2 La Ingeniería del Software

El objetivo de este apartado es dar una visión global del contexto de la Ingeniería del Software. Las cuestiones abordadas en el mismo, en general, no presentan un nivel de detalle excesivo, más allá del cometido de ubicar al lector en el ámbito de esta disciplina.

Para ello, primeramente se dará respuesta a una serie de preguntas, definiendo con ello **conceptos fundamentales** como qué se conoce como Software e Ingeniería del Software y algunas características de esta disciplina que ayudarán a conocer la **problemática** que la atañe. Fundamentalmente las publicaciones de I. Sommerville [2] y R.S. Pressman [3] se han utilizado para tratar estas cuestiones.

A continuación, se expondrán los **modelos de proceso** y **metodologías** más extendidas y de mayor relevancia para este estudio.

Por último, se dará uno de los estándares de facto en cuanto a la evaluación de los procesos en el ámbito de la Ingeniería del Software, el **modelo de madurez de la capacidad del software** del Instituto de Ingeniería del Software de EEUU (*SEI*)

2.1 Conceptos fundamentales

¿Qué se entiende por Software?

> *"Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora."*
>
> - RAE [4]

> *"Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación."*
>
> - IEEE 729 [5]

Como vemos en esta definición mucho más exacta, se incluye la documentación y los datos como parte de lo que se conoce como Software.

¿Qué se entiende por Ingeniería del Software?

> *"Ingeniería del Software es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como desarrollo de software o producción de software"*
>
> - B. Bohem

En esta definición del proceso de desarrollo Software, se introduce como parte inherente del producto a obtener, la perspectiva de las necesidades de usuario a las que debe dar respuesta:

> *"Aquellos en los que las necesidades del usuario se traducen en requerimientos, estos se transforman en diseño y este a su vez se implementa en código que es probado, documentado y certificado para su uso"*
>
> - G. Booch, I. Jacobson, y J. Rumbaugh [6]

En la siguiente, se introducen los conceptos de rentabilidad y fiabilidad y la operatividad en entornos reales:

> *"La Ingeniería del Software trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable que sea fiable y trabaje en máquinas reales."*
>
> - F. L. Bauer [7]

Y según otra de las definiciones que aporta la IEEE, se introduce el concepto de cuantificación del proceso:

> *1) La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; es decir, la aplicación de ingeniería al software.*
> *2) El estudio de enfoques como en (1).*
>
> - IEEE 610.12 [5]

Como compendio de los conceptos barajados en las definiciones expuestas se propone la siguiente definición de I. del Software:

> *"La aplicación práctica, sistemática, disciplinada y cuantificable del conocimiento científico para realizar el análisis de las necesidades del usuario y obtener el software y la documentación asociada requerida para su desarrollo, operación y mantenimiento de manera rentable, fiable, certificada y que opere en máquinas reales."*

2.2 Problemática de la Ingeniería del Software

¿Qué complejidad inherente tiene el proceso de desarrollo de Software?

El proceso de desarrollo es intensamente intelectual y se ve afectado por la creatividad y juicio de las personas involucradas en el mismo.

Aunque un proyecto de desarrollo de software es equiparable en muchos aspectos a cualquier otro proyecto de ingeniería, en el desarrollo de software hay una serie de desafíos adicionales relativos a la naturaleza del producto a obtener, como son la intangibilidad o la fiabilidad del mismo:

Intangibilidad
Hablamos de un producto intangible y habitualmente complejo, dado que su cometido es dar respuesta a la abstracción de un problema planteado por personas que normalmente desconocen esta disciplina.

Por esta cuestión, definir con exactitud los requisitos a cubrir y consolidarlos tempranamente se complica con frecuencia, haciendo inevitable el cambio, durante el desarrollo o una vez acabado el mismo.

Fiabilidad
Un producto software en sí es complejo siendo inviable conseguir el 100% de fiabilidad en un programa por muy reducida que sea su funcionalidad. Esto ocurre por la elevada combinatoria asociada a los distintos factores que intervienen en la ejecución del mismo y que impiden una verificación de las todas posibles situaciones que se puedan presentar, son ejemplo de estos factores:

- Datos introducidos por el usuario
- Datos almacenados en el sistema
- Interacción con el software base o sistema operativo
- Interacción o el hardware del sistema sobre el que se ejecuta
- Interacción con otras aplicaciones
- Etc.

Por otra parte, el ámbito de esta disciplina es tan amplio que se abordan problemas tan dispares como pueden ser el diseño de una simple Web de promoción de una empresa, una aplicación que de el soporte al trabajo diario de gestión de una organización, o el desarrollo del software de control de un misil en tiempo real. Por este motivo, las prácticas que aplican en un determinado ámbito no tienen por qué ser de aplicación en el resto, sí bien hay una serie de principios comunes que si pueden ser adaptados a prácticas concretas para cada situación.

Este concepto es tratado por Pressman y que lo define como "Marco común del Proceso" y que lo ilustra con la siguiente figura [3]:

Figura 1 - Marco común del proceso

Se definen un conjunto de **actividades** aplicables a cualquier proyecto, una serie de conjuntos de tareas que permitirán la adaptación de las actividades a cada proyecto en particular y requisitos del equipo de proyecto y un conjunto de **actividades de protección**, independientes de cualquier actividad del marco de trabajo, presentes durante todo el proceso y destinadas a cuestiones como la garantía de calidad, gestión de la configuración, gestión de riesgos, etc.

De una forma más concreta, las actividades del marco de trabajo se pueden asimilar a tres fases genéricas:

- **Definición**: Destinada identificar qué se pretende obtener. Establecer cuestiones como qué información debe procesar el sistema, qué funcionalidad implementar, cómo comportarse, etc. Durante esta fase tendrán lugar tareas como la ingeniería de sistemas o de información, planificación de proyecto y análisis de requisitos.

- **Desarrollo**: en la que efectivamente se da solución a las necesidades definidas en la fase anterior, y su cometido principal es la construcción del producto final. En esta fase tienen lugar tareas como el diseño, la codificación y pruebas del software.

- **Mantenimiento**: que se ocupa del cambio del producto obtenido en la fase de desarrollo. Este cambio puede venir motivado por distintas cuestiones, distinguiéndose fundamentalmente cuatro tipo de cambios:

    - **Correcciones** de defectos localizados tras el desarrollo, cuestión inevitable, como explicamos con anterioridad. Las modificaciones realizadas para dar solución a estos cambios se conocen como mantenimiento correctivo.

    - **Adaptaciones** del software ante el cambio del entorno original para el que fue desarrollado, cambios del hardware, del software base, etc. Realizadas por el mantenimiento adaptativo.

    - **Mejoras** que den respuesta a nuevos requisitos del cliente, cubiertas por el mantenimiento perfectivo

    - **Prevención** de la degradación que se produce por el cambio. El mantenimiento preventivo o reingeniería del software se ocupa de realizar cambios que faciliten la corrección, adaptación y mejora.

Por último, es destacable la visión genérica y multicapa que ofrece Pressman como fundamento de la Ingeniería del Software, donde herramientas, métodos y procesos se apoyan sobre un **enfoque de calidad** [3].

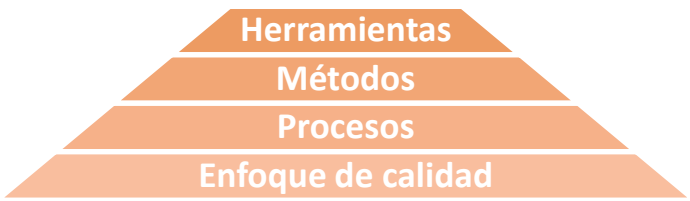

Figura 2 - Visión multicapa de la Ingeniería del Software

- **Calidad**: indispensable para cualquier disciplina de ingeniería. Idea reforzada por la tendencia actual de fomento de la cultura de mejora continua de procesos.

- **Procesos**: definen un marco de trabajo para el conjunto de áreas clave base del control de la gestión de proyectos y establecen el contexto de aplicación de los métodos técnicos y otras cuestiones como el aseguramiento de la calidad y gestión del cambio.

- **Métodos**: especifican cómo construir técnicamente el Software, abarcando desde las tareas de especificación de requisitos y el diseño hasta la construcción, pruebas y mantenimiento.

- **Herramientas**: son aquellas que darán un soporte más o menos automático a procesos y métodos.

2.3 Modelos de proceso

También conocidos como paradigmas, son abstracciones que si bien no describen detalladamente el proceso a seguir para el desarrollo software, representan diferentes estrategias o enfoques para abordar este problema:

> *"Una descripción simplificada de un proceso del software que presenta una visión de ese proceso. Estos modelos pueden incluir actividades que son parte de los procesos y productos de software y las personas involucradas en la ingeniería del software."*
>
> - Sommerville [2]

A continuación se describen los principales modelos de proceso:

- Lineal secuencial
- Basado en prototipos
- Evolutivo
  - Incremental
  - En espiral
- Basado en componentes

2.3.1 Lineal secuencial

También llamado ciclo de vida básico, utiliza un enfoque sistemático y secuencial de las actividades fundamentales que se mencionaron con anterioridad:

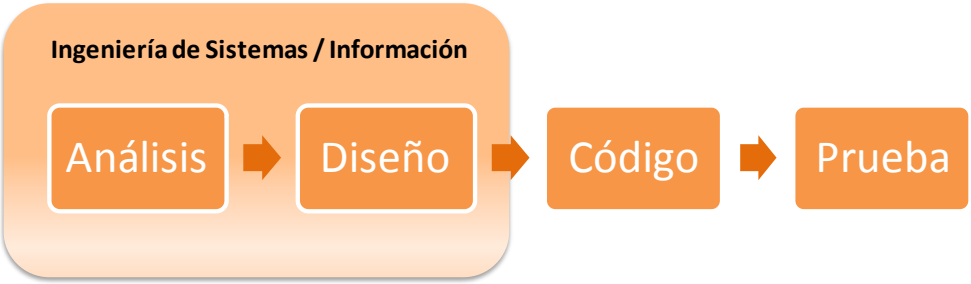

Figura 3 - Modelo lineal secuencial

En esta figura, propuesta por Pressman, se contempla el concepto de ingeniería de Sistemas / Información, que ofrece una visión que pone de manifiesto que el software suele formar parte de una solución mayor, y que su cometido es dar respuesta a un subconjunto de los requisitos a cubrir.

El modelo secuencial más conocido es el modelo en cascada, propuesto por Royce en 1970 [8].

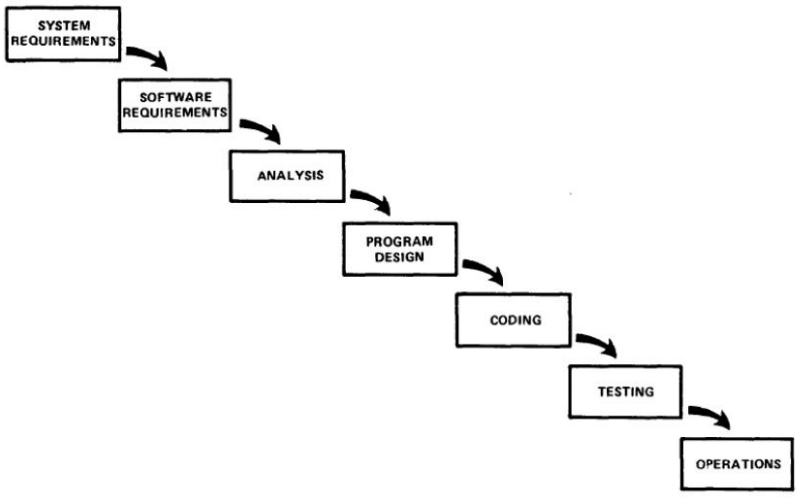

Figura 4 - Modelo en cascada

El uso de este modelo es frecuente en proyectos de tamaño considerable en el que se conocen bien los requisitos a los que se debe dar respuesta.

Son desventajas de este modelo:

- La necesidad de conocer los requisitos con exactitud desde el comienzo del proyecto, cuestión que no es habitual.
- El producto a obtener no se hace visible hasta el final, algo que provoca que los errores cometidos en la fase de análisis se propaguen por el resto de fases.

2.3.2 Basado en prototipos

Este modelo facilita la definición de los requisitos, mediante un diseño rápido centrado en la representación de los aspectos del software que serán visibles por el usuario mediante la construcción de un prototipo.

Una vez realizada la primera versión del prototipo, se irá refinando y validando hasta lograr la definición completa del sistema.

Figura 5 - Modelo basado en prototipos

La principal desventaja de este modelo es la percepción de "producto terminado" que puede dar el prototipo. La clave para evitarla es dejar claro desde el primer momento cuál es la función del prototipo y sus limitaciones.

2.3.3 Evolutivo

Surge para dar respuesta a una realidad: la necesidad de dar respuesta a la necesidad de rápida evolución del Software. La experiencia confirma que, con frecuencia, los requisitos cambian durante el desarrollo, cuestión que se ve reforzada por la presión sobre tiempos de entrega que dificultan la finalización de un producto completo y obligan a introducir versiones parciales, cada vez más completas, que den cierto grado de solución en un plazo menor.

2.3.3.1 Incremental

Combina elementos del modelo lineal aplicados repetidas veces con la filosofía de construcción de prototipos.

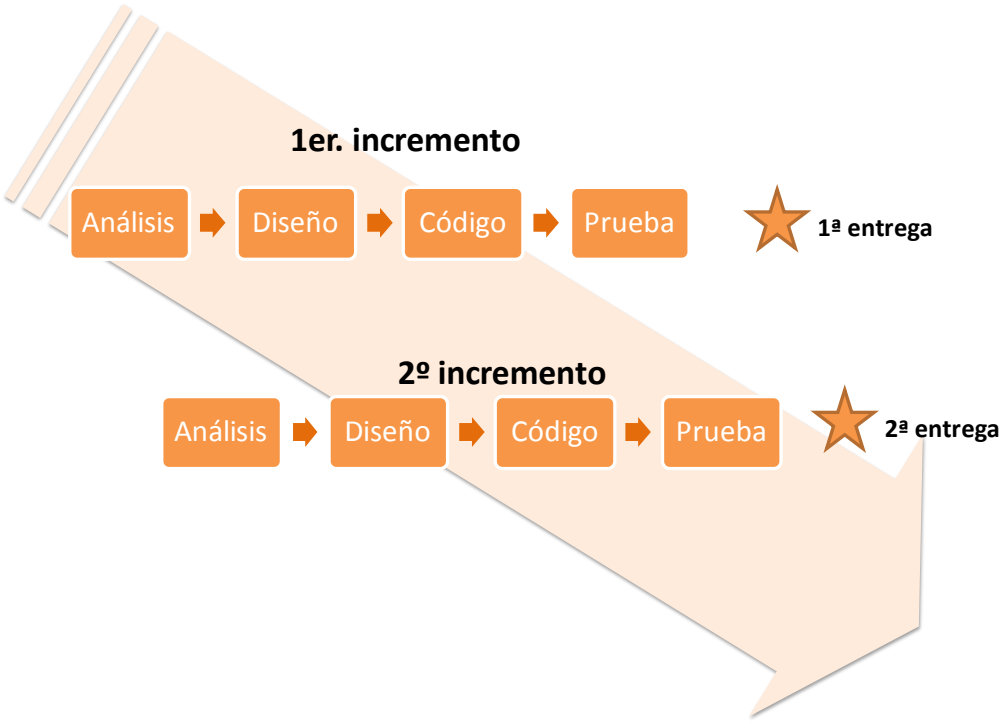

Figura 6 - Modelo incremental

A diferencia de la construcción de prototipos, el modelo incremental se centra en la entrega de un producto operativo en cada incremento, siendo los primeros incrementos versiones, si bien incompletas, que proporcionan la funcionalidad precisada además de una plataforma para la evaluación.

Este modelo también es útil cuando no se dispone de todos los recursos para realizar una implementación completa en una fecha establecida, limitándose la entrega a un primer incremento ajustado a la capacidad disponible.

2.3.3.2 En espiral

Combina igualmente la naturaleza iterativa del la construcción de prototipos con los aspectos sistemáticos del modelo lineal. Este modelo es propuesto por Boehm en 1988 [9] e incorpora los conceptos de análisis de riesgos y planificación como parte del proceso.

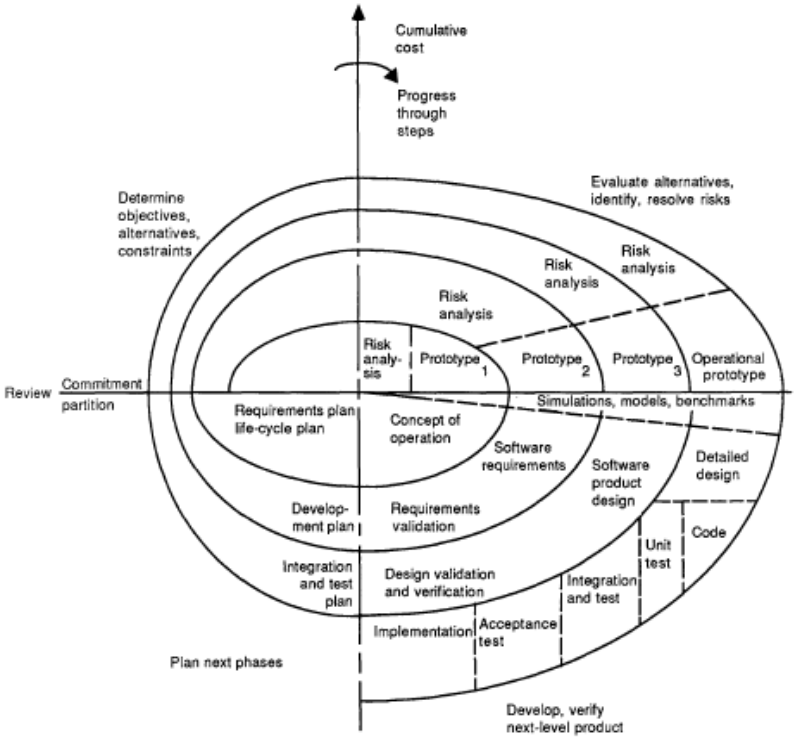

Figura 7 - Modelo en espiral

La espiral representa el avance del proyecto y ésta pasa por un conjunto de regiones que se corresponden con actividades del marco de trabajo.

El número de regiones propuestas por Boehm originalmente es de 4, aunque en la bibliografía revisada se distinguen hasta 6.

Una variante de este modelo, el modelo espiral Win-Win, también propuesto por Boehm en 1998 [10], interesante por introducir el concepto ganar-ganar como parte del proceso iterativo. El fundamento de esta variante se basa a que, con frecuencia, es difícil obtener del cliente los detalles necesarios para continuar el proceso, lo que se deriva en un proceso de negociación: la funcionalidad, rendimiento, y otros productos o características del sistema se sopesan frente al coste y al tiempo.

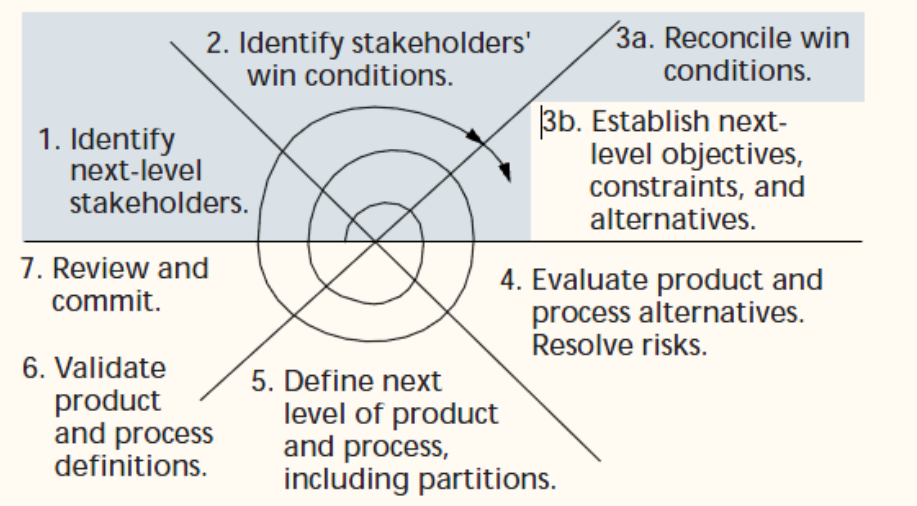

Figura 8 - Modelo en espiral WIN-WIN

Esta visión introduce un conjunto de actividades de negociación al principio de cada paso alrededor de la espiral.

2.3.4 Basado en componentes

Modelo evolutivo por naturaleza y que presenta características comunes con el modelo en espiral. También conocido como "Desarrollo basado en la reutilización", se apoya precisamente en el ese concepto: enfatizar la creación de clases y componentes que encapsulan datos y procedimientos para tratarlos de modo que se facilite su reutilización en distintos proyectos.

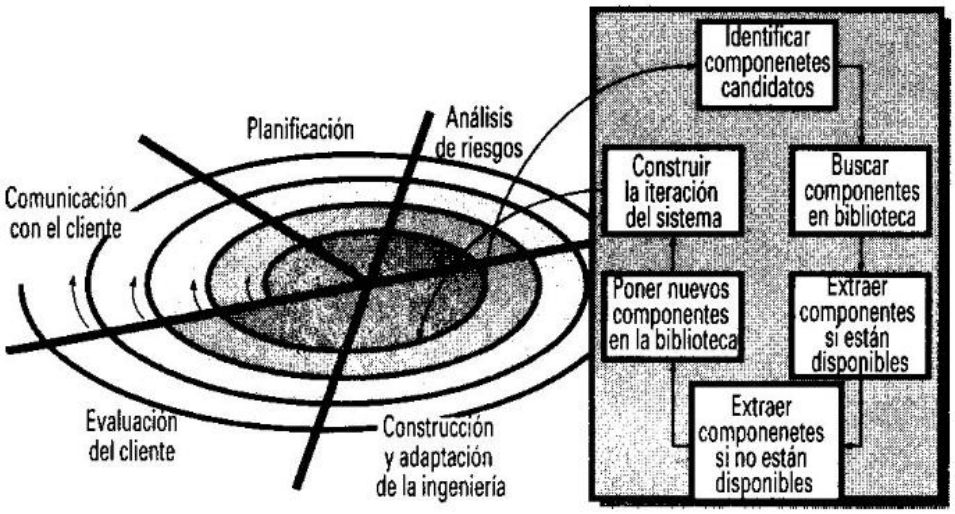

Figura 9 - Modelo basado en componentes

Este modelo introduce el concepto de "biblioteca de componentes" es un repositorio de elementos potencialmente reutilizables (clases y componentes).

A comienzo de la etapa de construcción, se distinguen las necesidades únicas del proyecto de las comunes entre distintos proyectos. Para darle respuesta a estas últimas se identifican los "componentes candidatos" de formar parte de la biblioteca.

Si existen los componentes candidatos existen en la biblioteca, se reutilizarán, y en otro caso, se construirán de tal modo que puedan ser incorporados posteriormente a la biblioteca.

De esta forma, se compone la primera iteración de la aplicación a construirse, mediante los componentes de la biblioteca y aquellos componentes únicos del proyecto (a priori, no reutilizables)

2.4 Metodologías

Las metodologías deben dar forma y detalle al proceso de desarrollo software y, adicionalmente, proporcionar un conjunto de prácticas y técnicas recomendadas, así como guías de adaptación a los distintos proyectos.

Habitualmente suelen ser combinación de modelos de proceso genéricos.

Para este estudio del arte, utilizaremos la siguiente distinción:

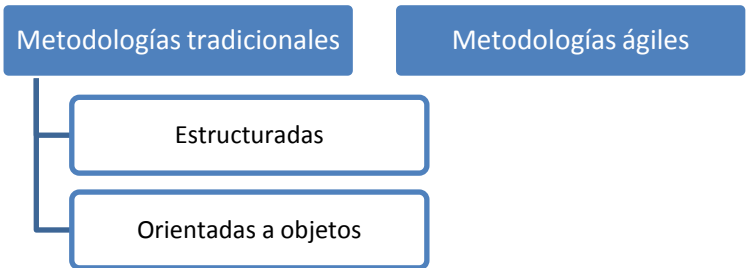

Figura 10 - Clasificación de metodologías

2.4.1 Metodologías tradicionales

Son aquellas caracterizadas por una fuerte planificación durante todo el proceso de desarrollo y por realizar una intensa etapa de análisis y diseño antes de la construcción del sistema, cuestión por las que también se conocen con el nombre de "Metodologías Pesadas"

Cabe realizar la siguiente clasificación en función del tipo de enfoque que utilizan para realizar las etapas análisis y diseño:

- **Estructuradas**
  Los datos se consideran separadamente de los procesos que los transforman y el comportamiento del sistema, tiende a desempeñar un papel secundario en la etapa de análisis, haciéndose un fuerte uso de la descomposición funcional.

  MÉTRICA, MERISE o SSADM son ejemplos de metodologías estructuradas.

- **Orientadas a objetos**
  Se considera el dominio del problema como un conjunto de objetos que interactúan entre sí.

  OOAD (Booch), OOSE (Jacobson), OMT (Rumbaugh), MÉTRICA son ejemplos de metodologías con enfoque orientado a objetos.

Por ejemplo, MÉTRICA v3, dispone de actividades que dan cobertura a ambos enfoques [11]:

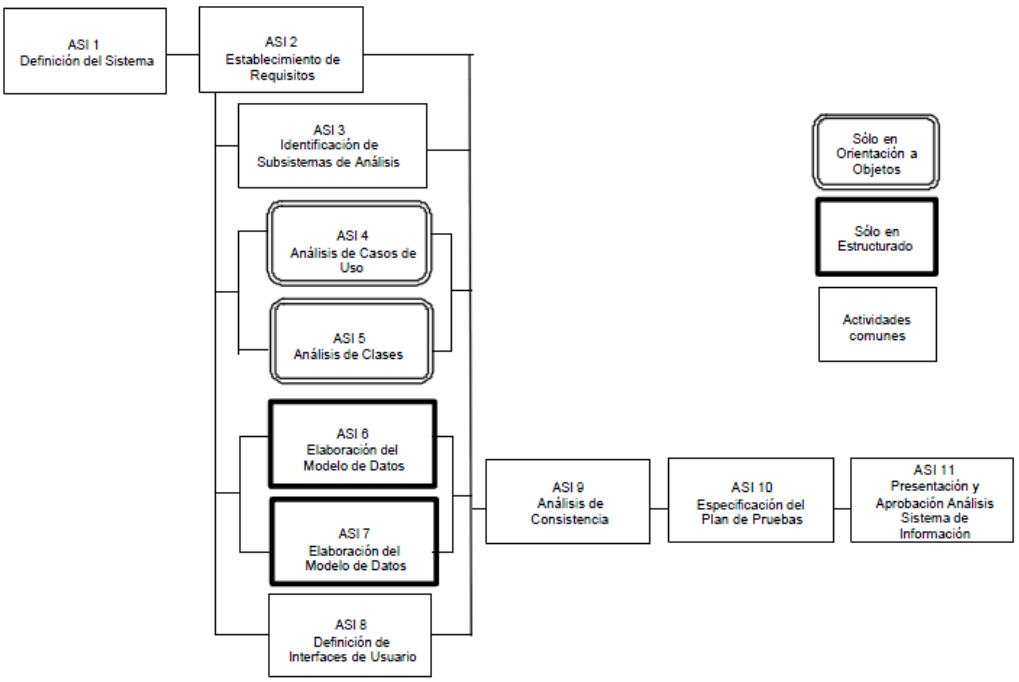

Figura 11 - Métrica V3, diagrama de actividades ASI

Es característico de las metodologías tradicionales, en general:

- Prestar especial atención al modelado y mantenimiento de los modelos.
- El énfasis en el control del proceso de desarrollo, metodología, el rigor de las actividades involucradas, las herramientas y la notación utilizada.
- El énfasis en la especificación detallada de procesos y un elevado número y especialización de roles.
- Limitar la participación del cliente a reuniones de control, reduciendo de manera significativa sus aportes.
- Asumir que no se van a presentar cambios una vez comenzado el proyecto y esperar que la arquitectura se defina tempranamente.
- Documentar de manera rigurosa toda actividad desarrollada en el proyecto.
- El aumento de los tiempos de espera por parte del usuario para ver los resultados.

2.4.2 Metodologías Ágiles

Los métodos ágiles surgen como respuesta a la dificultad de la aplicación práctica y efectiva de las metodologías tradicionales, para determinados proyectos. Esto es, porque asumen el cumplimiento de una serie de condiciones y restricciones durante la ejecución del proyecto que, en la práctica, casi nunca llegan a producirse y la rigidez normativa y la dependencia de planificaciones detalladas previas al desarrollo surgen como un escollo para obtener resultados satisfactorios.

Además, el progresivo aumento en el nivel de exigencia y expectativas de los usuarios y la necesidad de resultados más rápidos, sin detrimento de la calidad, dan pie a estos nuevos métodos.

En cualquier caso, la conveniencia o no de estos nuevos métodos es una cuestión de estudio para cada proyecto en particular.

Según las referencias consultadas las metodologías ágiles, en términos generales, ofrecen mayores probabilidades de éxito en proyectos que cumplan ciertas condiciones, siendo ejemplo típico de éstas:

- No involucrar a más de 15 o 20 desarrolladores
- No requerir más de cinco o seis meses de trabajo
- Un entorno de desarrollo apropiado para la comunicación entre los miembros del equipo
- Que el cliente o usuario esté dispuesto y disponible para el trabajo conjunto con el equipo de desarrollo.
- Que los requerimientos puedan ser formulados según va avanzando el proyecto.

Las diferencias más significativas entre las metodologías ágiles y tradicionales son las siguientes:

Tabla 1 - Metodologías tradicionales frente a Metodologías Ágiles

Metodologías Tradicionales		Metodologías Ágiles
Basadas en normas con origen en estándares del entorno de desarrollo	→	Basadas en heurísticas con origen en prácticas de codificación
Cierta resistencia a los cambios	→	Adaptación al cambio durante el proyecto
Impuestas externamente	→	Aceptadas internamente (por el equipo)
Proceso con mayor control, políticas y normas	→	Menor control del proceso, pocos principios
Existe un contrato prefijado	→	Existe un contrato flexible
La interacción entre el cliente y el equipo de desarrollo se realiza mediante reuniones	→	El cliente es parte del equipo de desarrollo
Grupos grandes y a menudo distribuidos	→	Grupos pequeños (<10) trabajando en un mismo lugar

Metodologías Tradicionales		Metodologías Ágiles
Más artefactos	→	Pocos artefactos
Más roles	→	Pocos roles
Menor énfasis en la arquitectura del software	→	La arquitectura del software es esencial y se expresa mediante modelos

A continuación se exponen conceptos de un conjunto de metodologías ágiles que se han considerado relevantes para este estudio:

2.4.2.1 XP – Programación Extema

Llamada Programación Extrema (eXtreme Programming) toma su nombre por teorizar un conjunto de principios y prácticas puramente de sentido común y llevarlos al extremo.

XP está especialmente indicado para dar respuesta a proyectos cuyos requisitos no están definidos y son cambiantes, y donde el riesgo técnico es elevado.

Los valores que inspiran XP son los siguientes:

Figura 12 - Valores que inspiran la programación extrema (XP)

- **Comunicación:** frecuente y fluida, alineando la visión del sistema de los programadores con la del cliente y favoreciendo la simplicidad.
- **Simplicidad**: comenzar con la solución más simple.
- **Retroalimentación**: a varios niveles, del resultado de las pruebas, del resultado de la ejecución de las tareas respecto a las estimaciones iniciales, etc.

- **Coraje**: también a varios niveles, trabajar para las necesidades presentes y no las futuras, alentando la mejora del código sin efectos externos (refactorización), persistencia frente a los problemas, etc.

De estos cuatro valores se concretan con cinco principios fundamentales:

- **Rapidez en la retroalimentación**: favoreciendo el aprendizaje, el tiempo que pasa entre una acción y su retroalimentación es crítico.

- **Asumir la simplicidad**: tratar cada problema asumiendo que su solución es la más simple, añadir complejidad sólo cuando realmente se necesite.

- **Cambio incremental**: evitar grandes cambios al mismo tiempo, solucionar los problemas de tamaño como secuencia de cambios más pequeños.

- **Aceptar el cambio**: resolviendo el problema más crítico preservando el mayor número de opciones.

- **Trabajo de calidad**: alentar que cada componente del equipo realice su trabajo con calidad, como mejor estrategia para que el producto final tenga calidad.

Siendo otros principios secundarios:

- **Enseñar a aprender**
- **Pequeña inversión inicial**
- **Jugar a ganar**
- **Experimentos concretos**
- **Comunicación abierta y honesta**
- **Trabajar con los instintos de las personas, no en contra.**
- **Aceptar la responsabilidad**
- **Adaptación local**
- **Viajar ligero de equipaje**
- **Mediciones honestas**

El ciclo de vida de un proceso XP consta de 6 fases, como se muestra en la siguiente figura:

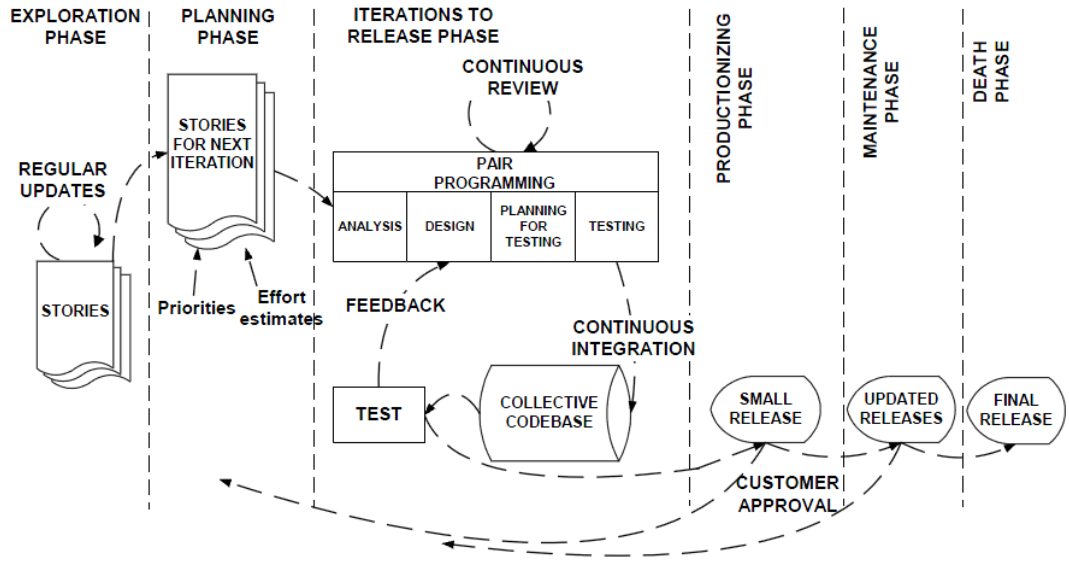

Figura 13 - Ciclo de vida del proceso en XP

- **Exploración**: el cliente expone en una serie de fichas "*story cards*" características que el Software debería cumplir en la siguiente iteración.

- **Planificación**: se establecen las prioridades de las story cards y se realiza una estimación de esfuerzos de la realización de las mismas, teniendo en cuenta que en la primera versión debe estar disponible en no más de 2 meses.

- **Iteraciones hasta la versión**: se construyen varias iteraciones del sistema, durante periodos de entre una y cuatro semanas de duración. La primera iteración está destinada a crear la arquitectura del sistema completo, las siguientes incluyen las story cards según la priorización del cliente y se realizan pruebas funcionales de las mismas.

- **Paso a producción**: se realizaran pruebas funcionales del sistema completo así como pruebas de rendimiento del mismo.

- **Mantenimiento**: una vez liberada la primera versión, se ocupará del mantenimiento del sistema y de la producción de nuevas versiones del mismo que den cobertura a nuevas necesidades del usuario.

- **Fin de proyecto**: La fase final del proyecto se producirá cuando entra en desuso, el software satisface las necesidades del cliente y éste no necesita características adicionales ni presenta defectos o problemas de rendimiento. También se produce si el resultado obtenido con el sistema ya no justifica su funcionamiento o si su coste del mantenimiento es excesivo.

XP define una serie de roles:

- **Cliente**: define los story cards y las pruebas funcionales que deberá cumplir el Software.
- **Programador**: codifica tanto el software como las pruebas unitarias.
- **Tester**: apoya al cliente en la definición de las pruebas, las ejecuta y difunde los resultados de las mismas al equipo de trabajo.
- **Tracker:** realiza el seguimiento de las tareas, comparando el progreso con las estimaciones iniciales y realimentando al equipo con esta información**.**
- **Coach**: responsable del proceso global, es un facilitador del uso de la metodología para el resto del equipo.
- **Consultor**: persona de apoyo, externo al equipo, que ayudará con su conocimiento específico en algún área relacionada con el proyecto en caso que sea necesario.
- **Gestor**: último responsable del proyecto, sirve de vínculo con el cliente y coordina y crea las condiciones adecuadas para que el trabajo se realice efectivamente.

Una misma persona podrá desempeñar varios roles, así, es posible que el rol de tracker, coach y gestor lo desempeñe una misma persona.

Además, XP promueve la aplicación disciplinada de las siguientes prácticas [12]:

- **El juego de la planificación**. El equipo realiza una estimación del esfuerzo requerido para el desarrollo de las story cards y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.

- **Entregas pequeñas y frecuentes:** se producen versiones parciales operativas del sistema en un tiempo reducido (no más de 3 meses), que constituyan un resultado de valor para el negocio.

- **Metáfora**: entendida como historia compartida por el cliente y el equipo que describe cómo debe 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**: son establecidas por el cliente antes de escribirse el código y se ejecutarán ante cada modificación del sistema, dirigiendo la producción de código.

- **Refactorización**: actividad de mejora del código sin alterar su comportamiento externo (eliminar duplicación, simplificarlo, flexibilizarlo mejorar su legibilidad y facilitar los posteriores cambios.

- **Programación en parejas**: Toda la producción de código se realiza con trabajo en parejas de programadores. Se obtendrán menor tasa de errores, mejor diseño, mayor satisfacción de los programadores, etc.

- **Propiedad colectiva del código:** Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

- **Integración continua:** Cada pieza de código es integrada en el sistema una vez que esté lista.

- **40 horas por semana**. El trabajo extra desmotiva al equipo, debiéndose trabajar un máximo de 40 horas por semana, en caso contrario probablemente exista un problema que debe corregirse.

- **Cliente in-situ:** el cliente conduce el trabajo hacia lo que aportará mayor valor de negocio y su implicación y disponibilidad para esta cuestión, por tanto, es crítica.

- **Estándares de codificación:** promueve el uso de estándares.

Estas prácticas se refuerzan entre sí, cuestión por la cual el máximo beneficio de las mismas se obtiene con su aplicación conjunta y equilibrada.

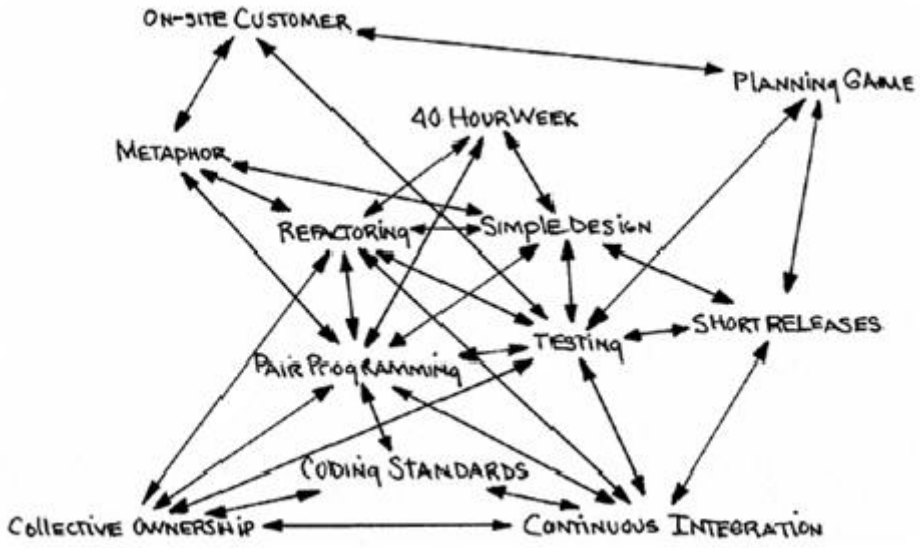

Figura 14 - Interrelación de las prácticas XP

2.4.2.2 SCRUM

Scrum es marco de trabajo iterativo e incremental para el desarrollo de proyectos, productos y aplicaciones.

Tiene su origen en el artículo "The new new product development" de Nonaka y Takeuchi en 1986 en el que se examinan las pautas de trabajo comunes de varias empresas en productos de éxito. En éstas, el trabajo no recorría fases a través de distintos equipos especializados, sino que un equipo altamente cualificado y multidisciplinar trabajaba conjuntamente desde el principio hasta el final. Estos autores utilizan por primera vez el término SCRUM (melé), como comparación de esta forma de trabajar con el término que se utiliza en rugby para designar la jugada en que el equipo forma un bloque compacto para seguir todos a una un fin común.

Formalmente, no es hasta 1995 en la "OOPSLA Business Object Design and Implementation Workshop" la presentación y definición del marco de trabajo de SCRUM, en el estudio "Scrum Development Process", por Ken Schwaber y Jeff Sutherland.

Los roles, artefactos y eventos principales que definen SCRUM se presentan en la siguiente figura [13]:

Figura 15 - Esquemático del ciclo de vida de SCRUM

Esta metodología se estructura en SPRINTS, ciclos de trabajo o iteraciones de duración fija (1 a 4 semanas) que se irán sucediendo, terminándose en una fecha específica aunque no se haya terminado el trabajo, nunca se alargan. Al comienzo de cada Sprint, se seleccionan los requisitos de una lista priorizada y se compromete el alcance del Sprint, durante el cual no se podrá cambiar.

Todos los días el equipo se reúne brevemente para informar del progreso, y actualizan unas gráficas sencillas que les orientan sobre el trabajo restante.

Al final de un Sprint, el equipo revisa el resultado de la construcción con los interesados del proyecto, obteniéndose comentarios y observaciones que se podrán incorporar al siguiente Sprint.

Se pone énfasis en que los productos deben funcionar al final del Sprint, que el código esté integrado, completamente probado y sea potencialmente entregable.

Principales roles en SCRUM

En esta metodología destacan principalmente el papel jugado por los siguientes roles:

- **Dueño de producto (DP):** es responsable de maximizar el retorno de inversión (ROI) del producto, entendido como los elementos de más valor de negocio y menos coste. Para esto, identifica y prioriza las funcionalidades para el siguiente Sprint e interactúa activa y frecuentemente con el equipo y revisa el resultado de cada iteración. En aplicaciones internas DP y cliente pueden ser la misma persona.

- **Equipo:** su papel principal es construir el producto. Dispone de todas las competencias y habilidades para entregar el producto (multifuncional) y goza de una elevada autonomía y responsabilidad (auto-gestionado), teniendo capacidad para decidir sus compromisos y cómo solventarlos.

- **Scrummaster:** es un facilitador de la metodología, con un sólido conocimiento de la misma e implicación, ayudando al DP y al equipo a entenderla y aplicarla con éxito. Puede ser un miembro del equipo pero nunca el DP.

Hágase notar el cambio en la filosofía de trabajo de los responsables: las labores de asignación tareas, informes de estado y otras tareas de gestión se sustituyen por labores de mentoring, coaching, ayuda a la resolución de problemas, aporte de ideas creativas y al desarrollo guiado de las habilidades de los miembros del equipo.

2.4.2.3 RUP

Rational Unified Process se define como un proceso de Ingeniería de Software cuyo objetivo es la producción de software de calidad que satisface los requisitos del usuario final, en un plazo y coste predecibles.

El proceso de desarrollo de software propuesto por RUP tiene tres características esenciales: está dirigido por los Casos de Uso, está centrado en la arquitectura, y es iterativo e incremental.

Dirigido por Casos de Uso

Los casos de uso se utilizan para la toma de requisitos, representando aquellas funcionalidades del sistema que proporcionarán al usuario un valor añadido. Sirven de guía para el diseño del sistema, su implementación y las pruebas, constituyendo una guía de trabajo y un elemento integrador que permitirá la trazabilidad entre los artefactos que son generados en las diferentes actividades del proceso de desarrollo.

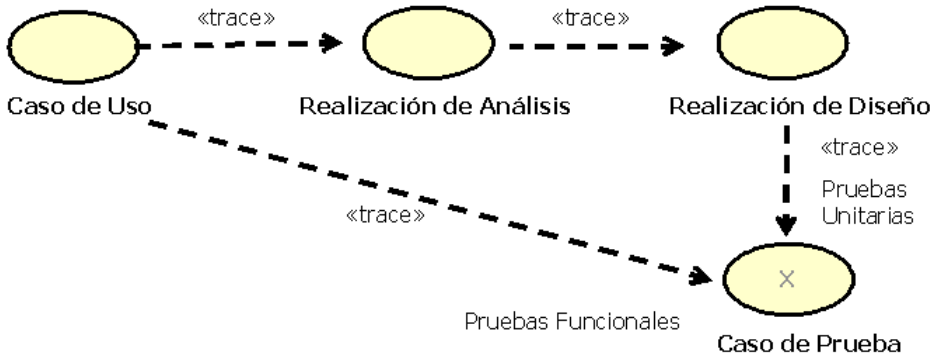

Figura 16 - R.U.P. Los casos de uso, elemento integrador

Proceso centrado en la arquitectura

Entendida como la organización de las partes más relevantes del sistema, permitiendo una perspectiva clara del sistema completo y una visión común para el equipo de desarrollo y usuarios.

Se han de tener en cuenta cuestiones como requisitos no funcionales del sistema (sistema operativo, gestor de bases de datos, protocolos, sistemas heredados, etc.) y elementos de calidad del sistema, rendimiento, reutilización y capacidad de evolución.

Arquitectura y Casos de Uso están estrechamente relacionados y evolucionan en paralelo durante todo el proceso de desarrollo: la arquitectura debe permitir el desarrollo de todos los Casos de Uso, actualmente y en el futuro.

Para definir la arquitectura se utiliza un modelo compuesto por múltiples vistas [14]. En el modelo 4+1, el equipo define el diseño en cuatro vistas o perspectivas y utiliza una quinta para validarlo.

Cada vista se centra en distintos aspectos del sistema y controlan de manera independiente requisitos funcionales y no funcionales:

- **Vista Lógica:** Muestra la estructura estática del sistema.
- **Vista Física:** Muestra el despliegue de la aplicación en la red de computadoras.
- **Vista de Procesos:** Muestra los hilos y procesos de ejecución así como la comunicación entre estos.
- **Vista de Desarrollo:** Muestra la estructura en modelos del código del sistema.
- **Vista de Escenarios o Casos de Uso**: contiene un conjunto representativo de los casos de uso que ilustran lo definido en el resto de vistas.

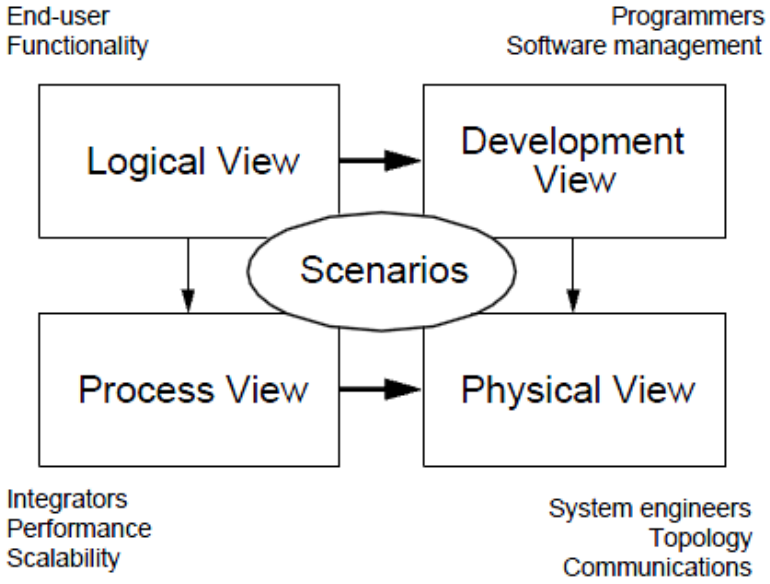

Figura 17 - R.U.P. Modelo 4+1

Iterativo e incremental

El proceso está dividido en cuatro fases y en cada una de ellas se pueden dar cuantas iteraciones se consideren necesarias. De cada iteración resultará un incremento del sistema que añadirá funcionalidad o mejorará la existente.

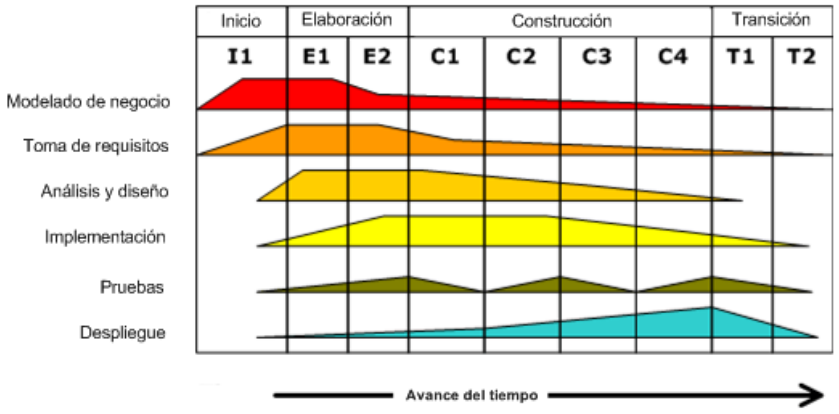

Figura 18 - R.U.P. Ciclo de vida

Como podemos ver en el gráfico, el esfuerzo en cada una de las actividades que tienen lugar (modelado de negocio, toma de requisitos, etc.) variará en función de la fase en que nos encontremos y la iteración dentro de ésta.

Fase de inicio

Es la fase de menor duración del proyecto. Debe ser relativamente breve para evitar un gran número de especificaciones iniciales que puedan desviar el proyecto por su posible variación en el tiempo.

Los objetivos fundamentales de la misma son:

- Establecer una justificación para caso de negocio a cubrir
- Establecer el alcance y limitaciones del proyecto
- Identificar los casos de uso y requisitos clave que pueden condicionar el diseño
- Identificar un conjunto de arquitecturas candidatas
- Identificar los riesgos del proyecto
- Preparar un plan de proyecto y una estimación de costes preliminar

Fase de elaboración
Los dos objetivos fundamentales de esta fase serán abordar los factores de riesgo identificados en la fase anterior y establecer y validar una arquitectura del sistema.

La validación de la arquitectura se realiza mediante la obtención de una arquitectura ejecutable "línea de base", una implementación parcial del sistema que incluirá el núcleo del mismo y sus componentes más significativos. Su construcción se realizará de manera iterativa y marcará el final de la fase la disponibilidad de esta de una arquitectura estable y con un buen comportamiento en cuanto a su escalabilidad, rendimiento y coste.

Por último se obtendrá un plan de proyecto detallado para la construcción, así como su estimación de costes.

Fase de construcción
Sobre la arquitectura de base, se irá construyendo el sistema en una serie de iteraciones cortas. El objetivo de cada iteración será obtener un incremento del sistema en funcionamiento y preparar el conjunto de casos de uso que se abordarán en la siguiente.

Fase de transición
En la fase final el sistema se despliega y se pone a disposición de los usuarios. El feedback obtenido acerca del sistema puede dar lugar a iteraciones en esta fase que lo irán refinando. Cuando el conjunto de requisitos se amplía o varía significativamente, da lugar a un nuevo ciclo inicio → elaboración → construcción → transición.

2.4.2.4 Otras metodologías ágiles

Otras metodologías ágiles sobre las que existe literatura pero no han sido revisadas son las siguientes:

- Crystal Clear
- Feature Driven Development (FDD)
- Adaptive Software Development (ASD)
- Agile Unified Process (AUP)
- Essential Unified Process (EssUP)
- Open Unified Process (OpenUP)

2.5 Modelo de Madurez del Software

CMMI (Capability Maturity Model Integration) es un modelo de calidad que establece una serie de pautas para mejorar los procesos de desarrollo y de gestión del desarrollo, adquisición y mantenimiento de productos y servicios. Este modelo integra, además de las prácticas propias de la Ingeniería del Software, prácticas para la Ingeniería de Sistemas y de adquisición de software.

CMMI surge para dar una aproximación sistémica a los problemas a los que se enfrentan la mayoría de las organizaciones dedicadas a la Ingeniería del Software. Para esta cuestión, define una serie de áreas proceso a implementar, especificando para cada una de ellas objetivos, prácticas esperadas y subprácticas recomendadas. Adicionalmente, dispone una serie de prácticas genéricas que pueden aplicarse a todos los procesos.

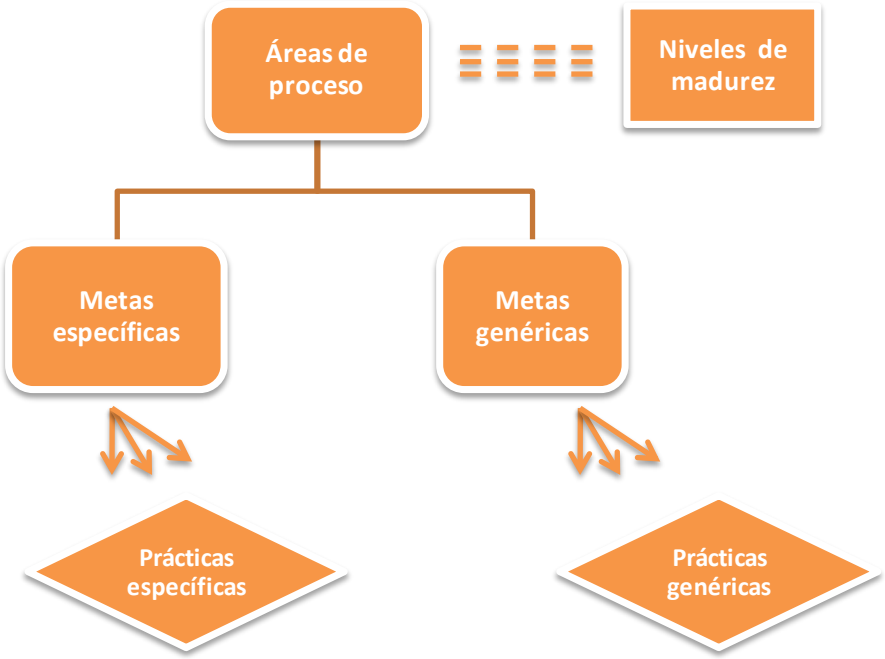

Figura 19 – CMMI: representación de áreas de proceso

Algunas de estas áreas de proceso son:

- Análisis de Causas y Resolución (CAR)
- Gestión de la configuración (CM)
- Gestión de Requerimientos (REQM)
- Desarrollo de Requerimientos (RD)
- Gestión Integrada de Proyectos (IPM)

Además de la prescripción del camino a seguir para la mejora, CMMI aporta una clasificación de las organizaciones en función de la madurez de sus procesos en cinco niveles:

- **Madurez 1 – Inicial**: ausencia total de procesos definidos.
- **Madurez 2 – Repetible**: procesos de administración establecidos para seguimiento de coste, tareas y funcionalidad.
- **Madurez 3 – Definido**: se incorporan actividades de administración de ingeniería documentada, estandarizada e integradas en una familia de procesos. Los proyectos utilizan una versión adaptada de esas normas para su desarrollo.
- **Madurez 4 – Administrado:** los proyectos se ejecutan en forma controlada, con métricas que permiten realizar mediciones confiables de procesos y productos.
- **Madurez 5 – Optimizado:** se incluye la mejora continua de procesos partiendo de la comparación y el análisis de las sucesivas mediciones de los proyectos.

A este modelo, se incorpora una vista de niveles de capacidad de los procesos para ofrecer, cuando así se necesita por los objetivos de negocio, una capacidad diferenciada por área de proceso:

- **Capacidad 0 – Incompleto:** área de proceso sin objetivos.
- **Capacidad 1 – Ejecutada:** se satisfacen objetivos específicos del área de proceso..
- **Capacidad 2 – Administrada:** el área de proceso está institucionalizada a partir de una política organizacional de uso de los procesos.
- **Capacidad 3 – Definida:** área de proceso institucionalizada a partir de un proceso definido.
- **Capacidad 4 – Cuantitativamente Administrada:** área de proceso institucionalizada a partir de una política organizacional de la administración cuantitativa de procesos.
- **Capacidad 5 – Optimizada:** área de proceso institucionalizada a partir de la optimización de sus procesos.