**1. Análisis de Sistemas de Software**

**1.1. Tipos de requerimientos**

**1.1.1. Requerimientos funcionales**

Define de los servicios que el sistema debe proporcionar, cómo debe reaccionar a una entrada particular y cómo se debe comportar ante situaciones particulares.

* **De sistema:** Establecen con detalle los servicios y restricciones del sistema. El documento de requerimientos del sistema, algunas veces denominado especificación funcional, debe ser preciso
* **De usuario:** Son un documento donde describes “qué” debe hacer el sistema en términos no técnicos y debe ser lo más detallado posible para evitar ambigüedades.

**1.1.2. Requerimientos no funcionales**

Son los que actúan para limitar la solución, se les conoce como restricciones o requisitos de calidad.

* **De producto:** Estos especifican límites o restricciones sobre el comportamiento del producto.
  + **Usabilidad:** Esfuerzo que hace el usuario para aprender el usar el software.
  + **Eficiencia:** Relacionado con desempeño en cuanto a tiempo de respuesta, número de operaciones por segundo, consumo de recursos de memoria, procesador, espacio en disco o red.
  + **Dependibilidad:** Engloba varios atributos
    - **Disponibilidad:** Disposición del sistema para prestar servicio correctamente.
    - **Confiabilidad:** Continuidad del servicio prestado por el sistema.
    - **Seguridad industrial:** Ausencia de consecuencias catastróficas para el usuario o el ambiente.
    - **Integridad:** Ausencia de alteraciones inadecuadas al sistema.
    - **Mantenibilidad:** Posibilidad de realizar modificaciones o reparaciones a un proceso sin afectar la continuidad del servicio.
  + **Seguridad:** Cumplir con atributos en el área de seguridad de tecnología de información, seguridad de datos, seguridad lógica, control de acceso a información (restricciones de acceso), autenticidad de la información, privacidad, etc.
* **Organizacionales:** Se derivan de las políticas y procedimientos de la organización como por ejemplo estándares de procesos o requerimientos de implementación.
  + **Entorno:** Describen el ambiente operativo en el que se debe desenvolver el sistema (los fierros).
  + **Operacionales:** Procedimientos operativos que describen cómo será usado el sistema dentro del contexto de la organización.
  + **Desarrollo:** Lenguaje de programación a usar, estándares de codificación, patrones (y antipatrones) de diseño y programación, herramientas para gestionar el desarrollo de software, entorno de desarrollo de software (ambiente de desarrollo), entorno de pruebas de software (ambiente de pruebas), etc.
* **Externos:** Factores externos al software que influyen sobre este como temas financieros, interoperabilidad con otros sistemas, regulaciones de salud, temas legales, licencias, regulaciones, certificaciones.
  + **Regulatorios:** Leyes y reglamentos que establecen que debe hacer el sistema y cómo debe hacerlo para cumplirlas. El foco de un sistema o nueva funcionalidad puede ser exclusivamente para cumplir una regulación.
  + **Éticos:** Requerimientos que aseguran que el sistema será aceptable para el usuario, público en general y se adapta a las costumbres de la sociedad en la que se desenvuelve o a la que presta servicios.
  + **Legislativos:** Características que debe cumplir el sistema para cumplir con la ley, por ejemplo, en el área de contabilidad (normas contables y estándares financieros), requerimientos de seguridad industrial (para sistemas críticos), entre otros aspectos.

**1.2. Técnicas y herramientas para la obtención, análisis, priorización y validación de requerimientos**

**1.2.1. Herramientas para la obtención de requerimientos**

* **Entrevistas**
  + Obtenemos información cualitativa
  + Requiere mayor preparación y experiencia por parte del analista
  + Comunicación uno a uno
  + Asegurarse de entrevistar al personal adecuado
* **Cuestionarios**
  + Permite reunir información proveniente de un grupo grande de personas
  + Puede proporcionar datos más confiables que otras técnicas
  + La respuesta puede ser limitada por falta de interés del encuestado
  + Asegurarse de encuestar al personal adecuado
* **Análisis de tareas**
  + Consiste en visitar la empresa y observar las tareas que se realizan
* **Análisis de dominio**
  + Analizar entorno de trabajo de la empresa y su modelo de negocio
  + Generar un diagrama de clases que represente todos los objetos involucrados en su modelo de negocio y flujo de actividades.
  + Ejemplo de la estación del metro
* **Introspección**
  + Observar la interacción del sujeto con el sistema o al realizar su trabajo
* **Grupo de trabajo**
  + Tipo reuniones JAD
  + Participan usuarios expertos del dominio de la empresa
* **Lluvia de ideas**
  + Obtener ideas y evaluarlas
* **Desarrollo conjunto de la aplicación**
  + Sesión intensiva para recopilar datos
  + El equipo está conformado por los principales usuarios, administradores y analistas de sistemas
  + Seleccionar de 8 a 12 usuarios que sepan explicar que necesitan para realizar sus tareas y den sugerencias para nuevas funcionalidades
* **Observación**
  + Planificar bien que aspectos se van a observar
  + Observar a tomadores de decisiones
  + Método STROBE
* **Diagrama de caso de uso**
  + Identificar a los actores y sus casos de uso
* **Prototipos**
  + Desarrollar prototipos
  + Los usuarios prueban el prototipo y retroalimentan en base a su experiencia
* **Escenarios**
  + Documentan el comportamiento del sistema cuando se le presentan eventos específicos
  + Los escenarios incluyen:
    - una descripción del flujo de datos
    - las acciones del sistema
    - la documentación de las excepciones que puedan surgir

**1.2.2. Herramientas para el análisis de requerimientos**

* **Matriz de trazabilidad**
  + Registrar las relaciones existentes entre la necesidad dada por un interesado (stakeholder), el requerimiento de proyecto y la solución implementada.
  + Nos ayuda a identificar
    - La necesidad original de requerimientos específicos que se estén cambiando
    - Los interesados que debemos involucrar en el cambio de alcance
    - Entregables del proyecto que serán afectados.
* **Descomposición funcional**
  + Se realiza para identificar y entender los componentes o partes que constituyen un todo (o función global)
  + Lograr que la función global pueda ser reconstruida a partir de sus partes
  + Es vital identificar las interacciones entre componentes
* **Especificación por sentencias textuales**
  + Especificaciones textuales en lenguaje natural, que se documentan en matrices de trazabilidad o definiciones del alcance
  + No usa herramientas visuales como los casos de uso
  + Simplemente es una descripción más detallada del requerimiento en lenguaje natural
* **Checklists**
  + Preguntas o revisiones que se realizan sobre los requerimientos de software que sean presentados de forma escrita
  + Sirve de marco de trabajo y procedimental para revisar el requerimiento, facilitando su análisis de forma estructurada
  + Los requerimientos se pueden revisar sobre la matriz de trazabilidad de requerimientos o sobre la definición del alcance
* **Inspección**
  + Revisar el software de forma visual para confirmar que los requerimientos estén presentes.
* **Análisis de impacto**
  + Prever efectos que tendrá un cambio propuesto, identificando qué elementos son los que se verán afectos y de qué forma lo serán
  + En particular los códigos fuente y los elementos de las bases de datos
* **Trazabilidad – dependencia**
  + Se utiliza cuando se presenta una solicitud de cambio
  + Consiste en un conjunto de
    - cada pregunta tiene unas posibles respuestas
    - cada respuesta tiene asociado un puntaje
    - El puntaje total de las respuestas seleccionadas define el nivel de impacto de acuerdo a políticas establecidas
    - El impacto es una medida que define el nivel de efecto que un cambio tiene sobre los usuarios y procesos de negocio
  + Si el cambio es factible se realiza, de lo contrario se rechaza
* **Entrevistas**
  + Se basa llevando el proceso de solicitud de cambio mediante una entrevista
  + Se debe describir el problema de forma detallada

**1.2.3. Herramientas para la priorización de requerimientos**

**Aspectos de la priorización**

* Importancia
* Costo
* Castigo
* Volatilidad
* Riesgo

**Proceso analítico de jerarquización – AHP**

* Consiste en analizar todas las posibles paridades de requerimientos clasificados de manera jerárquica
* La escala que va de 1 a 9, siendo 1 la de más baja importancia y 9 la más alta
* El número total de comparaciones que se llevan a cabo es “n x (n-1)/2”, siendo n la cantidad de requerimientos

**Asignación numérica – agrupamiento**

* Consiste en “entregar” a los stakeholders 100 unidades imaginarias pudiendo ser horas o dinero, para que las distribuyan entre los requerimientos, viendo de esta manera cuáles requerimientos tienen un mayor “valor”.
* El problema con esta técnica se presenta cuando se tiene una gran cantidad de  
  requerimientos ya que se disminuye la cantidad de unidades que se puede asignar a cada uno

**Ranking**

* Basada en una escala ordinal
* A los requerimiento se les asigna una posición en el ranking, que va de 1 a n, siendo n la cantidad de requerimientos
* A diferencia del proceso de jerarquía analítica, el puesto 1 es el de mayor importancia

**Requerimientos top – ten**

* Consiste en hacer que los usuarios escojan su “top 10” de requerimientos, sin tener  
  necesariamente un orden específico dentro de los mismos
* Es de gran utilidad para cuando se tiene varios stakeholders con puntos de vista distintitos

**1.2.4. Herramientas para la validación de requerimientos**

**Aspectos de la priorización**

* Validez
* Consistencia
* Completitud
* Realismo

**Prototipado de interfaz de usuario**

* Elaborar prototipos de interfaces de usuario
* Hay dos tipos
  + Desechables
    - Se utilizan sólo para la validación de los requisitos
    - Posteriormente se desechan
    - Pueden ser prototipos en papel o en software
  + Evolutivos
    - Una vez utilizados para la validación de los requisitos se mejora su calidad y se convierten progresivamente en el producto final

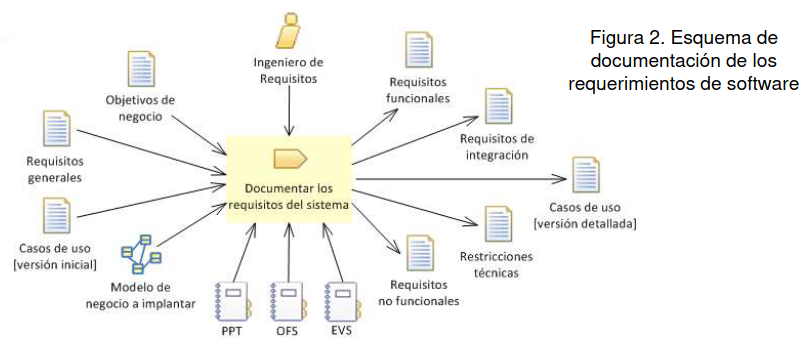
**Recorrido de casos de uso**

* Asociada a la inspección de código fuente
* El objetivo es encontrar conflictos en el producto que se revisa
* Permite plantear alternativas
* Los participantes aumentan su conocimiento del producto en cuestión

**1.3. Técnicas y herramientas de documentación de requerimientos**

**Objetivo de la documentación de requerimientos**

El objetivo principal de la tarea de "documentación de los requisitos del software" es continuar con la definición del programa a desarrollar, tomando como punto de partida los requisitos generales y los casos de uso en su versión inicial, y considerando los objetivos del negocio y el modelo de negocio a implantar.  
En los casos de uso que se considera necesario se van completando con más información y los requisitos generales se van detallando en requisitos funcionales, no funcionales, de integración y en restricciones técnicas.



**1.3.1. Herramienta REM**

**¿Qué es REM?**

REM (Requirements Management) es una herramienta experimental gratuita de Gestión de Requisitos diseñada para soportar la fase de Ingeniería de Requisitos de un proyecto de desarrollo software de acuerdo con la metodología definida en la Tesis Doctoral "Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de información”.

**2. Diseño de Sistemas de Software**

**2.1. Diseño arquitectónico de software**  
**2.2. Diseño de módulos, componentes y de datos de software**  
**2.3. Diseño de interfaces**

**3. Desarrollo de Sistemas de Software**

**3.1. Lenguajes de desarrollo de software**  
**3.2. Paradigmas de programación**

**¿Qué son los paradigmas de programación?**

Es una manera o estilo de programación de software. Se trata de un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas para resolver problemas computacionales.

**Paradigma imperativo**

Los programas consisten en una sucesión de instrucciones o conjunto de sentencias, como si el programador diera **órdenes concretas**. El desarrollador describe en el código **paso por paso** todo lo que hará su programa.

Otros enfoques subordinados al paradigma de programación imperativa son:

* **Programación estructurada:** Es un tipo de programación donde el flujo de control va de arriba hacia abajo y se define mediante bucles anidados, condicionales y subrutinas
* **Programación procedimental:** Este paradigma de programación consiste en basarse en un número muy bajo de expresiones repetidas, englobarlas todas en un procedimiento o funcióny llamarlo cada vez que tenga que ejecutarse.
* **Programación modular:** consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más manejable y legible. Se trata de una evolución de la programación estructurada para resolver problemas de programación más complejos.

**Paradigma declarativo**

En este paradigma el código no explica detalladamente como se resuelve un problema, sino que usa funciones que simplifican dichas tareas, pero en realidad el desarrollador no sabe cómo trabaja exactamente esa función de forma interna, por ejemplo, un SELECT en SQL.

Este paradigma a su vez se divide en dos:

* **Programación lógica:** Se usa para inteligencia artificial.
* **Programación funcional:** Utiliza lambdas y streams.

**Paradigma de programación orientada a objetos**

En este modelo de paradigma se construyen modelos de objetos que representan elementos (objetos) del problema a resolver, que tienen características y funciones.

Permite separar los diferentes componentes de un programa simplificando su creación, depuración y posteriores mejoras.

Disminuye los errores y promociona la reutilización del código.

La programación orientada a objetos se sirve de diferentes conceptos como:

* Abstracción
* Encapsulación
* Herencia
* Polimorfismo

**Programación reactiva**

Este paradigma se basa en escuchar lo que emite un evento o cambios en el flujo de datos, en donde los objetos reaccionan a los valores que reciben de dicho cambio. Las librerías más conocidas son Project Reactor y RxJava, React/Angular usan RxJs para hacer uso de la programación reactiva.

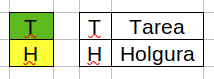
**3.3. Entornos de desarrollo**  
**3.4. Gestión de datos**  
**3.5. Plataformas de desarrollo**

**4. Gestión de Proyectos de Software**

**4.1. Gestión de tiempos, costos, recursos humanos y de riesgo**

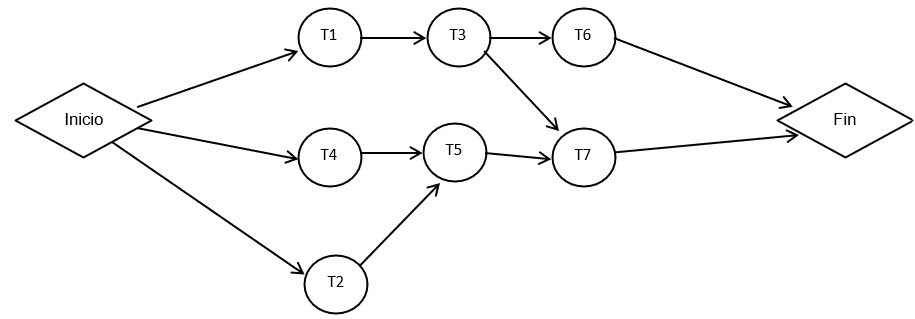
**4.1.1. Diagrama de Pert**

**Simbología de los diagramas**

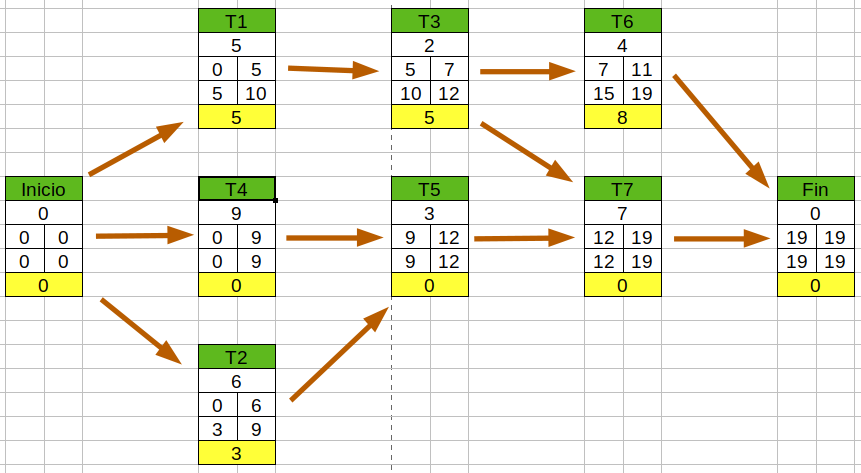
 

**Ejercicio 1:** Encontrar la ruta crítica.

|  |  |  |
| --- | --- | --- |
| **Tarea** | **Tiempo (meses)** | **Precedencia** |
| T1 | 5 |  |
| T2 | 6 |  |
| T3 | 2 | T1 |
| T4 | 9 |  |
| T5 | 3 | T4 |
| T6 | 4 | T3 |
| T7 | 7 | T3, T5 |



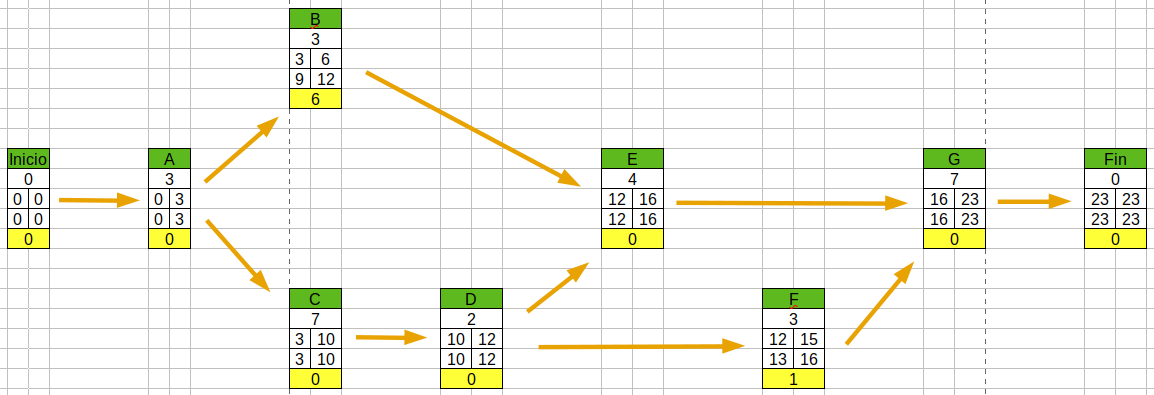
**Solución:** La ruta crítica es Inicio –> T4 –> T5 –> T7 –> Fin



**Ejercicio 2:** Encontrar la ruta crítica.

|  |  |  |
| --- | --- | --- |
| Tarea | Tiempo (meses) | Precedencia |
| A | 3 |  |
| B | 3 | A |
| C | 7 | A |
| D | 2 | C |
| E | 4 | B, D |
| F | 3 | D |
| G | 7 | E, F |

**Solución:** La ruta crítica es Inicio –> A –> C –> D –> E –> G –> Fin



**Ejercicio 3:** Encontrar la ruta crítica.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Tarea** | **Precedencia** | **Tiempo (meses)** | **ES** | **EF** | **LS** | **LF** | **H** |
| **Inicio** | - |  | 0 | 0 | 0 | 0 | 0 |
| **A** | **-** | **4** | 0 | 4 | 1 | 5 | 1 |
| **B** | **-** | **8** | 0 | 8 | 0 | 8 | 0 |
| **C** | **-** | **3** | 0 | 3 | 6 | 9 | 6 |
| **D** | **A** | **3** | 4 | 7 | 5 | 8 | 1 |
| **E** | **A** | **6** | 4 | 10 | 8 | 14 | 4 |
| **F** | **C** | **5** | 3 | 8 | 9 | 14 | 6 |
| **G** | **B, D** | **6** | 8 | 14 | 8 | 14 | 0 |
| **Fin** | E, F, G |  | 14 | 14 | 14 | 14 | 0 |

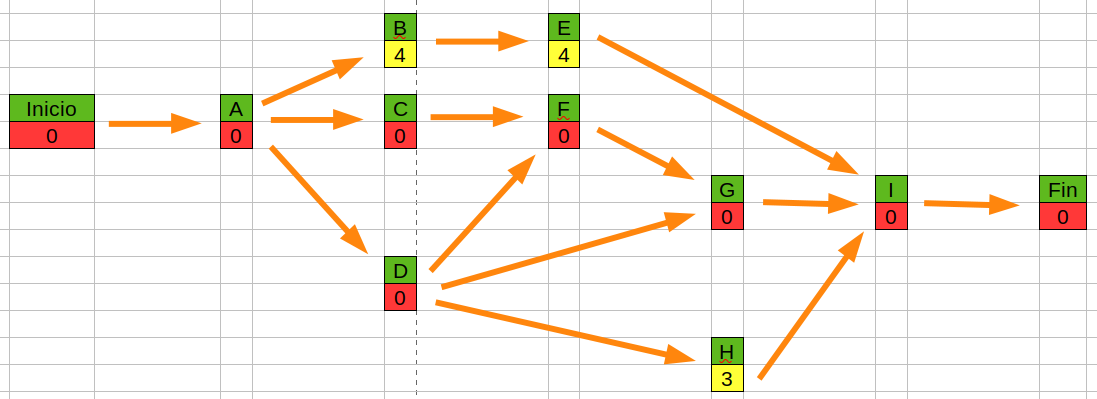
**Solución:** La ruta crítica es Inicio –> B –> G –> Fin

**Ejercicio 4:** Encontrar la ruta crítica.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Tarea** | **Precedencia** | **Tiempo (meses)** | **ES** | **EF** | **LS** | **LF** | **H** |
| **Inicio** | - |  | 0 | 0 | 0 | 0 | 0 |
| **A** | **-** | **3** | 0 | 3 | 0 | 3 | 0 |
| **B** | **A** | **2** | 3 | 5 | 7 | 9 | 4 |
| **C** | **A** | **4** | 3 | 7 | 3 | 7 | 0 |
| **D** | **A** | **4** | 3 | 7 | 3 | 7 | 0 |
| **E** | **B** | **6** | 5 | 11 | 9 | 15 | 4 |
| **F** | **C, D** | **6** | 7 | 13 | 7 | 13 | 0 |
| **G** | **D, F** | **2** | 13 | 15 | 13 | 15 | 0 |
| **H** | **D** | **3** | 7 | 10 | 12 | 15 | 3 |
| **I** | **E, G, H** | **3** | 15 | 18 | 15 | 18 | 0 |
| **Fin** | I |  | 18 | 18 | 18 | 18 | 0 |

**Solución:** Las rutas críticas son

* Inicio –> A –> D –> G –> I –> Fin
* Inicio –> A –> C –> F –> G –> I –> Fin
* Inicio –> A –> D –> F –> G –> I –> Fin



**4.1.3. Proyectos de riesgo**

**4.1.4. Hojas de verificación de proyectos**

**4.1.6. Método de análisis por puntos**

**¿Qué es el análisis por puntos o APF?**

Es considerada la **principal herramienta para la medición funcional de productos de software y de los procesos involucrados en su desarrollo**. Esta comprende conceptos que pueden ser entendidos tanto por el desarrollador como por los usuarios.

Esta técnica nos permite estimar la **duración y costo** de un proyecto en base a las **funcionalidades y características** que posee el software.

**4.1.7. Método Wideband Delphi**

Es una técnica de estimación estructurada en grupo. Se basa fundamentalmente en que varios expertos creen estimaciones ante ciertos planteamientos y generen respuestas, estas respuestas se evaluarán y se espera que se llegue a una media entre las opiniones de los expertos.

**Características**

* **Es un proceso iterativo**: Esto permite a los especialistas reflexionar sus opiniones y emitir nuevos juicios.
* **Se mantiene en el anonimato**: Esto evita las influencias negativas del grupo.
* **Se tiene una retroalimentación controlada:** El grupo coordinador transmite la posición a los expertos, destaca las participaciones significativas.
* **Respuesta estadística del grupo:** Se manejan valores numéricos de cada respuesta y se obtiene la mediana.

**Procedimiento**

* **Definición del problema:**
* **Establecer un grupo coordinador:** 
  + De 2 a 5 personas
  + Estudiar y afinar el proceso de trabajo
  + Aprobar la lista de expertos
  + Prepara cuestionarios
  + Realiza retroalimentación
  + Supervisa la marcha del proceso
  + Toma medidas cuando sea necesario
* **Formar grupo de expertos:**
  + Al menos 3 expertos, la cantidad depende de la problemática
  + Ellos emiten los juicios
  + Son el eje principal del método
  + Se les debe dar mucha retroalimentación (por email, teléfono, etc.) para evitar que abandonen desde la primera ronda.
* **Etapas de iteración**
  + Ronda 1

1. Los coordinadores elaboran una pregunta abierta sobre cuáles son los temas y elaboran un cuestionario para enviárselo a los expertos.

2. Los expertos responden el cuestionario que les fue enviado y envían la respuesta a los coordinadores.

* + Ronda 2

1. Con las respuestas obtenidas, los coordinadores elaboran otro cuestionario para enviárselo a los expertos.

2. Los expertos responden este segundo cuestionario y envían de nuevo la respuesta a los coordinadores.

* Ronda 3, 4, etc.
  + Sigue el mismo flujo de la ronda 2
  + Generalmente se manejan solo 2 o 3 rondas
  + El objetivo del método es alcanzar el criterio de estabilización, es decir que se obtenga una estabilidad satisfactoria, esto analizando las respuestas de las preguntas, con ello nos daremos cuenta del número de rondas que necesitaremos.

**Ventajas**

* Se eliminan los procesos cara a cara
* Permite la participación de un gran número de personas sin que haya caos.
* El horizonte de análisis puede ser muy variado

**Desventajas**

* El tiempo de ejecución puede llevar hasta 45 días en promedio
* El costo, ya que la gente involucrada es experta y querrán que se les page como tal.
* El éxito del método depende de la calidad de los participantes y que no haya sesgos en la elección correcta.

**4.1.8. Método MosCow**

El método MoSCoW es una técnica de clasificación de requisitos, actividades, tareas o historias de usuario, en base a la prioridad que cada una de estas exige.

Esto nos ayuda a saber que actividades son más indispensables que otras y poder definir el orden en que debemos realizarlas.

**Siglas**

* **M (Must have):** Tareas de prioridad alta que deben realizarse sin excepciones.
* **S (Should have):** Tareas de prioridad no crítica, pero que deberían realizarse.
* **C (Could have):** Tareas deseables, pero no estrictamente necesarias.
* **W (Won’t have):** Tareas que se tiene claro que por el momento no se van a realizar.

**4.1.9. Talleres Workshop**

Consiste en realizar talleres, entre determinadas personas, en las que los asistentes aportan, en pie de igualdad, conocimientos, impresiones, ideas y preguntas acerca del tema sobre el que versa el taller.

Un Workshop no es lo mismo que una reunión, tienen propósitos diferentes.

**Características de una Reunión**

* **Propósito:** Compartir información entre los miembros del equipo.
* **Alcance:** Permiten una cobertura superficial de muchos temas.
* **Longitud:** Se miden en medias horas u horas.
* **Estructura:** Son de naturaleza pasiva donde los asistentes escuchan y absorben.
* **Preparación:** La preparación es menor en comparación con el workshop debido a que el workshop es más complejo.

**Características de un Workshop**

* **Propósito:** Son un método para resolver un problema, desarrollar un plan o alcanzar una decisión.
* **Alcance:** Cubren una cobertura profunda de un problema
* **Longitud:** Se miden en medios días o días.
* **Estructura:** Fomentan la participación activa en actividades como bosquejar, aportar ideas, diagramación de afinidad o creación de artefactos para organizar y capturar el progreso del grupo.
* **Preparación:** La preparación es algo tardada debido a la longitud, materiales, herramientas requeridas, actividades que necesitan ser planificadas y la aceptación que debe lograrse antes de que se lleve a cabo el Workshop.

**4.1.10. Modelo COCOMO y su aplicación, sacar resultados del modelo**

**¿Qué es el modelo COCOMO?**

Constructive Cost Model es un modelo de regresión basado en **LOC**, es decir, **número de líneas de código**.

Es un modelo de estimación de costos de procedimiento para proyectos de software y a menudo, se usa como un proceso para predecir de manera confiable los diversos parámetros asociados con la realización de un proyecto, como **el tamaño, el esfuerzo, el costo, el tiempo y la calidad**.

**¿Parámetros clave que definen la calidad de un software?**

* **Esfuerzo:** Cantidad de trabajo que se requerirá para completar una tarea. Se mide en unidades meses-persona.
* **Programación:** Simplemente significa la cantidad de tiempo necesario para completar el trabajo, que es, por supuesto, proporcional al esfuerzo realizado. Se mide en unidades de tiempo como semanas, meses.

**Tipos de proyectos**

* **Orgánico:** Se dice que un proyecto de software es de tipo orgánico si el tamaño del equipo requerido es suficientemente pequeño, el problema se comprende bien y se ha resuelto en el pasado y también los miembros del equipo tienen una experiencia nominal con respecto al problema.
* **Adosado:** Se dice que un proyecto de software es del tipo Adosado si las características vitales como el tamaño del equipo, la experiencia y el conocimiento de los diversos entornos de programación se encuentran entre las de orgánico e integrado. Los proyectos clasificados como Pareados son comparativamente menos familiares y difíciles de desarrollar en comparación con los orgánicos y requieren más experiencia y mejor orientación y creatividad. Por ejemplo: los compiladores o diferentes sistemas integrados se pueden considerar del tipo Semi-Independiente.
* **Integrado:** Un proyecto de software que requiere el nivel más alto de complejidad, creatividad y experiencia se incluye en esta categoría. Dicho software requiere un equipo de mayor tamaño que los otros dos modelos y también los desarrolladores deben tener suficiente experiencia y creatividad para desarrollar modelos tan complejos.

**Tipos de modelos**

* **Básico:** Se puede utilizar para cálculos rápidos y ligeramente aproximados de los costos de software. Su precisión está algo restringida debido a la ausencia de suficientes consideraciones de factores.
* **Intermedio:** Tiene en cuenta estos factores de costo
* **Detallado:** También tiene en cuenta la influencia de las fases individuales del proyecto, es decir, tiene en cuenta ambos factores de costo y también se realizan cálculos por fases para obtener un resultado más preciso.

**Estimación del esfuerzo**

Las siguientes fórmulas se utilizan para todos los tipos de modelos.







Las contantes a, b, c y d cambian dependiendo del modelo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Proyectos de software** | **a** | **B** | **C** | **D** |
| Orgánico | 2.4 | 1.05 | 2.5 | 0,38 |
| Semi separado | 3,0 | 1.12 | 2.5 | 0,35 |
| Incorporado | 3.6 | 1,20 | 2.5 | 0,32 |

* **Modelo Básico**
* **Intermedio**
* **Detallado**

**4.1.11. Análisis de solicitud y modificaciones**

**4.2. Calidad de software**

**4.2.1. ISO 9001**

**¿Qué es y para qué sirve esta norma?**

Un sistema de gestión de calidad ISO 9001 le ayudará a **supervisar y gestionar continuamente la calidad en toda su empresa para que pueda identificar áreas de mejora**. A nivel internacional, es el sistema de calidad de elección.

**4.2.2. ISO 45001**

**¿Qué es y para qué sirve esta norma?**

Es la norma internacional para **sistemas de gestión de seguridad y salud en el trabajo, destinada a proteger a los trabajadores y visitantes de accidentes y enfermedades laborales**.

**4.2.3. ISO 25000**

**¿Qué es y para qué sirve esta norma?**

Constituye una serie de normas basadas en ISO/IEC 9126 y en ISO/IEC 14598 cuyo objetivo principal es **guiar el desarrollo de los productos de software** mediante la especificación de requisitos y evaluación de características de **calidad**.

**4.2.4 Pruebas de software**

* **Pruebas de caja blanca**
  + **Pruebas estructurales**
    - Valida que los flujos internos de la aplicación cumplen con la planificación establecida en el código de la aplicación.
    - El objetivo es asegurar que la aplicación cumple con el flujo establecido para valores inesperados de entrada y no hay anomalías.
    - Permite identificar que parte del código no ha sido probado (pruebas de cobertura), que flujos nunca van a ser recorridos (condiciones) o la accesibilidad entre distintos componentes de la aplicación.
* **Pruebas de caja negra**
  + **Pruebas funcionales**
    - Validar la funcionalidad del producto o componente.
    - El diseño de estas pruebas se realiza en base a los requisitos y casos de uso.
    - Pueden ser bloqueadas para todos los niveles de pruebas anteriormente.
  + **Pruebas no funcionales**
    - Validan el comportamiento del sistema, especialmente la infraestructura.
    - Pueden ser consideradas para todos los niveles.

**4.2.5 Atributos de calidad**

Atributos de calidad validados por la norma ISO 25010



**Compatibilidad:** Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software.

Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* **Interoperabilidad:** Facilidad de comunicación con componentes externos.
* **Coexistencia:** Estar en un contexto con otros sistemas.

**Usabilidad:** Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones.

Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* **Reconocimiento de idoneidad:** Capacidad del producto que permite al usuario entender si el software es adecuado para sus necesidades.
* **Capacidad de aprendizaje:** Capacidad del producto que permite al usuario aprender su aplicación.
* **Capacidad para ser usado:** Capacidad del producto que permite al usuario operarlo y controlarlo con facilidad.
* **Protección contra errores de usuario:** Capacidad del sistema para proteger a los usuarios de hacer errores y de darle FeedBack sobre los mismos.
* **Estética de la interfaz de usuario:** Capacidad de la interfaz de usuario de agradar y satisfacer la interacción con el usuario.
* **Accesibilidad:** Capacidad del producto que permite que sea utilizado por usuarios con determinadas características y discapacidades.

**Fiabilidad / Confiabilidad:** Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados.

Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* **Madurez:** Capacidad del sistema para satisfacer las necesidades de fiabilidad en condiciones normales.
* **Disponibilidad:** Capacidad del sistema o componente de estar operativo y accesible para su uso cuando se requiere.
* **Tolerancia a fallos:** Capacidad del sistema o componente para operar según lo previsto en presencia de fallos hardware o software.
* **Capacidad de recuperación:** Capacidad del producto software para recuperar los datos directamente afectados y reestablecer el estado deseado del sistema en caso de interrupción o fallo.

**Adecuación Funcional:** Representa la capacidad del producto software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificadas.

Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* **Completitud funcional**: Grado en el cual el conjunto de funcionalidades cubre todas las tareas y los objetivos del usuario especificados.
* **Corrección funcional**: Capacidad del producto o sistema para proveer resultados correctos con el nivel de precisión requerido.
* **Pertinencia funcional**: Capacidad del producto software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados.

**Eficiencia de desempeño:** Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones.

Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* **Comportamiento temporal**: Los tiempos de respuesta y procesamiento y los ratios de throughput de un sistema cuando lleva a cabo sus funciones bajo condiciones determinadas en relación con un banco de pruebas (benchmark) establecido.
* **Utilización de recursos**: Las cantidades y tipos de recursos utilizados cuando el software lleva a cabo su función bajo condiciones determinadas.
* **Capacidad**: Grado en que los límites máximos de un parámetro de un producto o sistema software cumplen con los requisitos.

**Seguridad:** Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos.

Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* **Confidencialidad**: Capacidad de protección contra el acceso de datos e información no autorizados, ya sea accidental o deliberadamente.
  + **Integridad**: Capacidad del sistema o componente para prevenir accesos o modificaciones no autorizados a datos o programas de ordenador.
  + **No repudio**: Capacidad de demostrar las acciones o eventos que han tenido lugar, de manera que dichas acciones o eventos no puedan ser repudiados posteriormente.
  + **Responsabilidad**: Capacidad de rastrear de forma inequívoca las acciones de una entidad.
  + **Autenticidad**: Capacidad de demostrar la identidad de un sujeto o un recurso.

**Mantenibilidad:** Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas.

Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* **Modularidad**: Capacidad de un sistema o programa de ordenador (compuesto de componentes discretos) que permite que un cambio en un componente tenga un impacto mínimo en los demás.
* **Reusabilidad**: Capacidad de un activo que permite que sea utilizado en más de un sistema software o en la construcción de otros activos.
* **Analizabilidad**: Facilidad con la que se puede evaluar el impacto de un determinado cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software, o identificar las partes a modificar.
* **Capacidad para ser modificado**: Capacidad del producto que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.
* **Capacidad para ser probado**: Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.

**Portabilidad:** Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, software, operacional o de utilización a otro.

Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

* **Adaptabilidad**: Capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados de hardware, software, operacionales o de uso.
  + **Capacidad para ser instalado**: Facilidad con la que el producto se puede instalar y/o desinstalar de forma exitosa en un determinado entorno.
  + **Capacidad para ser reemplazado**: Capacidad del producto para ser utilizado en lugar de otro producto software determinado con el mismo propósito y en el mismo entorno.

**4.2.6 Metodología ADD**

**¿Qué es ADD?**

Attribute Driven Design (ADD) es un **método para sistematizar la definición de la arquitectura de software**.

La estrategia utilizada por ADD es **descomponer a un sistema en varios elementos** y **aplicar tácticas arquitectónicas** en busca de **satisfacer el cumplimiento de los atributos de calidad**. En cada nivel de descomposición son validados los requerimientos de calidad y los requerimientos funcionales.

**Elementos de entrada para ADD**

* Atributos de calidad
* Los requerimientos funcionales
* Las decisiones de diseño definidos en la etapa de requerimientos de un proyecto y priorizados por los stakeholders

Antes de iniciar la implementación del método es importante que los atributos de calidad hayan sido priorizados por los stakeholders de acuerdo a las metas de negocio y la misión del cliente. Los requerimientos de calidad deben estar expresados en escenarios de calidad específicos.

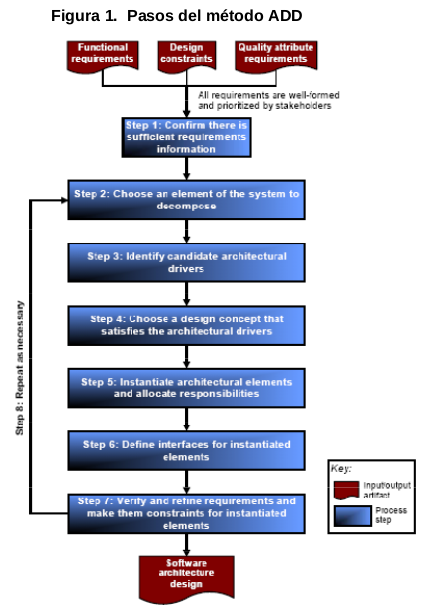
**Resultados de aplicar ADD**

* Conjunto de decisiones de diseño estructuradas.
* Interconexión y coordinación de mecanismos.
* Aplicación de patrones y tácticas arquitectónicas para especificar partes de la arquitectura.
* Requerimientos de atributos de calidad cubiertos.
* No se obtienen interfaces detalladas.

Una vez aplicado el método se obtiene un **diseño o arquitectura inicial de software** la cual contiene módulos, componentes y conectores que ilustran la relación entre elementos.

**Pasos de ADD**

1. Confirmar que existe suficiente información de requerimientos.
2. Elegir un elemento del sistema para descomponerlo.
3. Identificar drivers arquitectónicos desde el conjunto de escenarios de calidad y requerimientos funcionales.
4. Elegir atributos primitivos que satisfagan las características arquitectónicas.
5. Instanciar los elementos arquitectónicos y asignar responsabilidades.
6. Definir interfaces para elementos instanciados.
7. Verificar y refinar los requerimientos (casos de uso y escenarios de calidad) y aplicar las restricciones a los elementos instanciados.
8. Repetir del paso 2 al 7 para los siguientes elementos del sistema que se desee descomponer.



**4.2.7. Principios de seguridad**

1. Partir siempre de un **modelo de permisos mínimos**, es mejor ir escalando privilegios por demanda de acuerdo a los perfiles establecidos en las etapas de diseño.
2. Si se utiliza un lenguaje que no sea compilado, asegurarse de **limpiar el código que se pone en producción**, para que no contenga rutinas de pruebas, comentarios o cualquier tipo de mecanismo que pueda dar lugar a un acceso indebido.
3. **Nunca confiar en los datos que ingresan a la aplicación**, todo debe ser verificado para garantizar que lo que está ingresando a los sistemas es lo esperado y además evitar inyecciones de código.
4. **Hacer un seguimiento de las tecnologías utilizadas para el desarrollo**. Estas van evolucionando y cualquier mejora que se haga puede dejar obsoleta o inseguras versiones anteriores.
5. **Todos los accesos que se hagan a los sistemas deben ser validados**.
6. Para intercambiar información sensible **utilizar protocolos para cifrar las comunicaciones**, y en el caso de almacenamiento **la información confidencial debería estar cifrada utilizando algoritmos fuertes y claves robustas**.
7. **Cualquier funcionalidad**, campo, botón o menú nuevo **debe agregarse de acuerdo a los requerimientos de diseño**. De esta forma se evita tener porciones de código que resultan siendo innecesarias.
8. **La información almacenada en dispositivos móviles debería ser la mínima**, y más si se trata de contraseñas o datos de sesión. Este tipo de dispositivos son los más propensos a ser que se pierdan y por lo tanto su información puede ser expuestas más fácilmente.
9. **Cualquier cambio que se haga debería quedar documentado**, esto facilitará modificaciones futuras.
10. **Poner más cuidado en los puntos más vulnerables**, no hay que olvidar que el nivel máximo de seguridad viene dado por el punto más débil.

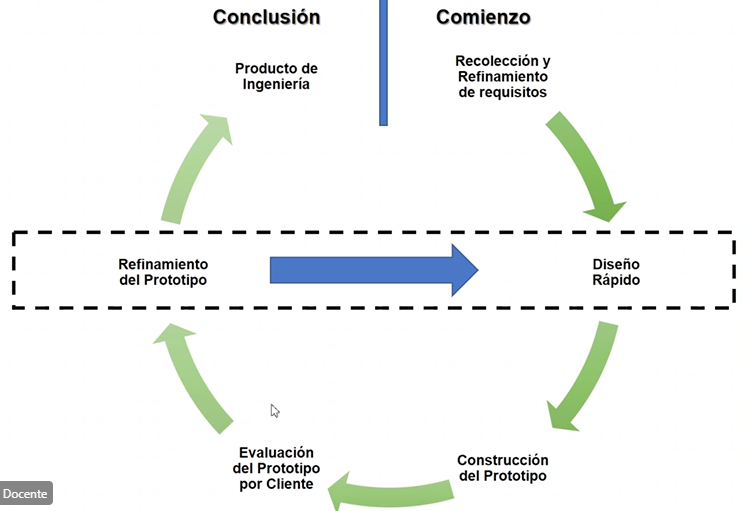
**4.3. Metodologías de desarrollo**

**4.3.1. Metodología Cascada**

**Puntos clave**

* Ideal para proyectos pequeños.
* No se pueden ver los resultados hasta que el proyecto ya está bastante avanzado.

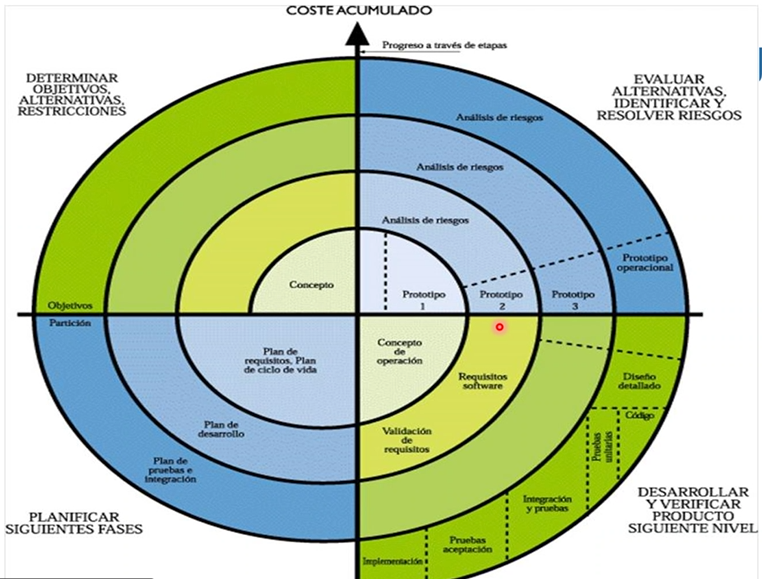
**4.3.2. Metodología por prototipos**



**Puntos clave**

* Usado para proyectos medianos donde se busca algo visible de inmediato
* No incluye evaluación de riesgos.
* Desarrolla rápido un prototipo para que los usuarios lo prueben y den un feedback.
* Arregla errores y agrega nuevos requerimientos en cada iteración después del feedback
* Modelo iterativo basado en prueba y error.

**4.3.3. Metodología espiral**



**Puntos clave**

* Se usa para proyectos complejos con larga duración como los ERP.
* Desarrolla por prototipos incluyendo un análisis de riesgos.
* Combinación entre cascada y prototipos
* Se divide en cuatro etapas:
  + planificación
  + análisis de riesgo
  + desarrollo de prototipo
  + evaluación del cliente

**4.3.4. Metodología incremental**

**Puntos clave**

* Se construye el software de manera progresiva
* En cada etapa incremental se agrega una nueva funcionalidad
* Permite ver resultados más rápido que el modelo en cascada
* Es más flexible que los modelos cascada, prototipos y espiral

**4.3.5. Metodología RAD**

**¿Qué es RAD o Rapid Application Development?**

Es una metodología con base en un modelo de cascada que se centra en desarrollar aplicaciones rápidamente por medio de iteraciones frecuentes y aprobaciones con comentarios continuos de los clientes.

Al priorizar los lanzamientos de **prototipos ágiles y rápidos**, RAD incide en la usabilidad del software, los **comentarios de los usuarios** y la entrega rápida a través de una planificación a largo plazo y **un único conjunto de requisitos iniciales** para la creación de elementos, como las aplicaciones personalizadas (aplicaciones hechas a la medida específicamente para las necesidades de un cliente).

**Objetivo**

Desarrollar **prototipos de la aplicación de forma veloz interactuando con el usuario** formando prototipos cada vez más adecuados a la funcionalidad que se requiere.

Para ello utiliza principalmente **software de generación automática de código**, requiriendo **desarrolladores especializados** en el software en cuestión para evitar en mayor medida posible los problemas de código que estos programas generan, haciendo el mantenimiento y pulido de la aplicación o sistema más laborioso, con la ventaja de tener el software usable en menor tiempo.

**Principios básicos de RAD**

* La participación activa de los usuarios es imprescindible.
* Iterativamente realiza la producción de software, en lugar de enfocarse en un prototipo.
* Produce la documentación necesaria para facilitar el futuro desarrollo y mantenimiento.

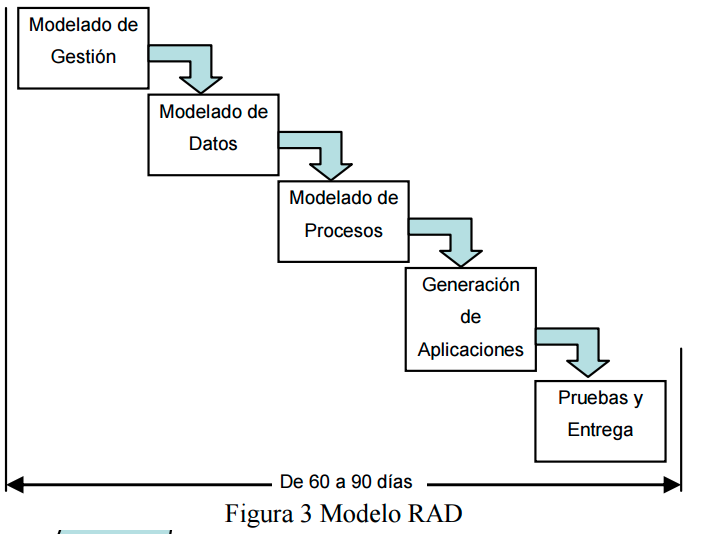
El método comprende el desarrollo ITERATIVO, la construcción de prototipos y el uso de utilidades de tipo CASE.

**Características de RAD**

* **Equipos híbridos:** Personal para elaborar el software como analistas, diseñadores, programadores, etc.
* **Herramientas especializadas:** Herramientas especializadas para el desarrollo de software como IDEs, lenguajes de programación, etc.
* **Timeboxing**
  + Gestión del tiempo fijo que se asigna al realizar el software.
  + Algunas funciones secundarias se eliminan para cumplir con el tiempo de entrega.

**Fases de RAD**

* **Modelado de gestión:** Recopilación de información del proceso de negocio. Respondiendo las siguientes preguntas:
  + ¿Qué información conduce al proceso de negocio?
  + ¿Quién la genera?
  + ¿A dónde va la información?
  + ¿Quién la procesa?
* **Modelado de datos: S**e definen los almacenes de datos y cómo se relacionan los almacenes entre sí.
* **Modelado de proceso:** Se utiliza para añadir, modificar, suprimir o recuperar un objeto de datos.
* **Generación de aplicaciones:** Para esto se utiliza una herramienta de cuarta o quinta generación que permite crear el software y facilitar la construcción del programa (IDEs, lenguajes de programación, etc.).
* **Pruebas y entrega:** El proceso de desarrollo finaliza realizando pruebas de calidad del software diseñado con la herramienta RAD, posteriormente se realiza la implementación de la aplicación.



**Ventajas de RAD**

1. Comprar puede ahorrar dinero en comparación con construir.
2. Los entregables pueden ser fácilmente trasladados a otra plataforma.
3. El desarrollo se realiza a un nivel de abstracción mayor.
4. Visibilidad temprana.
5. Mayor flexibilidad.
6. Menor codificación manual.
7. Mayor involucramiento de los usuarios.
8. Posiblemente menos fallas.
9. Posiblemente menor costo.
10. Ciclos de desarrollo más pequeños.
11. Interfaz gráfica estándar.

**Desventajas de RAD**

1. Comprar puede ser más caro que construir.
2. Costo de herramientas integradas y equipo necesario.
3. Progreso más difícil de medir.
4. Menos eficiente.
5. Menor precisión científica.
6. Riesgo de revertirse a las prácticas sin control de antaño.
7. Más fallas (por síndrome de “codificar a lo bestia”).
8. Prototipos pueden no escalar, un problema mayúsculo.
9. Funciones reducidas (por “timeboxing”).
10. Dependencia en componentes de terceros: funcionalidad de más o de menos, problemas legales.

**Puntos clave**

* Desarrollar software de alta calidad en un corto periodo de tiempo
* Los costes son mucho más altos
* El desarrollo es más flexible
* Requiere una mayor intervención de los usuarios
* El código puede contener más errores
* Las funciones son limitadas debido al poco tiempo del que se dispone para desarrollarlas
* El objetivo es iterar el menor número posible de veces para conseguir una aplicación completa de forma rápida.

**4.3.6. Metodología RUP**

**¿Qué es RUP o Proceso Unificado Relacional?**

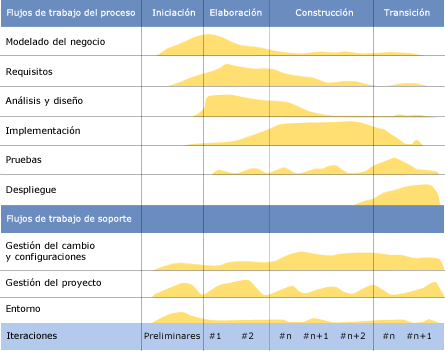
Es un proceso de desarrollo de software el cual utiliza el lenguaje unificado de modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de **sistemas orientados a objetos.**

Este proceso se usa preferiblemente para **proyectos complejos con equipos grandes**.

**Características sistemas orientados a objetos**

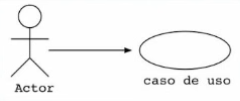
* Forma diciplinada de asignar tareas y responsabilidades (quien hace qué, cuándo y cómo)
* Pretende implementar las mejores prácticas en Ingeniería de Software.
* **Desarrollo iterativo**
* Administración de requisitos
* **Uso de arquitectura basada en componentes**
* Control de cambios
* Modelado visual del software
* Verificación de la calidad del software

**Fases del ciclo de vida de RUP**



n son las iteraciones y el número varía dependiendo el avance obtenido en cada iteración.

* **Inception:** Inicio
  + Definir y acordar el alcance del proyecto con los patrocinadores.
  + Identificar los riesgos asociados al proyecto
  + Proponer una visión muy general de la arquitectura de software
  + Producir el plan de las fases y el de iteraciones posteriores
* **Elaboration:** Elaboración
  + Se seleccionan los casos de uso que permiten definir la arquitectura base del sistema y se desarrollaran en esta fase.



* + Se realiza la especificación de los casos de uso seleccionados y el primer análisis de dominio del problema, se diseña la solución preliminar.
* **Construction:** Desarrollo
  + Completar la funcionalidad del sistema, para ello se deben clarificar los requerimientos pendientes.
  + Administrar los cambios de acuerdo a las evaluaciones realizadas por los usuarios.
  + Se realizan las mejoras para el proyecto.
* **Transition:** Cierre
  + Asegurar que el software esté disponible para los usuarios finales.
  + Ajustar los errores y defectos encontrados en las pruebas de aceptación.
  + Capacitar a los usuarios y proveer el soporte técnico necesario.

**Elementos de RUP**

* **Actividades:** Son los procesos que se han de realizar en cada etapa/iteración
* **Trabajadores:** Personas involucradas en cada actividad del proyecto.
* **Artefactos:** Herramientas empleadas para el desarrollo del proyecto. Puede ser un documento, un modelo, un elemento del modelo.
  + Inicio
    - Documento visión
    - Especificación de requerimientos
  + Elaboración
    - Diagramas de caso de uso
  + Construcción

Documento Arquitectura que trabaja con las siguientes vistas.

* + - Vista lógica
      * Diagrama de clases
      * Modelo entidad relación
    - Vista de implementación
      * Diagrama de secuencia
      * Diagrama de estados
      * Diagrama de colaboración
    - Vista conceptual
      * Modelo de dominio

**Mejores prácticas de RUP**

* Desarrollar el software de forma iterativa
* Gestionar requisitos
* Utilice arquitecturas basadas en componentes
* Modelar visualmente el software
* Verifique la calidad del software
* Controlar los cambios de software

**4.3.7. Metodología SCRUM**

**Puntos clave**

* Basada en la metodología incremental
* Divide los requisitos y tareas de forma similar a Kanban
* Se itera sobre bloques de tiempos cortos y fijos (entre dos y cuatro semanas) para conseguir un resultado completo en cada iteración
* Las etapas son:
  + planificación de la iteración (planning sprint)
  + ejecución (sprint)
  + reunión diaria (daily meeting)
  + demostración de resultados (sprint review)
* Cada iteración por estas etapas se denomina también sprint.

**4.3.8. Metodología Lean**

**Puntos clave**

* Ideal para que equipos pequeños y muy capaces elaboren las tareas en poco tiempo
* Los activos más importantes son las personas y su compromiso
* El aprendizaje, las reacciones rápidas y potenciar el equipo son fundamentales
* El tiempo y los costes pasan a segundo plano

**4.3.9. Metodología XP**

**Puntos clave**

* Basada en las relaciones interpersonales, que se consideran la clave del éxito
* Su principal objetivo es crear un buen ambiente de trabajo en equipo y que haya un feedback constante del cliente
* El trabajo se basa en 12 conceptos:
  + Diseño sencillo
  + Testing
  + Refactorización
  + Codificación con estándares
  + Propiedad colectiva del código
  + Programación en parejas
  + Integración continua
  + Entregas semanales
  + Integridad con el cliente
  + Cliente in situ
  + Entregas frecuentes
  + Planificación