilitan a los programas manipular la información.
- Los **documentos** que describen el funcionamiento y la forma de uso de los programas.

Sommerville, en cambio, lo define como "los programas de cómputo y su documentación asociada".

Ingeniería del software – es una disciplina de la ingeniería que tiene que ver con todos los aspectos de la producción de software, desde su inicio hasta que se sustituye. Al ser una ingeniería, implica la aplicación de métodos, uso de herramientas, técnicas... de forma sistemática y organizada. Éstos se aplican en todos los aspectos de la producción: los técnicos, la gestión de recursos humanos, la gestión del tiempo, el coste, la calidad...

Atributos del buen software – un software de calidad debe ser:

- **Confiable**: no debe causar daños físicos ni económicos en caso de fallo. Siempre debería ser seguro, fiable y estar protegido frente a errores.
- **Eficiente**: no debe desperdiciar recursos del sistema.
- **Usable**: debe contar con interfaces y documentación adecuadas a sus usuarios.
- **Mantenible**: el software tiene que poder evolucionar y seguir cumpliendo sus especificaciones.

  "CoEfUMa"

Preguntas "sin resolver" de la ingeniería de software (al final del tema 2 se pueden resolver)

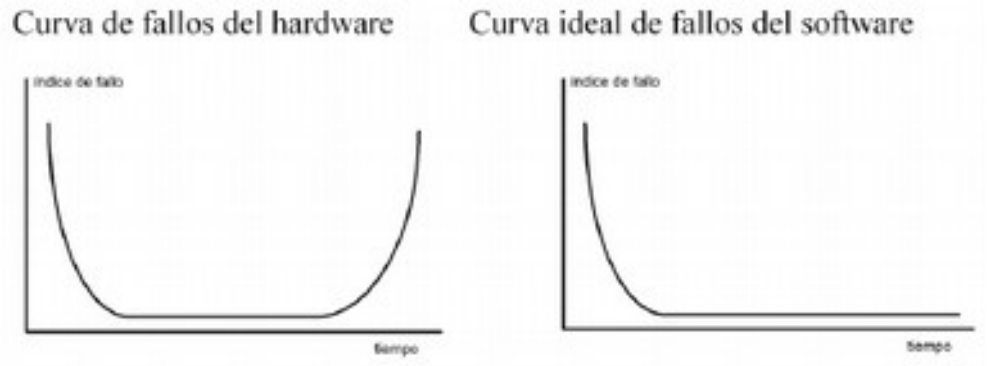
- ¿Por qué lleva tanto tiempo terminar los programas?
- ¿Por qué es tan elevado el coste?
- ¿Por qué no es posible encontrar todos los errores antes de entregar el software al cliente?
- ¿Por qué resulta tan difícil constatar el progreso conforme se desarrolla el software?

Desarrollo de software – es la secuencia de actividades que conducen a la elaboración de un producto software. Fundamentalmente se puede subdividir en **especificación** del software, **desarrollo**, **validación** y **mantenimiento**.

Características del software

"El software se degrada, pero se desarrolla (a medida si se quiere) y se reutiliza"

- **No se fabrica, se desarrolla** – esto implica que el coste de desarrollo de la primera unidad es superior a los costes de producción de otros campos, pero el coste unitario decrece hasta casi hacerse imperceptible en comparación con cualquier otra ingeniería.
- **No se estropea, se degrada** – a diferencia de los otros productos que empiezan con una alta tasa de fallos y la reducen hasta que empieza a crecer otra vez, el software ideal solo tiene la primera pendiente: empieza con una alta tasa de fallos y la reduce hasta estabilizarse para siempre. Pero se degrada, por lo que con el tiempo y con las actualizaciones y mejoras necesarias, va aumentando y reduciendo su tasa de fallos. Normalmente, tras una mejora no se llega a alcanzar el bajo nivel de fallos de las anteriores versiones.



- **Se puede desarrollar a medida** (personalizado) – en este aspecto se pueden encontrar distintos tipos de software:
  - **Productos genéricos**: software enlatado, producido por una organización para introducir en el mercado dirigido a tipos de clientes. Suponen la mayor parte del gasto.
  - **Productos a medida**: son los desarrollados bajo pedido. En el desarrollo de estos se produce el mayor esfuerzo.
  - **ERPs (Enterprise Resource Plannings)**: soluciones genéricas tuneadas para adaptarse a problemas particulares
- **Es reutilizable** – dispone de activos (o artefactos), que son secciones de un proyecto que se podrían reutilizar en otros proyectos. Estos bloques reutilizables son:
  - **Bibliotecas** – son un conjunto de subrutinas temáticas de las que disponen la mayoría

de los lenguajes modernos.

- **Módulos** o componentes – estos se introdujeron con la utilización de técnicas de programación estructurada y modular, pero al dedicarse poco esfuerzo al diseño de módulos suficientemente generales para ser reutilizables, al final tampoco se suelen reutilizar.
- **Frameworks** – se trata de conjuntos de métodos y objetos que también pueden cumplir funciones generales.
- **Patrones de diseño** – ya se explicaron en Diseño de software, el empleo y la reutilización de estos patrones también puede facilitar que se reutilicen otros artefactos relacionados.
- Un aspecto que todavía no se llega a reutilizar son las **listas de requisitos** o especificaciones que un cliente necesita.

La reutilización reduce los costes, aumenta la productividad, mejora la calidad (tanto del software como del mantenimiento) y facilita el control y la planificación. Los problemas son que suele tener como consecuencia programas menos óptimos (al usar bloques generales), inversiones iniciales cuantiosas, reestructuraciones en la organización y en el proceso de producción y que la recuperación de activos también conlleva trabajo.

Pueden seguirse **estrategias que favorezcan la reutilización**: mirando qué es reutilizable en cada proyecto, difundiendo tal información y obligando al reuso mostrando cómo funcionan los activos ya programados. En el futuro podría existir un repositorio de activos de cualquier tipo que facilitase esto todavía más.

El software heredado

Se trata de un un tipo de software generado hace bastante tiempo y que se ha seguido usando sin estar documentado totalmente y sin que el programador que lo desarrolló esté accesible.

Características

- Es **crítico** para las empresas que lo usan debido a su antigüedad.
- Ha sido **modificado** de forma **continuada** para ser adaptado a nuevos entornos (hardware, red, sistemas de bases de datos...) o para implementar nuevas funcionalidades.
- Suelen tener **poca calidad**.
  - Tiene un **diseño complejo** o carece del mismo.
  - Contiene secciones de **código oscuro** (que no se conoce muy bien cómo funciona).
  - Tiene una **documentación muy pobre** o inexistente (tanto en instrucciones, changelog y pruebas)

Consecuencias

- El **mantenimiento** de este software es muy **costoso** debido a la inexistencia de la documentación cuando un programador nuevo lo modifica.
- Su evolución tiene un **alto riesgo para la empresa** debido a la posibilidad de fallo.

Tipos de software

El software y los problemas que aborda pueden clasificarse de distintas formas:

Tipos de problemas

- Con solución por **pasos específicos**: pueden usarse algoritmos, suelen resolverse mediante lenguajes procedimentales.
- Que **pueden definirse formalmente**: en estos no necesitamos definir cómo resolver el problema, si no definir el problema en sí, para lo que suelen usarse lenguajes declarativos, como SQL.
- Problemas con **soluciones concretas**: pero cuya solución general la desconocemos, se resuelven con inteligencia artificial mediante redes neuronales.
- Problemas basados en **conocimiento heurístico**: se usan sistemas expertos, basados en el análisis de reglas que pueden llegar a ser contradictorias.

Software clasificado por categoría (Pressman)

- **Empotrado**: son los que se encuentran en la memoria ROM de los sistemas para lavadoras, hornos, coches...
- **De sistemas**: son los que hacen de interfaz entre programas y programas o entre programas y hardware. Sistemas operativos, compiladores...
- **De aplicación**: son los que resuelven una necesidad específica. Software a medida.
- **Científico y de ingeniería**: se caracteriza por sus algoritmos devoradores de números, empleados básicamente para cálculo numérico, aunque la tendencia es que se aproximen a la categoría de software de sistemas.
- **De línea de productos**: son los que se diseñan para una capacidad específica y la utilización de muchos clientes diferentes (Office).
- **De aplicaciones basadas en la web**: empezaron como un conjunto de archivos de hipertexto ligados, pero cada vez evolucionan más para ofrecer muchos más tipos de funcionalidades.
- **De inteligencia artificial**: utiliza algoritmos no numéricos para resolver problemas inabordables mediante análisis directos. Son la robótica, sistemas expertos, juegos...

Clasificaciones de Sommerville

- Por **estructura** (orientados a...): función, componentes, listas y objetos.
- Por **función**: programas (o sistemas de usuario), interfaces Hombre-Máquina, herramientas de software, librerías, sistemas de uso genérico (compiladores, SO's...), bases de datos y sistemas basados en web.
- Por **plataforma de cómputo**: sistemas embebidos, sistemas de cómputo distribuido, sistemas de cómputo paralelo, sistemas de tiempo real, sistemas basados en chips, sistemas wereables y sistemas de cómputo ubicuos.

Problemas del desarrollo del software

Como todas las producciones, el desarrollo del software también puede tener problemas.

- **Planificación y estimación de costes imprecisos** - Es muy difícil estimar costes si no hay realizaciones previas ni mediciones, por eso la **primera norma del CMMI es medir**. Otro tema causante de problemas es que los responsables de proyectos no suelen ser expertos informáticos y cuando lo son, no suelen tener conocimientos de gestión.
- **Baja productividad** causada por mayor duración de la esperada - Suele darse cuando las especificaciones son ambiguas o incorrectas, provocando el realizar modificaciones sobre la marcha y falta de documentación. Las soluciones a esto son la **administración de requisitos**, las metodologías ágiles y la gestión de la configuración administrada mediante la gestión del cambio.
- **Mala calidad a la entrega del producto** - Para evitar este problema se ideó la existencia del proceso del **aseguramiento de la calidad** (que se realiza durante todo el desarrollo del proyecto), así como los procesos de verificación y validación.
- **Rediseño del producto** - En caso de que el cliente no esté satisfecho, aunque los requisitos estuviesen claros, puede ser obligado el rediseño del producto. Para evitar esto, hay que propiciar que **el cliente entienda los requisitos** que necesita para el producto que quiere.

Pese a todos los problemas, las tendencias son buenas: según el CHAOS Report de 2013, en 1994 sólo un 16% de los proyectos acababan en plazo con éxito, mientras que un 31% se cancelaban. En 2012 el porcentaje de éxitos en plazo aumentó hasta un 39%, mientras que el porcentaje de cancelaciones se redujo hasta el 18%.

Leyes de la evolución del software (Leyes de Lehman)

- **Cambio continuado** – Un programa que se usa en un entorno real necesariamente debe cambiar o **se volverá** progresivamente **menos útil** en ese entorno.
- **Complejidad creciente** – A medida que el programa en evolución cambie, su estructura tiende a ser cada vez más compleja. Se deben dedicar recursos extras para preservar y **simplificar su estructura**.
- **Evolución prolongada** del programa – La evolución de los programas es un proceso autorregulativo. Los **atributos** de los sistemas, tales como tamaño, tiempo entre entregas y el número de errores documentados, son aproximadamente **invariables** para cada entrega del sistema.
- **Estabilidad organizacional** – Durante el tiempo de vida de un programa, su **velocidad de desarrollo** es aproximadamente **constante** e independiente de los recursos dedicados al desarrollo del sistema.
- **Crecimiento continuado** – La **funcionalidad** ofrecida por los sistema tiene que **crecer** continuamente para mantener la satisfacción de los usuarios.
- **Decremento de la calidad** – La calidad de los sistemas comenzará a disminuir a menos que dichos sistemas se adapten a los cambios de su entorno de funcionamiento.
- **Realimentación del sistema** – Los procesos de evolución incorporan sistema de

realimentación multiagente y multibucle, lo cual significa que la salida del programa influye en la entrada del mismo. Por ejemplo, los comentarios de los usuarios finales o el uso que le den al programa influirán en el desarrollo del siguiente incremento.

"Los programas, en su evolución, son cada vez **más complejos**, ya que están **cambiando continuamente** a ritmo **constante** creciendo en **funcionalidad** gracias a la **realimentación**, aunque siempre con los **mismos atributos** y decreciendo cada vez más su **calidad**."

Mitos del software

Para la **administración**:

- Ya hay un libro de estándares y procedimientos para construir el software.
- Si un proyecto se atrasa, basta con añadir personal.
- Si subcontrato un proyecto, puedo despreocuparme.

Para el **cliente**:

- Un enunciado general basta para escribir programas.
- Puedo cambiar continuamente los requisitos porque el software es flexible y adaptable.

Para el **desarrollador**:

- Una vez que hago el programa y funciona, terminé.
- Si no se ejecuta el programa, no se puede evaluar su calidad.
- El único producto de mi trabajo que se entrega en un proyecto con éxito es el programa en funcionamiento.
- La I.S. obliga a documentar para hacer el proceso más lento.

Tema 2 – El proceso del software

Conceptos

Ingeniería del software

- Según **Sommerville**, es una **disciplina de la ingeniería** que concierne a todos los aspectos de la producción del software.
- Según **Fritz Bauer**, es el **establecimiento y uso** de principios de ingeniería robustos, orientados a obtener software económico, fiable y que funcione de manera eficiente sobre máquinas reales.
- Según el **IEEE**, es la **aplicación de un enfoque** sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería al software.

Ciclo de vida – sucesión de etapas por las que pasa el software desde que se inicia un proyecto hasta que se deja de usar. Cada una de estas etapas conlleva una serie de tareas a realizar y una serie de documentos (o software) que serán la salida de cada una de estas fases y servirán de entrada para la siguiente fase.

Modelos de procesos – **descripción de los procesos** involucrados en el desarrollo del software sin llegar a precisar cómo se desarrollan. Ejemplos: IEEE 1074, ISO 12207-1, ISO/IEC TR 15504-2.

Procesos – conjunto de actividades y tareas.

Actividad – conjunto de tareas.

Tarea – cualquier acción que transforma una entrada en salidas.

Técnicas – cualquier **recurso** utilizado para llevar a cabo una tarea. Normalmente gráficas con apoyos textuales.

Métodos y/o procedimientos – definiciones de la forma de ejecutar las tareas. Determinan el modo en que se utilizan las técnicas en cada fase del desarrollo.

Herramientas – cualquier software que nos ayude en cualquier etapa del proceso de desarrollo del software. Ejemplo: CASE (Computer-aided software engineering, herramientas de 4ª generación).

Programación estructurada – concepto que surge a nivel académico a finales de los 60, pasando al entorno industrial a mediados de los 70.

Diseño estructurado – surge a mediados de los 70. Se asciende a un nivel de abstracción y se piensa en módulos.

Análisis estructurado – surge a finales de los 70, busca eliminar la especificación narrativa.

Proceso del Software

Según **Pressman**, el proceso del software se divide en tres fases principales:

Fase de definición

Esta es la fase que se centra en **qué es lo que hay que producir**. Se identifican respuestas a preguntas tales como ¿qué información tiene que ser procesada? o ¿qué función y rendimiento son los que se esperan?

En esta fase se distinguen tres actividades:

- **Análisis del sistema**: define el papel de cada elemento relacionado con el sistema informático que se pretende desarrollar, precisando el papel del software en ese sistema.
- **Análisis de requisitos del software**: define la función del software con poco nivel de precisión. Esta tarea debe llevarse a cabo por el desarrollador y el cliente, y como resultado tendrá un documento de especificación de requisitos.
- **Planificación**: plantea la organización de las tareas que se llevarán a cabo en la realización del proyecto.

Fase de desarrollo

Resuelve la cuestión del **cómo se desarrolla tal software**.

- Se empieza con el **diseño**, especificando las estructuras de datos, funciones, arquitectura, detalles, interfaces y procedimientos.
- La siguiente tarea es la **codificación** de toda la especificación generada anteriormente, traducirla a código.
- Por último, se realizan las **pruebas**, lo cual es una actividad que Sommerville clasifica como otra fase.

Mantenimiento

Esta fase se centra en los diferentes cambios a producir en el código. Estos son de distintos tipos:

- **Corrección**: son cambios asociados a errores detectados por el cliente en la aplicación cuando ésta ya se encuentra operativa.
- **Adaptación**: son modificaciones en el programa original para adaptarlo a cambios en su entorno original.
- **Prevención**: cambios que buscan mejorar el programa para que sea más fácil mejorarlo, adaptarlo o corregir sus defectos.

[Salto al Tema 3 – ciclos de vida]

IEEE 1074

Ofrece una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software.

Este estándar proporciona el conjunto de actividades que constituyen los **procesos obligatorios para el desarrollo y mantenimiento del software**. Están organizado en 17 procesos con un total de 65 actividades. Estos procesos se dividen en 4 secciones lógicas:

- **Modelo del ciclo de vida del software**: son los procesos orientados a la selección del ciclo de vida. Éstos orientarán a los demás.
    - No se establece ni define un ciclo de vida específico, pero sí **requiere que se seleccione** y utilice uno, el que mejor se adapte al proyecto en particular.
    - Esta selección se hará en función de todas las variables que la puedan afectar y a las que pueda afectar.
- **Procesos de gestión del proyecto**: procesos que **inician, supervisan y controlan** otros procesos a lo largo del ciclo de vida. Estos procesos son:
    - **Proceso de iniciación**: se crea el ciclo de vida y se establecen los planes para gestionar el proyecto.
    - **Proceso de supervisión y control**: relacionado con el seguimiento, los informes, la gestión del coste, los calendarios, los problemas y el rendimiento.
    - **Gestión de calidad**: trata los procesos de aseguramiento de la calidad, la satisfacción del cliente y la mejora de la calidad.
- **Procesos orientados al desarrollo**: se incluyen los realizados **antes, durante y después** del desarrollo.
    - **Antes**: análisis de la necesidad de un sistema (con valoración de alternativas) y asignación de requisitos al software y al hardware.
    - **Durante**: análisis de requisitos, diseño, implementación y pruebas.
    - **Después**: instalación, soporte al usuario, mantenimiento y retirada.
- **Procesos integrales**: son los necesarios para asegurar la terminación y calidad de los procesos (verificación, validación, gestión de la configuración, documentación y formación).

ISO 12.207–1

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

Procesos principales

Son aquellos que resultan útiles a las personas que inician o realizan el desarrollo, la explotación o el mantenimiento del software durante su ciclo de vida. Estas personas son los stakeholders, es decir: los compradores, los suministradores, el personal de desarrollo, los usuarios y el personal de mantenimiento del software.

- **Adquisición** – Son las actividades y tareas del comprador para preparar y publicar la solicitud de ofertas, seleccionar un suministrador y gestionar desde adquisición a la recepción del producto. Esto último es muy importante, ya que implica que cuando se subcontrata a otra empresa no podemos desentendernos del resultado que ésta ofrezca.

- **Suministro** – Son las actividades del suministrador para preparar una propuesta que responda a una solicitud e identificar los procedimientos y recursos para garantizar el éxito del proyecto.
- **Desarrollo** – Comprende un gran número de actividades.
    - **Análisis de requisitos del sistema**: se especifican los requisitos del sistema, incluyendo las funciones y las capacidades que debe incluir, así como los requisitos y las restricciones.
    - **Diseño de la arquitectura del sistema**: establece la arquitectura de alto nivel, que especificará los principales componentes hardware y software.
    - **Análisis de los requisitos software**: se establecen y se documentan estos requisitos, incluyendo una especificación completa de todas sus características.
    - **Diseño de la arquitectura del software**: transformación de los requisitos del software en una estructura de alto nivel que identifica sus componentes principales.
    - **Diseño detallado del software**: se diseña detalladamente cada componente del software, así como las bases de datos. También se actualizan los manuales de usuario y se documentan los requisitos que deben cumplir las pruebas.
    - **Codificación y prueba del software**: se desarrollan los componentes software y las bases de datos, se prueba que cumplen los requisitos y se actualizan los manuales.
    - **Integración del software**: se integran los componentes del software en una unidad y se prueban según sea necesario. También se actualizan los manuales de usuario.
    - **Prueba del software**: se cualifica el software mediante pruebas en función de los requisitos especificados.
    - **Integración del sistema**: se integran los elementos software, hardware y operaciones manuales.
    - **Prueba del sistema**: análoga a la del software, pero con los requisitos de cualificación del sistema.
    - **Instalación**: del sistema en el entorno de explotación final donde vaya a funcionar. Es recomendable no sustituir al anterior sistema de inmediato, si no mantenerlos paralelamente para comprobar el funcionamiento correcto del nuevo sistema.
    - **Soporte del proceso de aceptación del software**: el desarrollador debe dar su apoyo a la revisión de aceptación y prueba del software por parte del comprador.

      "Centrarse primero en el **sistema** (**analizando** y **diseñando**). Hacer lo mismo con el **software**, pasando luego a un **diseño detallado** y a la **codificación**. **Integrar** y **probar** el **software**, haciendo después lo mismo con el **sistema** completo. Y para terminar, **instalarlo** y dar **soporte**."
- **Explotación** – Son las actividades relacionadas con la **operación y uso** del software, además del soporte operativo a los usuarios. Éstas se aplican al sistema completo.
- **Mantenimiento** – Se relacionan con la modificación del software o la documentación por diversas razones: errores, mejoras, adaptaciones o migración (y retirada) del software.

Procesos de soporte

Sirven de apoyo al resto de procesos y se aplican en cualquier momento del ciclo de vida

- **Documentación** – Registra toda la información producida a lo largo de todo el ciclo de vida. Incluye todas las actividades relacionadas con esta: planificar, diseñar, desarrollar, producir, editar, distribuir y mantener los propios documentos. Además involucra a todas las personas del proyecto: directores, ingenieros, desarrolladores, usuarios...
- **Gestión de la configuración** – Es un conjunto de procesos administrativos y técnicos que se aplican durante todo el ciclo de vida con diversos objetivos.
    - Identificar, definir y establecer la **línea base de los elementos configurables** del software del sistema.
    - Hacer el **control de cambio** de los elementos.
    - Registrar e informar del **estado de elementos** y sus peticiones de modificación.
    - Asegurar que todos los elementos sean **completos, correctos y consistentes**.
    - Controlar el almacenamiento, la **manipulación** y la entrega de los **elementos**.
- **Verificación** – Determina que los **requisitos** están completos y son correctos, además de comprobar que los productos de **cada fase** cumplen esos requisitos y condiciones impuestos sobre ellos en las anteriores. Este proceso puede subcontratarse para que una organización de servicios se responsabilice.
- **Validación** – Determina si el software o **sistema final** cumple con los **requisitos** para su uso. Este proceso también puede subcontratarse.
- **Revisión conjunta** – Sirve para evaluar el estado del software y sus productos en un punto del desarrollo. Estas revisiones se celebran tanto a nivel de gestión como del proyecto.
- **Auditoría** – Permite determinar, en **hitos** predeterminados, si se han cumplido los requisitos, los planes y el contrato.
- **Aseguramiento de la calidad** – Asegura, aportando confianza, que los productos y procesos cumplen los requisitos y se ajustan a lo previsto. Para este proceso se usan los resultados de otros, como son la verificación, la validación... Este proceso puede ser interno o contratarse a una empresa externa.
- **Resolución de problemas** – Consiste en analizar y eliminar problemas descubiertos en el desarrollo o cualquier otro proceso. Tiene como objetivo aportar un medio que asegure que todos los problemas descubiertos se analizan y solucionan.

Procesos de la organización

Son los procesos para la gestión y formación del personal, así como la mejora del proceso. Ayudan a hacer una organización efectiva y tienen lugar a nivel organizativo.

- **Gestión** – Son las actividades relacionadas con la planificación, el seguimiento, el control, la revisión y la evaluación del proyecto.
- **Mejora** – Permite valorar, medir, controlar y mejorar los procesos del ciclo de vida.
- **Infraestructura** – Son las actividades relacionadas con dotar a los procesos de

infraestructura hardware, software, herramientas, técnicas y normas.

- **Formación** – Actividades relacionadas con mantener al personal formado, lo cual incluye proporcionarle materiales y planes de formación.

ISO/IEC TR 15504-2

Es un modelo de referencia para los procesos y la capacidad del proceso. Podría entenderse como una ampliación o **extensión de la ISO-12207-1**, la norma anterior.

Los procesos que añade esta norma con respecto a la anterior son:

Dentro de los **procesos principales**:

- **Obtención de requisitos** – Tiene como objetivo reunir, procesar y seguir la evolución de las necesidades y requisitos del cliente a lo largo de toda la vida del producto.

Dentro de los **procesos de la organización**:

- **Alineamiento de la organización** – Asegura que los individuos en la organización ven, comprenden y entienden igual los objetivos de negocio para que funcionen de forma efectiva.
- **Gestión de recursos humanos** – Proporciona a los individuos de la organización y el proyecto las habilidades y el conocimiento para realizar sus roles de manera efectiva y poder trabajar juntos como un grupo.
- **Medida** – Recoge y analiza los datos relativos a productos desarrollados y a procesos implementados en la unidad organizativa para gestionar de forma efectiva los procesos y demostrar objetivamente la calidad del producto.
- **Reutilización** – Promueve facilita la reutilización de productos de trabajo desde una perspectiva de la organización.

Además, el proceso de gestión de esa categoría ahora se descompone en 4:

- **Gestión** – Se centra solamente en organizar y controlar la iniciación y realización de cualquier proceso dentro de la organización para que logre los objetivos.
- **Gestión del proyecto** – Identificar, establecer, coordinar y controlar las actividades, tareas y recursos necesarios para que el producto producido que cumpla los requisitos.
- **Gestión de la calidad** – Controlar la calidad de los productos y asegurar que satisfacen al cliente. Esto implica enfocar el control de la calidad del producto y el proceso tanto a nivel de proyecto como de organización.
- **Gestión del riesgo** – Identificar y reducir continuamente los riesgos en un proyecto a lo largo de su ciclo de vida. Similar a la gestión de la calidad, implica enfocar el control de la calidad del producto y del proceso tanto a nivel de proyecto como de organización.

Descripción de procesos

Todos los procesos de las normas están descritos por un conjunto de atributos. Estos son:

- **Identificador** del proceso: identifica la categoría del proceso y un número secuencial.

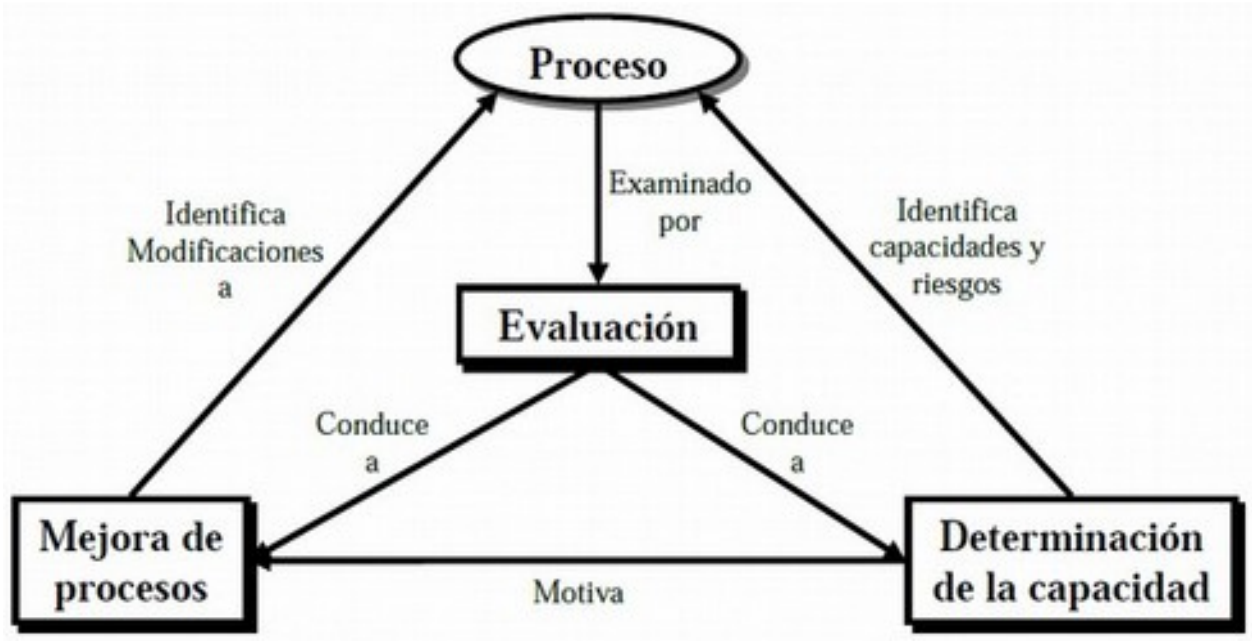
- **Nombre** del proceso: una frase descriptiva.
- **Tipo** de proceso: existen 5 tipos, 3 de alto nivel (básico, extendido y nuevo) y 2 de segundo nivel (componente y componente extendido).
- **Propósito** del proceso: un párrafo que explica el propósito del proceso.
- **Salida** del proceso: la descripción de los resultados observables del proceso.
- **Notas** del proceso: una lista opcional de notas informativas.

Evaluación de procesos software

La crisis del software y los problemas causados por el software heredado son las consecuencias de desarrollar **software sin una metodología** orientada a la calidad. Esto supone la creación de programas de **baja calidad**, el incumplimiento de los presupuestos, requisitos o restricciones y el fracaso del proyecto.

Tras todo esto, las empresas adquirieron un interés por encontrar un **proceso que les garantizase los resultados**, pero a pesar de conocerlo, su aplicación nunca es total o consistente. Como consecuencia, las empresas llegan a desconocer en qué punto se encuentran sus procesos.

De este modo surge una serie de **modelos orientados a guiar a las empresas** en su evolución desde su estado actual hasta uno evolucionado que garantice la optimización en la producción de software de calidad. El modelo guiará a la empresa de la siguiente manera:



CMMI

En inglés Capability Maturity Model Integration, se traduce como **Integración del Modelo de Capacidad de Madurez**. Este modelo presenta dos formas diferentes: un modelo continuo y uno discreto.

La **representación continua** se enfoca en la **capacidad de los procesos** y son las organizaciones las

que eligen las áreas del proceso en las que se quiere incidir para la mejora continua.

La **representación discreta** o por etapas se enfoca en la madurez de la **organización entera** y sigue un camino predeterminado.

Propone **22 áreas del proceso** (PA) divididas en 4 categorías (PAs de **gestión del proceso**, PAs de **gestión del proyecto**, PAs de **ingeniería** y PAs de **soporte**), las cuales pueden estar en uno de los siguientes 6 niveles de capacidad (cada uno siempre implica el cumplimiento del anterior):

0. **Incompleto**: el área de proceso aún no se realiza o todavía no alcanza las metas del nivel 1 de capacidad.
1. **Realizado**: todas las tareas específicas del área del proceso han sido satisfechas. Las tareas de trabajo requeridas para producir el producto específico han sido realizadas. "**Simplemente se cumple**".
2. **Gestionado**: todo el trabajo asociado con el área del proceso se ajusta a una política organizacional definida; toda la gente del proyecto tiene acceso a los recursos adecuados para realizar su labor; los clientes están implicados activamente en el proceso cuando lo requiere y todas las tareas de trabajo están monitorizados y son evaluados de acuerdo a la descripción del proceso. "**Se cumple con normalidad**".
3. **Definido**: el proceso está adaptado al conjunto de procesos estándar de la organización, de acuerdo con las políticas de adaptación de la misma, y contribuye a la las mediciones y otras mejoras del proceso para los activos del proceso organizacional. "**Es costumbre en la empresa**".
4. **Administrado de forma cuantitativa**: el área del proceso se controla y mejora mediante mediciones y evaluación cuantitativa. "**Se puede saber con exactitud numérica en qué grado se cumple**"
5. **Optimizado**: el área del proceso se adapta y mejora mediante el uso de medios estadísticos para conocer las necesidades cambiantes del cliente y mejorar de manera continua la eficacia del área del proceso que se está considerando. "**Siempre se intenta pulir detalles**".

Componentes de las PA

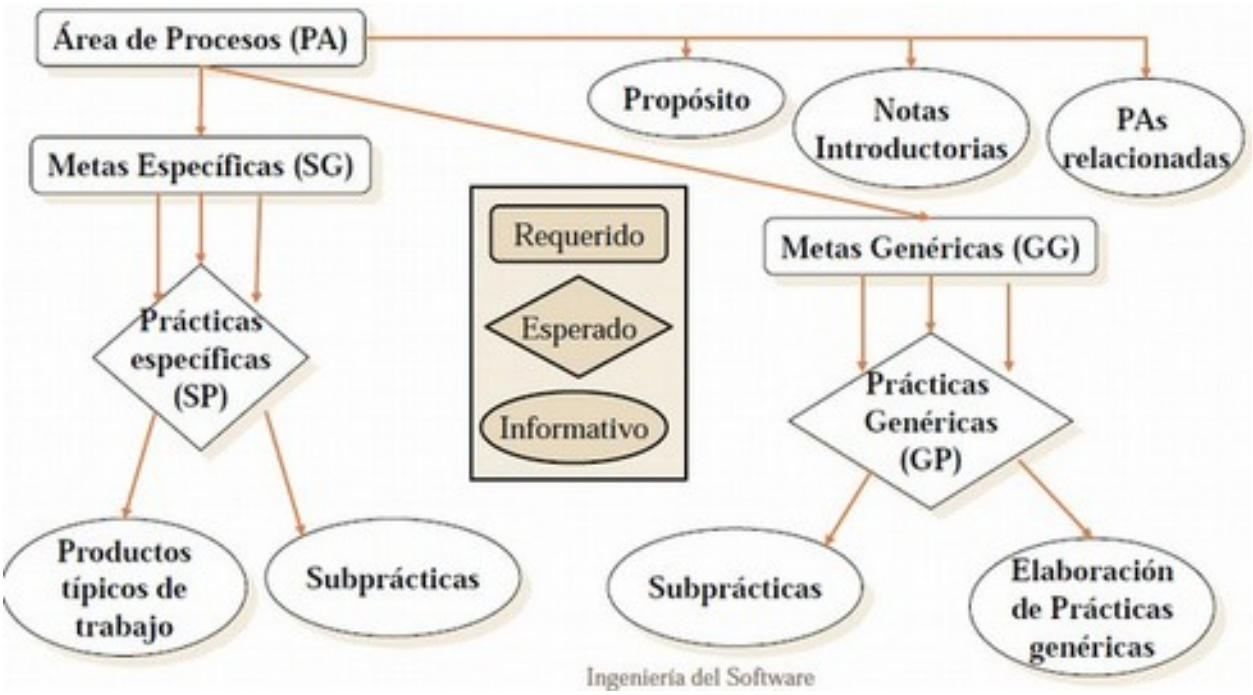
El CMMI define cada área de proceso en función de "**prácticas específicas**" (PE) requeridas para alcanzar una serie de "**metas específicas**" (ME). Las metas específicas establecen las características que deben existir para que las actividades implicadas por un área de proceso sean efectivas. Las prácticas específicas convierten una meta en un conjunto de actividades relacionadas con el proceso.

Además de las metas y prácticas específicas, el CMMI también define una serie de cinco **metas genéricas** (MG), con sus prácticas genéricas (PG), relacionadas con cada área del proceso. Cada una de ellas se corresponde con un **nivel de capacidad**, por lo que para lograr un nivel de capacidad particular se debe alcanzar la meta genérica para ese nivel y las prácticas genéricas que corresponden a esa meta. Estas metas genéricas son:

1. Alcanzar las **metas específicas**
2. Institucionalizar un proceso **gestionado**
3. Institucionalizar un proceso **definido**

4. Institucionalizar un proceso **cuantitativamente gestionado**
5. Institucionalizar un proceso en **optimización**

Así, un área de procesos puede entenderse de la siguiente manera:



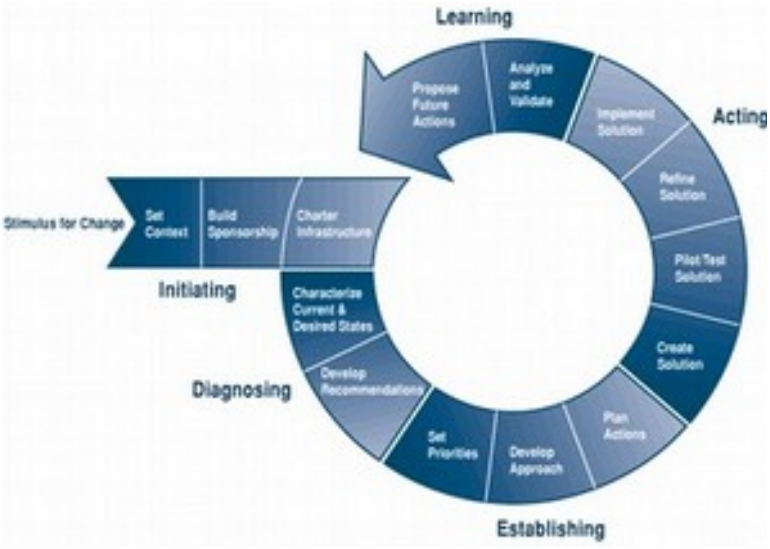
Beneficios del CMMI

Está demostrado que el CMMI ofrece una gran **mejora en el funcionamiento** de la organización. Reduce costes, plazos, aumenta la productividad, la calidad, la satisfacción del cliente... Y pese a que pueda tener un gran coste adoptarla, suele tener un retorno de la inversión de 4:1.

También mejora la **satisfacción de los empleados**, ya que evita situaciones de crisis que favorecen la sobreasignación del trabajador; **formaliza los procesos**, de manera que siempre sabemos qué hacer y se fijan las responsabilidades; se **asegura la calidad** buscando las causas de los fallos, y se **asegura la formación**, poniendo medios para garantizar que sabemos qué hay que hacer y cómo.

El modelo ideal

En conclusión, en el modelo ideal, empezaríamos diagnosticando el estado de los procesos, para pasar a establecer la estrategia, ejecutarla y seguir aprendiendo para seguir en mejora continua.



Tema 3 – Ciclos de vida

Tema 2: "**Ciclo de vida** – Sucesión de etapas por las que pasa el software desde que se inicia un proyecto hasta que se deja de usar. Cada una de estas etapas conlleva una serie de tareas a realizar y una serie de documentos (o software) que serán la salida de cada una de estas fases y servirán de entrada para la siguiente fase."

Ciclo de vida en cascada

Es el ciclo de vida **clásico y el más usado**, que incluye toda la vida del producto: desarrollo, pruebas, uso y mantenimiento. Se trata de un enfoque sistemático, disciplinado y secuencial, en el que cada fase empieza cuando termina la anterior y para pasar de una fase a la siguiente es preciso conseguir todos los objetivos de la anterior.

Fases

Estas fases son las siguientes: Ingeniería del sistema → Análisis → Diseño → Codificación → Prueba → Utilización. Durante la fase de utilización se realiza también el mantenimiento hasta que el producto se sustituye. Más detalladamente:

- **Ingeniería y análisis del sistema** – En esta fase se trata de definir la interrelación del software con los otros elementos del sistema más complejo en el que está englobado. Comprende los requisitos globales a nivel del sistema, así como un cierto análisis y diseño a nivel superior, sin mucho detalle.
- **Análisis de requisitos** – Se realiza un análisis detallado de los componentes del software: datos a manejar, funciones a desarrollar, interfaces... El ingeniero debe comprender qué datos se van a manejar, cuál va a ser la función que tiene que cumplir el software, cuáles son las interfaces requeridas y el rendimiento que se espera lograr. Estos requisitos del sistema y del software deben documentarse correctamente y revisarse con el cliente.
- **Diseño** – El diseño debe aplicarse a cuatro características del software: la **estructura de datos**, la **arquitectura de las aplicaciones**, la **estructura interna** de los programas y las **interfaces**. Con el diseño debe poder conocerse la arquitectura, funcionalidad y la calidad antes de codificar.
- **Codificación** – Consiste en la traducción del diseño a un lenguaje de programación compilable para ejecutarlo.
- **Pruebas** – En esta fase es crucial comprobar el rango de entradas en el programa más amplio, no sólo los casos normales, de manera que se prueben **todos los módulos** que forman parte del sistema.
- **Utilización** – Aquí comienza la vida útil del software. Esta fase se solapa con las posteriores hasta que el software deja de utilizarse.
- **Mantenimiento** – los cambios que se realizarán en esta etapa tendrán tres causas: **errores en el código**, **cambios en el entorno** del software o cambios (o **ampliaciones**) en el proyecto. El mantenimiento supone volver atrás, a las etapas de codificación, diseño o análisis en función de la magnitud del cambio.

- **Sustitución** – La vida del software es limitada y cualquier aplicación acaba siendo sustituida por otra más amplia, rápida o bonita y fácil de usar. Aún así, es una tarea que se debe llevar a cabo planificándose cuidadosamente y de forma organizada. Suele implicar el desarrollo de programas para la interconexión de ambos sistemas.

Aportaciones

- Es el más **simple**, conocido y fácil de usar.
- **Define una serie de procesos** a realizar que posteriormente se formalizarán en normas.
- Permite generar software **eficientemente** de acuerdo con las especificaciones.
- Ayuda a **prevenir que se sobrepasen las fechas** de entrega y los costes esperados.
- Al final de cada fase, el personal técnico y los usuarios **pueden revisar el progreso**.

Críticas

- Es **muy costoso volver a una fase anterior** del proyecto y los requisitos no siempre pueden establecerse desde el primer momento.
- En realidad no es secuencial, **hay iteraciones y exige refinamiento** (por la fase de mantenimiento)
- Hasta que el ciclo concluye, **no hay una versión funcional** del programa, que es la que en realidad ayuda a detectar más fallos.
- Existen muchos procesos con **dependencias FS** (Finish to Start) hacia otros, por lo que es fácil que se bloqueen.
- **Acentúa el fracaso** de la Ingeniería de Software hacia el usuario final.

Construcción de prototipos

Dado que mediante el ciclo de vida en cascada se achacaba el hecho de que no se disponía de una versión operativa hasta el final y que también podía darse el caso de que los requisitos no estuviesen bien fijados, se ideó la construcción por prototipos, basada en la entrega de versiones cada vez más mejoradas y adaptadas.

Elección

Pero no todos los proyectos pueden desarrollarse con este modelo, ya que requiere una gran interacción con el cliente para que pruebe el software y muestre su conformidad con las versiones. Debe elegirse este modelo cuando:

- Haya un gran nivel de **incertidumbre**
- El programa deba realizar **mucha interacción** con el usuario y tenga **algoritmos refinables**.
- El programa **no sea muy complejo**, de forma que no se tarde mucho en realizar cada uno de los prototipos.
- El **cliente esté dispuesto** a probar distintos prototipos.

Tipos de prototipo

- **Simulador de interfaz**, puede realizarse en papel o ejecutable en ordenador, para describir la IPO.
- **Versión no eficiente**, que tenga todas o parte de las funciones pero con características que deban ser mejoradas.
- Versión con **subconjuntos de funciones**, que sirvan para evaluar el rendimiento de algoritmos, necesidades de capacidad de almacenamiento, velocidad de cálculo...

Todos estos tipos de prototipo nos ayudarán a **especificar requisitos** (comprender el problema), **analizar alternativas** (explorando otras soluciones mediante la generación de prototipos) y analizar la viabilidad y **rendimiento de cada sección** de programa.

Fases

Las fases principales de la construcción por prototipos serían las siguientes:

(**Recolección y refinamiento** de requisitos → **Diseño** rápido → Construcción de un **prototipo** → **Evaluación**) → **Desarrollo final** del producto definitivo

Nota: los ( ) indican un bucle.

Aportaciones

- Permite **aprovechar trabajo** ya realizado.
- Permite **refinar requisitos**.
- Permite **refinar el diseño** inicial, así como el diseño de las pantallas e informes.

- Garantiza que los **algoritmos** y los **módulos** están **probados** antes de la codificación final.
- Permite **analizar alternativas**, así como la viabilidad de las soluciones.

Críticas

- Resulta **imposible predecir** fiablemente el **coste** del proyecto.
- El **resultado final** suele ser un **prototipo**, por lo que puede tener algunas de sus características:
  - Se asumen **elecciones apresuradas** de arquitectura, plataforma de desarrollo y alternativas, por lo que también resulta difícil de mantener.
  - Contiene gran cantidad de **errores latentes**, por lo que es poco fiable.
  - El **desarrollo** es **poco eficiente**.

Desarrollo en incrementos

Suele presentarse como una **fusión entre** los modelos de **cascada** y **prototipos**. Basa el desarrollo del proyecto en un análisis inicial y un conjunto de incrementos que van aumentando la funcionalidad del software y refinando la anterior, de manera que el producto final es la integración del resultado de sucesivos refinamientos. De todas formas, a diferencia del modelo por prototipos, el modelo incremental se centra en obtener un **producto operativo aunque incompleto en cada iteración**.

Fases

El modelo incremental tendría las siguientes fases:

Análisis de requisitos del sistema → Análisis de requisitos Software → (Diseño preliminar → Diseño detallado → Codificación y pruebas → Explotación y mantenimiento)

Técnicas de 4ª generación

Consiste en el uso de herramientas que tienen por objetivo facilitar al desarrollador a especificar algunas características del software a alto nivel. Los generadores de código más habituales cubren uno o varios de los siguientes aspectos:

- **Acceso a bases de datos**: mediante lenguajes de consulta de alto nivel (derivados del SQL) permiten al desarrollador abstraerse de la estructura de los ficheros o tablas y de sus índices.
- **Generación de código**: a partir de una especificación de los requisitos pueden generar automáticamente toda la aplicación.
- **Interacción y definición de pantallas**: permiten diseñar la interfaz dibujándola directamente además de facilitar el control del cursor y la gestión de errores en los datos de entrada.
- **Generación de informes**: de forma similar a la de pantallas.
- **Manipulación de datos**.

- **Capacidades gráficas** de alto nivel.
- Capacidad de **hojas de cálculo**.

Fases

Las fases generales de este ciclo de vida son:

Recolección de requisitos → Estrategia de diseño → Generación de código → Prueba → Utilización. También, al igual que en los modelos anteriores, la utilización se solapa con el mantenimiento y la sustitución.

En proyectos pequeños suele saltarse la estrategia de diseño, ya que no cobra una gran importancia, dado que se trata de una descomposición modular de requisitos y la definición de la arquitectura.

En las pruebas sobre todo se mide la eficiencia del código generado y se validan los requisitos formulados por el cliente.

Aportaciones

- Permiten **reducir** bastante el **tiempo** de codificación.
- Ayudan a los desarrolladores a **fijar el diseño** del proyecto.

Críticas

- No son más fáciles de usar que los lenguajes de tercera generación, ya que **no consiguen prescindir de la codificación**, si no disfrazarla de especificación.
- El **código** producido es **ineficiente**, ya que el software hecho a mano siempre es más adaptado al problema. De todas formas, la reducción en el tiempo de desarrollo y el aumento de la velocidad de procesamiento hace que cada vez esta menor eficiencia quede compensada.
- Sólo son **aplicables a software de gestión**, aunque cada vez están surgiendo más campos de aplicación.

Modelo en espiral

Combina las principales ventajas del ciclo en cascada con las del modelo de prototipos. Es un modelo nuevo propuesto a finales de los 80 y que no ha sido tan usado como los anteriores pero se espera que se use cada vez más. Aún así, es más realista que el modelo clásico.

Su principal característica es el **análisis de riesgos**. Los prototipos se utilizan como mecanismo de reducción del riesgo, permitiendo valorar la opción de no embarcarse en un proyecto demasiado arriesgado que podría acabar en fracaso.

Análisis de riesgos

Antes de explicar el proceso, es conveniente fijar los conceptos acerca de este proceso:

- **Activos**: elementos del sistema de información que soportan la misión de la Organización.

- **Amenazas**: son causa potencial de incidentes que pueden causar daños a un sistema de información (activos) o a una organización. (definición de UNE 71504:2008).
- **Salvaguardas** (o contra medidas): medidas de protección desplegadas para suprimir o reducir el daño de las amenazas.
- **Riesgo**: estimación del grado de exposición a que una amenaza se materialice.
- **Análisis de riesgos**: proceso sistemático para estimar la magnitud de los riesgos a los que está expuesta la Organización.
- Proceso de gestión/**administración de riesgos**: proceso destinado a modificar el riesgo.

"Los activos interesan por su valor y están expuestos a amenazas. Estas amenazas pueden causar una cierta degradación, que tendrá un impacto mayor o menor en la Organización. El riesgo es la medición de ese impacto, en función de la probabilidad de que la amenaza tenga lugar y el valor del activo."

Tipos de activos

- **Soportes de información**: dispositivos para el almacenamiento de datos (servidor).
- **Equipamiento auxiliar**: complementa el material informático (smartphone para pruebas).
- **Redes de comunicaciones**: permiten intercambiar datos (internet).
- **Instalaciones**: acogen equipos informáticos y de comunicaciones (local de la empresa).
- **Personas**: que explotan u operan los demás activos.

Dimensiones de los activos

Los activos se dividen en dos grandes grupos: información y servicios:

Dimensiones de la **información**:

- **Confidencialidad**: daño que causaría si la conociese alguien que no debería.
- **Integridad**: daño en caso de que la información esté dañada, corrupta, manipulada...
- **Disponibilidad**: daño que causaría el que no estuviese disponible.

Dimensiones de los **servicios**:

- **Autenticidad**: daño que causaría no saber quién hace o ha hecho cada cosa.
- Trazabilidad del **uso del servicio**: daño que causaría no saber a quién se le presta el servicio.
- Trazabilidad del **acceso a los datos**: daño que causaría no saber quién accede a los datos.

Tipos de amenazas

- De origen natural.
- Del entorno (de origen industrial).
- Defectos en las aplicaciones.
- Causadas por las personas de forma accidental.
- Causadas por las personas de forma deliberada.

Administración del riesgo

Para una correcta administración del riesgo debe empezarse por la **identificación del riesgo**. Para esto es necesario:

- Determinar los **activos relevantes** para la organización.
- Determinar a qué **amenazas están expuestos** dichos activos.
- **Definir el riesgo** en función de la probabilidad de que un activo esté expuesto y del valor de esa amenaza.
- **Clasificar los riesgos** en taxonomías: los del proyecto afectan a la calendarización o recursos; los del producto afectan a la calidad o desempeño de software, y los del negocio afectan a la organización responsable del desarrollo.

Tras identificar los riesgos, es necesario realizar el **análisis de riesgos**, es decir, evaluar para cada uno la **probabilidad** de que ocurra (alta, moderada o baja) y sus **consecuencias de impacto** o degradación (catastrófico, serio, tolerable...).

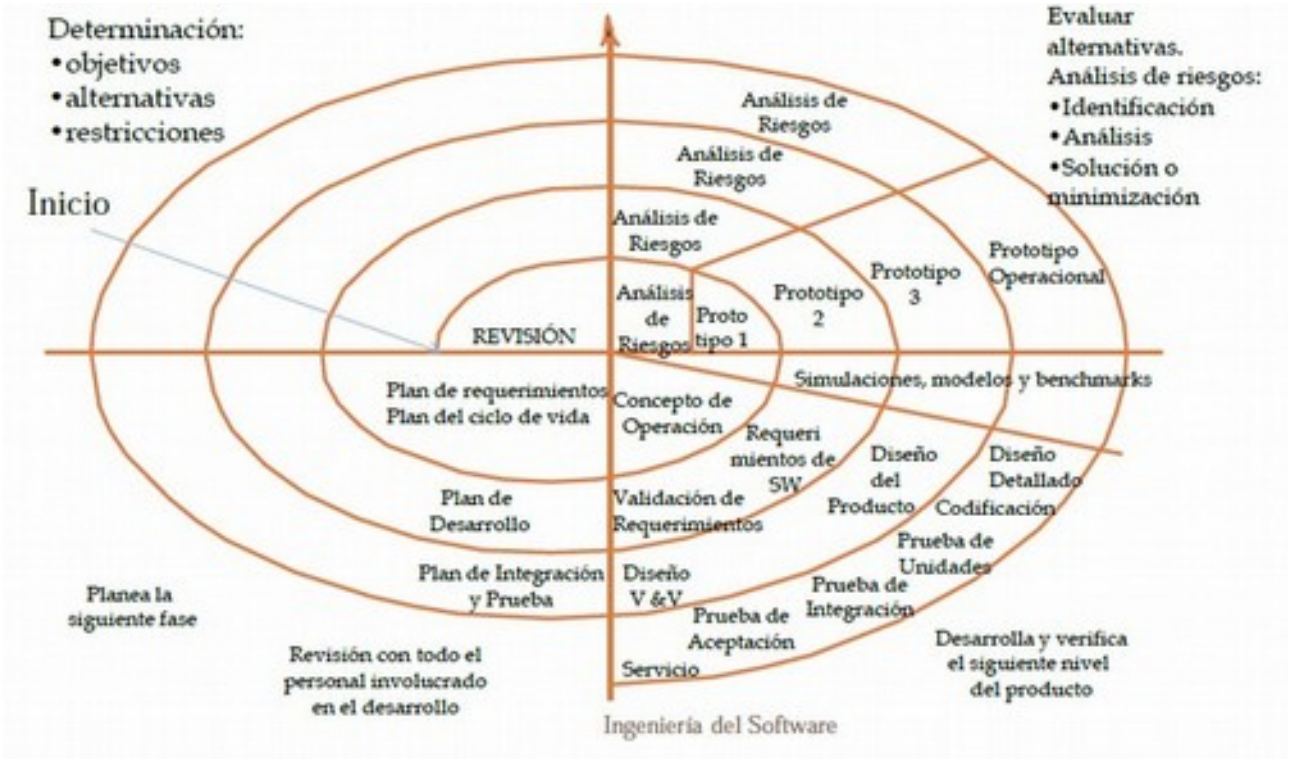
Una vez que se hace este análisis, se procede con la **planificación de riesgos**, esto es, realizar estrategias para la administración de riesgos: estrategias de **prevención**, de **minimización**, planes de **contingencia** o **transferencia** en función del tipo de riesgo.

Una vez que se completa la finalización, sólo queda la **supervisión de riesgos**, supervisar el proyecto en base a **indicadores asociados a cada riesgo** para poder detectarlos a tiempo, minimizando los daños con las tareas previstas en caso de aparición.

Fases

El modelo en espiral se divide en actividades estructurales o regiones neutras:

- **Definición de objetos** – objetos específicos, alternativas y restricciones.
- **Análisis de riesgo** – identificación, análisis detallado y búsqueda de estrategias que los minimicen.
- **Desarrollo y validación** – generación de los entregables de los procesos clásicos.
- **Revisión y planificación** – valoración de resultados y planificación del siguiente ciclo.



Críticas

- **No se adapta** bien a un **contrato**. Un planteamiento posible sería un contrato por iteración.
- **Requiere** habilidad de **gestión de riesgos**, ya que un riesgo no detectado equivale a un requisito mal definido en los modelos secuenciales.
- Puede ser **difícil de controlar** y de convencer al cliente de que es controlable.

[Recordar el ejemplo de clase del programa para ayudar a decidir a los bomberos.]

Desarrollo ágil

Hasta ahora se han visto modelos pesados con una aproximación sistemática y disciplinada. Son procesos fuertemente orientados a la documentación, la cual no es un objetivo, sino el medio para alcanzarlo garantizando su calidad. Todo ese formalismo da calidad al producto y al proceso, pero ralentiza demasiado el desarrollo.

En el manifiesto para el desarrollo ágil se señala la necesidad de valorar un conjunto de activos por encima de otros:

- Los **individuos y sus intenciones** por encima de los procesos y las herramientas que usen.
- El **software en funcionamiento** por encima de la documentación extensa.
- La **colaboración con el cliente** por encima de la negociación del contrato.
- La **respuesta al cambio** por encima del seguimiento de un plan.

En el manifiesto se acepta la importancia de todas las cosas citadas anteriormente, pero se valora más a las que están por encima.

Aportaciones

El desarrollo ágil trata de aliviar el peso de las metodologías pesadas reduciendo la presión sobre la calidad de los modelos, los cuales solamente deben ser suficientemente buenos. Con esto trata de superar todas las limitaciones de las metodologías pesadas, permitiendo **adaptarse al cambio** con facilidad y **superar la fragilidad** asociada a las debilidades humanas.

La primera manera de superar esas limitaciones es la **gestión del cambio**: mientras que las metodologías pesadas intentaban controlar el cambio limitando los cambios permitidos y los momentos de cambio, las ligeras intentan **acelerar las entregas** del software y **hacer participar** activamente al **cliente** pidiéndole que evalúe el producto entregado y permitiendo que establezca nuevos requisitos y nuevas prioridades.

Para superar la fragilidad asociada a las debilidades humanas, las metodologías ligeras permiten que la aplicación de los **modelos se adapte al equipo**, admitiendo además la falta de eficiencia en el planteamiento, mientras que las pesadas obligan a la construcción de modelos fijos y el desarrollo sistemático.

Fuerte dependencia del Personal

Que el desarrollo ágil funcione depende en un grado muy alto de los rasgos del personal:

- **Competencia**: el equipo debe tener un talento innato, habilidades específicas relacionadas con el software y un conocimiento general del proceso que el equipo haya elegido.
- **Enfoque común**: todos deben tener claro el objetivo básico, la entrega rápida del producto.
- **Colaboración estrecha con todos los stakeholders** relacionados con el proyecto: los otros miembros del equipo, el cliente y sus gerentes.
- Habilidades para la **toma de decisiones** en cuanto a cuestiones técnicas y del proyecto, que son claves para la entrega rápida.
- Capacidad de **resolución de los problemas** confusos derivados de la no concreción de requisitos.
- Confianza y respeto mutuo, propios de un **equipo cuajado** (equipo de fútbol con experiencia)
- **Organización propia**: esto incluye tres factores.
    - El equipo ágil se organiza a sí mismo para el trabajo que debe hacerse.
    - El equipo organiza el proceso que mejor se ajusta a su ambiente local.
    - El equipo organiza el programa de trabajo para que se alcance de mejor manera la entrega del incremento de software.

La alianza ágil

Es la definición de **12 principios** con los que se quiere alcanzar la agilidad. Estos son los siguientes:

1. Nuestra mayor prioridad es **satisfacer al cliente** mediante la entrega temprana y continua de software valioso.
2. Bienvenidos los **requisitos cambiantes**, incluso en fases tardías del desarrollo. La estructura de los procesos ágiles cambia para la ventaja competitiva del cliente.
3. **Entregar** con frecuencia **software en funcionamiento**, desde un par de semanas hasta un par de meses, con una preferencia por la escala de tiempo más corta.
4. La **gente de negocios** y los **desarrolladores** deben **trabajar juntos** a diario a lo largo del proyecto.
5. Construir proyectos alrededor de **individuos motivados**. Darles el ambiente y el soporte que necesitan, y confiar en ellos para obtener el trabajo realizado.
6. El método más eficiente y efectivo de transmitir la información hacia y dentro de un equipo de desarrollo es la **conversación cara a cara**.
7. El **software en funcionamiento es la medida primaria** de progreso.
8. Los procesos ágiles promueven el **desarrollo sostenible**. Los patrocinadores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de manera indefinida.
9. La atención continua a la **excelencia técnica** y al **buen diseño** mejora la agilidad.
10. La **simplicidad** -el arte de maximizar la cantidad de trabajo no realizado- es esencial.
11. Las mejores arquitecturas, los mejores requisitos y los mejores diseños emergen de **equipos autoorganizados**.
12. A **intervalos regulares** el equipo refleja la forma en que se puede **volver más efectivo**; entonces su comportamiento se ajusta y adecua en concordancia.

Hay muchos modelos que siguen los principios de la alianza ágil. Estos son:

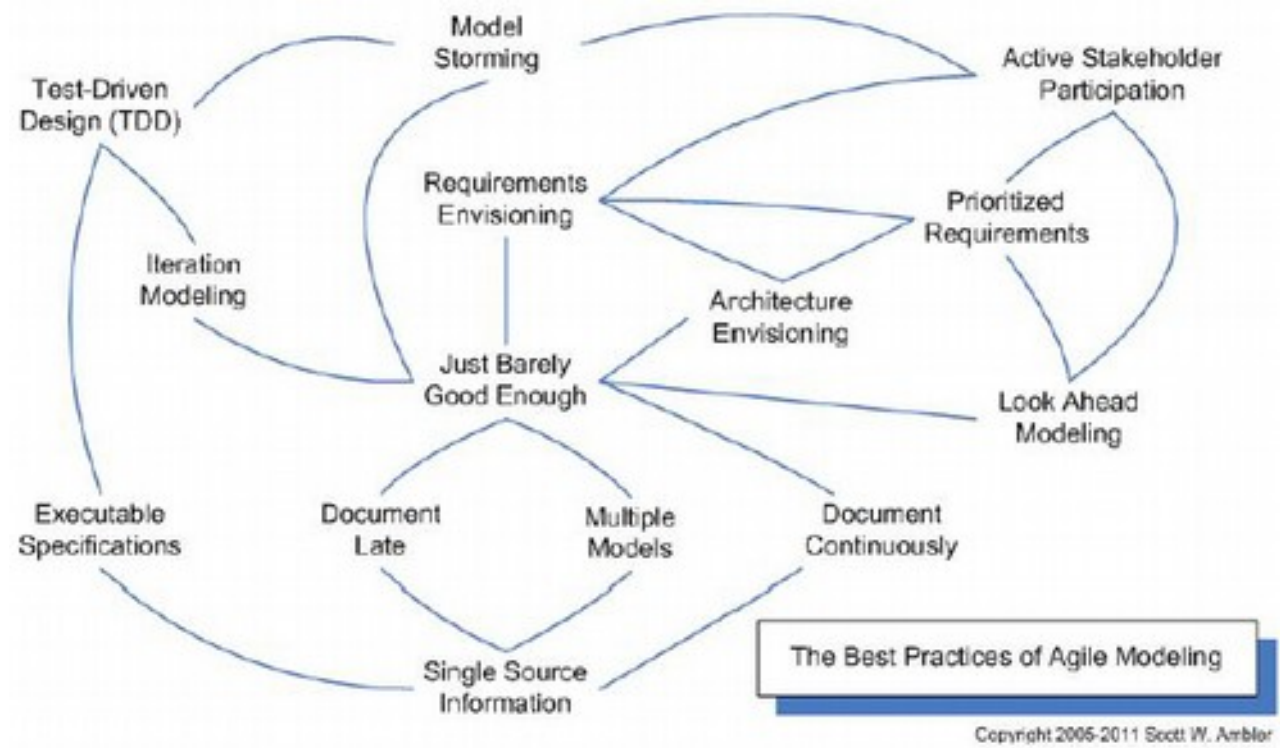
- Desarrollo Adaptativo de Software (DAS)
- Método de desarrollo de sistemas dinámicos (MDSD)
- Melé (Scrum)
- Cristal
- Desarrollo conducido por características (DCC)
- Modelado ágil (MA)
- Programación Extrema (PE)

Modelado ágil (AM)

En inglés Agile Modeling (AM), es una metodología basada en la práctica para alcanzar una documentación y un modelado efectivos para programas software.

A un nivel más detallado, MA es una **colección de valores, principios y prácticas**.

En un nivel más alto, MA es una colección de las mejores prácticas, representadas en el siguiente esquema.

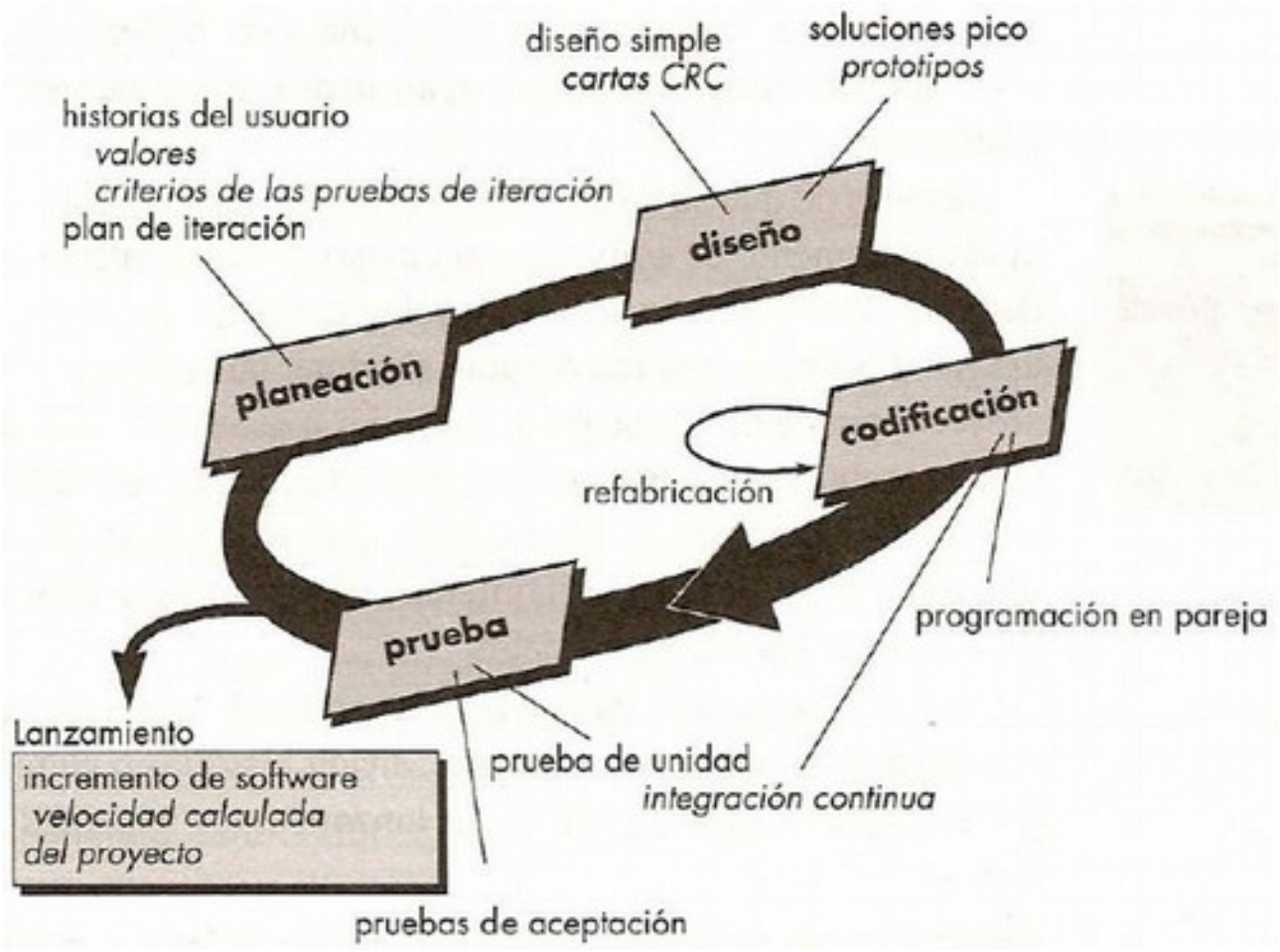


Además de los principios del desarrollo ágil, consistentes con MA, existen una serie de principios suplementarios de los que caben destacar los siguientes:

- **Conocer los modelos y las herramientas**: es necesario entender las fortalezas y debilidades de cada modelo y las herramientas con los que se creó.
- Usar **múltiples modelos**: cada modelo debe presentar un aspecto diferente del sistema y sólo aquellos que proporcionen un valor para quienes están destinados deben usarse.
- **Modelar con un propósito**: antes de realizar un modelo, debe tenerse claro para qué se utilizará, lo cual hará que el tipo de notación a usar y el grado de detalle requerido sean obvios.
- El **contenido es más importante** que la representación: es más importante que el modelo comunique un contenido valioso aunque la notación no sea perfecta.
- **Viajar ligero**: sólo se deben conservar y mantener los modelos que de verdad hagan falta y permitan al equipo comunicarse entre ellos y con los propietarios del proyecto.
- **Adaptar al equipo** ágil: el enfoque del modelado debe adaptarse a las necesidades del equipo ágil.

Programación extrema (PE)

La PE utiliza un **enfoque orientado a objetos** como su paradigma de desarrollo preferido. Abarca un conjunto de reglas y prácticas que ocurren en el contexto de las cuatro actividades del marco de trabajo que se detallan a continuación:



Planificación

Se empiezan creando unas **Historias de usuario** (casos de uso) y se colocan en una carta índice para que el usuario les asigne una prioridad en función del valor característico o la función para su negocio). Estas historias pueden depender de la presencia de otras y pueden crearse, dividir, eliminar o cambiar de valor en cualquier momento.

A cada historia se le **asigna un coste en semanas** y en caso de que alguna tenga un coste mayor de 3 semanas, debe sobrescribirse.

Después se inicia el **plan de construcción**, en el que los clientes y el equipo agrupan y seleccionan juntos las historias del siguiente incremento y acuerdan las fechas de entrega del incremento. Así, las historias seleccionadas se implementan de inmediato, seleccionándose en las primeras entregas las que tengan un valor más alto.

En cada lanzamiento se **estima la velocidad del proyecto** y se usa para estimar las fechas de los siguientes incrementos y determinar si se ha hecho un compromiso excesivo, permitiendo cambiar el contenido de los lanzamientos o las fechas de entrega final.

Diseño

El primer principio del diseño de PE es mantenerlo simple, en inglés **KISS** (Keep It Simple, Stupid). Siempre se prefiere un diseño simple respecto a una representación más compleja, ya que el diseño solamente debe ofrecer una **guía de implementación** para una historia. Se desaprueba el diseño de funcionalidades extra, ya que si se quieren, se requerirán más tarde.

PE apoya el uso de **tarjetas CRC** para facilitar la orientación a objetos. Éstas identifican y organizan las clases orientadas al objeto que son relevantes para el incremento del software.

Cuando el diseño es complejo, se recomienda **programar un prototipo** de esa porción del diseño, llamado **solución de pico**, que se evalúa. El propósito es reducir el riesgo cuando comience la verdadera implementación.

PE también apoya la **refabricación**, que es el proceso de cambiar un sistema software de manera que no altere el comportamiento externo pero que mejore la estructura interna.

Como únicos productos resultantes del diseño quedan las tarjetas CRC y las soluciones de pico.

Codificación

Antes de codificar deben prepararse las **pruebas de unidad** que debe superar el código, de manera que se pueda programar para superar esas pruebas.

Es común en PE la **programación en parejas**, de manera que se trabaje sobre la misma estación de trabajo pero jugando papeles distintos.

Se busca siempre una **integración continua**, de manera que generando código se integren todas las partes. Este código de integración lo puede realizar un equipo de integración o el propio equipo de desarrollo. La integración continua permite el uso de la **prueba de humo**, un banco de pruebas que sin ser exhaustivo, prueba todo el sistema permitiendo encontrar errores importantes.

Prueba

Las **pruebas** se realizan **diariamente**, organizándose pruebas individuales en un conjunto universal de pruebas y pudiendo realizarse automáticamente mediante herramientas especializadas (jUnit).

Estas pruebas permiten **indicar el progreso** del proyecto y avisar cuando las cosas salen mal. Como consecuencia directa, permite arreglar problemas pequeños cada poco tiempo, lo cual es muy beneficioso, ya que es más fácil que arreglar un problema grande antes de la fecha límite.

En cada entrega el cliente especifica unas **pruebas de aceptación** que se derivan de las historias y están orientadas a características generales y funcionalidad del sistema, así como a elementos visibles y revisables por el cliente.