CICLOS DE VIDA

Índice de contenido

1Definición del proceso software y ciclo de vida....2
2Parámetros que se deben abordar para decidir el ciclo de vida....3
3Modelos de ciclo de vida....4
3.1Cascada....4
3.2Incremental....5
3.3Espiral....6
3.4Prototipado desechable....8
3.5Prototipado evolutivo....8
3.6Modelo genérico para el desarrollo orientado a objetos....10
3.7Metodologías ágiles....10
4Ejercicios....10

1 Definición del proceso software y ciclo de vida

Un proceso software se corresponde con la colección de actividades o tareas que comienza con la identificación de una necesidad y concluye con la retirada del software que satisface esa necesidad. Las actividades deben estar interrelacionadas de forma que la entrada de una tarea sea la salida de la anterior. Las principales tareas que consta un proceso de software son las siguientes:

- **Obtención de requisitos software.** Incluye el análisis del problema y concluye con una especificación completa del comportamiento externo que debería tener el sistema a construir.
- **Diseñar.** El diseño del sistema debe realizarse a dos niveles: alto nivel o diseño preliminar, y bajo nivel o diseño detallado. En el diseño preliminar se descompone el sistema software en sus componentes principales, estos componentes se subdividen a su vez en componentes más pequeños. Este proceso iterativo continúa hasta un nivel adecuado en el que los componentes pueda ser tratados en el diseño de bajo nivel. Generalmente, estos módulos realizan una única función bien detallada y pueden venir descritos por su entrada, su salida y la función que realizan. En el diseño detallado se definen y documentan los algoritmos que llevarán a cabo la función a realizar para cada módulo.
- **Implementar.** Consiste en transformar los algoritmos definidos durante el diseño de bajo nivel en un lenguaje comprensible por la computadora. La codificación suele llevarse acabo en dos niveles: la conversión del algoritmo en un lenguaje de alto nivel; y la transformación del lenguaje de alto nivel en lenguaje máquina. Generalmente, el primer nivel es realizado por personas y el segundo nivel se realiza automáticamente por el compilador.
- **Realizar pruebas.** Se necesita un proceso de comprobación o pruebas para eliminar errores. Las pruebas pueden ser:
  - *pruebas unitarias*, comprueban cada módulo implementado en busca de errores.
  - *pruebas de integración*, interconectan conjuntos de módulos previamente probados, asegurándose de que el conjunto se comporta tan bien como lo hacían independientemente.
  - *pruebas de sistema*, pretenden asegurar que la totalidad del sistema software, se comporte de acuerdo con la especificación de requisitos inicial.
- **Instalación.** Tras las pruebas, el sistema software y su entorno hardware pasan a la fase operativa.
- **Mantenimiento.** El mantenimiento consiste en la detección continuada de errores y su reparación. La ampliación, por su parte, se corresponde con la adición al sistema de nuevas capacidades.

El ciclo de vida se define como la orientación que debe seguir se para obtener a partir de los requisitos del cliente un producto final. También puede definirse como la forma de recorrer las tareas definidas en el proceso software. Por tanto, un ciclo de vida debe definir:

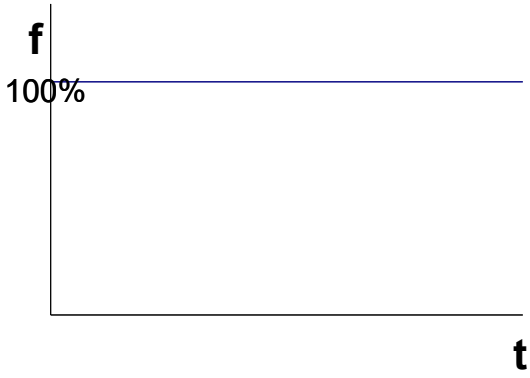
- El orden de las tareas del proceso software.
- Los criterios de transición para pasar de una fase a otra.

2 Parámetros que se deben abordar para decidir el ciclo de vida

Para poder seleccionar el ciclo de vida que más acorde a las necesidades del problema tenemos en consideración varios parámetros. Estos son: definición del problema, comprensión del problema, estabilidad de las necesidades y equipo de desarrollo.

- **Definición del problema.** Aquí se volara si el problema está bien definido, es decir, el usuario tiene claro las necesidades que tiene y que es lo que quiere como producto software (bien/mal).
- **Comprensión del problema.** Aquí se valora el grado de comprensión por parte del que realiza el proyecto. Por ejemplo, si conoce el negocio donde se va ubicar el nuevo producto a desarrollar, el manejo de cierto software requerido por el cliente, etc. (bien/mal).
- **Estabilidad de las necesidades (volatilidad).** Aquí se valorar si el cliente va variando la necesidad de sus demandas durante el tiempo (estable/variable).
- **Equipo de desarrollo.** En este parámetro se valoran dos subparámetros:
  i) **Experiencia en el dominio** (si/no).
  ii) **Experiencia en la técnicas de desarrollo** (si/no).

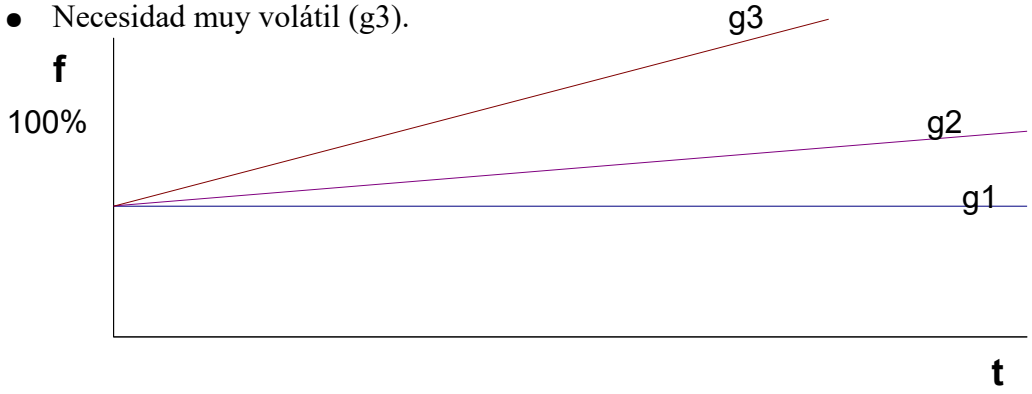
Para explicar las características de cada ciclo podemos tener en cuenta la siguiente gráfica:



La anterior gráfica nos representa que se cumplen al 100% las necesidades del cliente ante el producto software pedido.

Dentro de la gráfica funcionalidad/tiempo, también se puede representar la estabilidad de la necesidad, en la siguiente gráfica se puede ver tres forma de representar la estabilidad:

- Necesidad estable (g1).
- Necesidad poco volátil (g2).
- Necesidad muy volátil (g3).

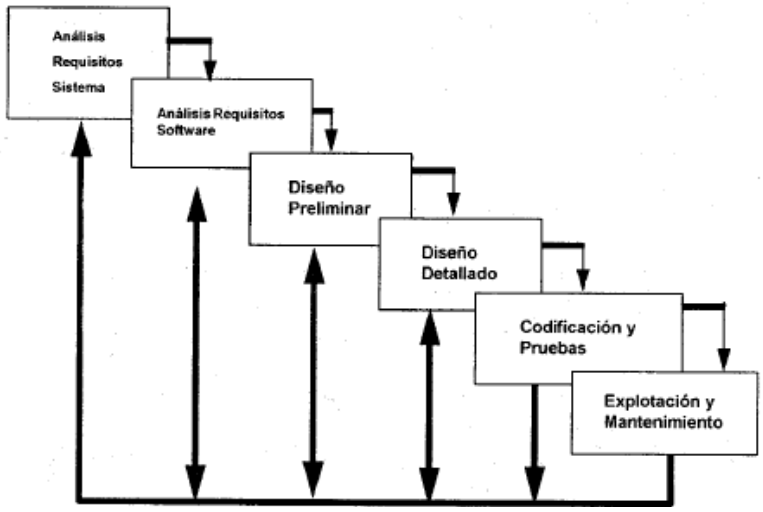


3 Modelos de ciclo de vida

3.1 Cascada

Características:

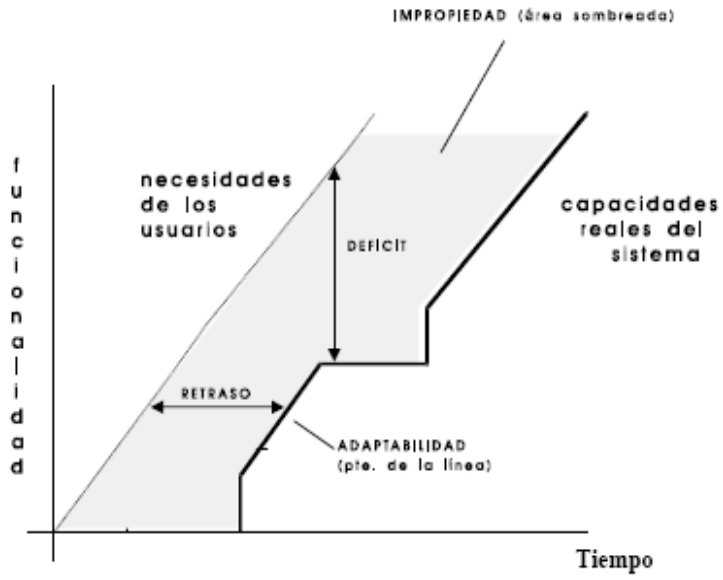
- Cada producto se produce en secuencia.
- Hasta que no acaba una fase no se empieza la siguiente.



Pueden encontrarse dos subtipos:

- tradicional. Se pasan por todas las fases del ciclo de forma secuencia.
- Mejora iterativa. Se pueden retroceder a una fase anterior o repetir la misma fase tantas veces como sea necesario hasta su refinamiento. Ésto podría ser útil cuando el usuario no tiene tanta experiencia.

Si reflejamos el ciclo de vida en la gráfica que hemos comentado anteriormente (funcionalidad/tiempo) quedaría de la siguiente forma:



Y en cuanto a los parámetros comentados en el apartado anterior, se podría utilizar este ciclo de vida cuando dichos parámetros tengan las siguientes valores:

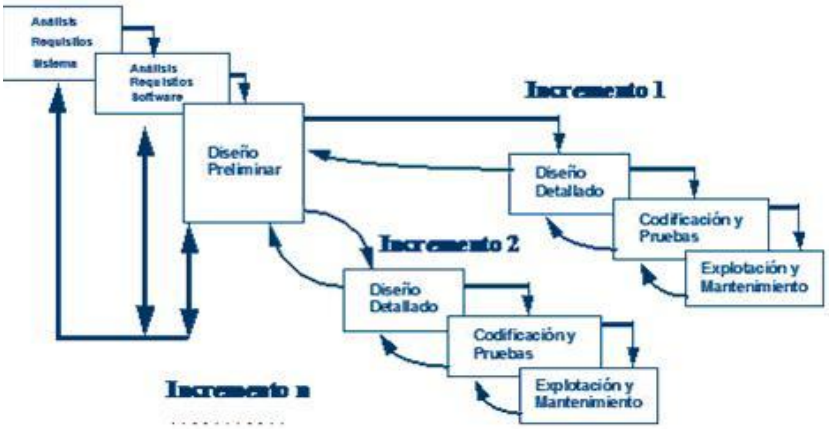
- **Definición del problema.** Bien definido.
- **Comprensión del problema.** Bien comprendido.
- **Estabilidad de las necesidades (volatilidad).** Nada o poco volatil.
- **Equipo de desarrollo.**
  i) **Experiencia en el dominio.** si
  ii) **Experiencia en la técnicas de desarrollo.** si

3.2 Incremental

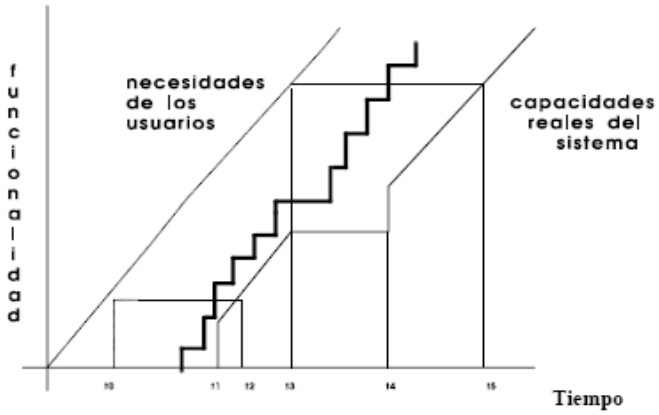
El ciclo de vida incremental es una repetición del ciclo de vida en cascada, aplicándose este ciclo en cada funcionalidad del programa a construir.

Alguno de los beneficios del ciclo de vida incremental son los siguientes:

- Construir un sistema pequeño tiene menos riesgo que construir todo el sistema.
- Al desarrollar independientemente las funcionales, es más fácil llegar a ver los requisitos que quiere el cliente.
- Si se detecta un error grave sólo se desecha la última iteración.
- No es necesario para el principio del proyecto conocer todos los requisitos, estos se podrán incluir en los siguientes incrementos.



Si reflejamos el ciclo de vida en la gráfica que hemos comentado anteriormente (funcionalidad/tiempo) quedaría de la siguiente forma:

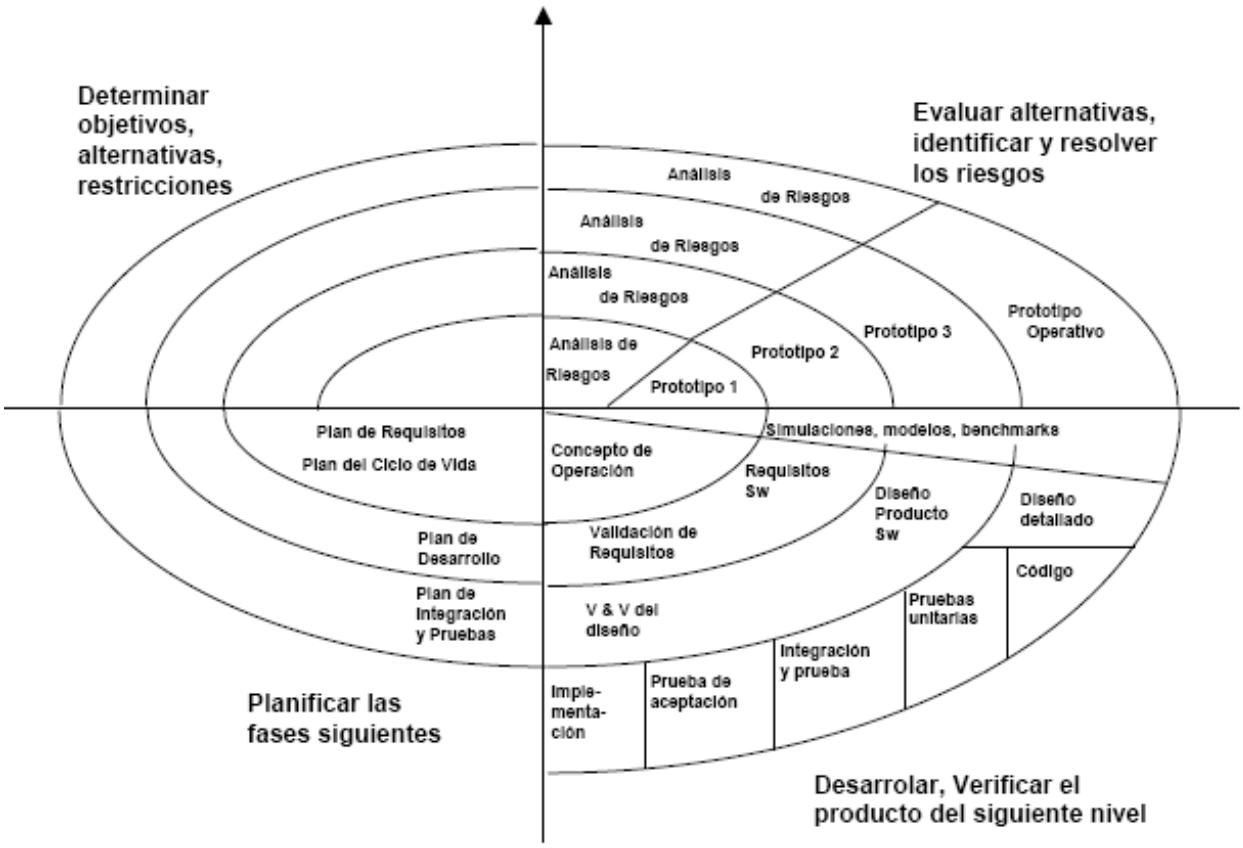


Y en cuanto a los parámetros comentados en el apartado anterior, se podría utilizar este ciclo de vida cuando dichos parámetros tengan las siguientes valores:

- **Definición del problema.** Mal o medio definido. Si puede hacer una fase de requisitos y sobre esa ya intentar que el cliente se defina mejor.
- **Comprensión del problema.** Mal o medio comprendido. Se hace lo que se ha entendido mostrando eso al cliente antes de hacer el siguiente incremento.
- **Estabilidad de las necesidades (volatilidad).** Variable, ya que permite el cambio de requisitos, es decir, se puede incrementar los requisitos.
- **Equipo de desarrollo.**
  i) **Experiencia en el dominio.** si
  ii) **Experiencia en la técnicas de desarrollo.** si

3.3 Espiral

Propuesto inicialmente por Boehm en 1988. Consiste en una serie de ciclos que se repiten. Cada uno tiene las mismas fases y cuando termina da un producto ampliado con respecto al ciclo anterior. En este sentido es parecido al modelo incremental, la diferencia importante es que tiene en cuenta el concepto de riesgo. Un riesgo puede ser muchas cosas: requisitos no comprendidos, mal diseño, errores en la implementación, etc. Un representación típica de esta estructura se muestra en la siguiente figura.



En cada iteración Boehm recomienda recopilar la siguiente lista de informaciones:

- **Objetivos**: Se hacen entrevistas a los clientes, se les hace rellenar cuestionarios, etc.
- **Alternativas**: Son las diferentes formas posibles de conseguir los objetivos. Se consideran desde dos puntos de vista

    - Características del producto.
    - Formas de gestionar el proyecto.
- **Restricciones**:
    - Desde el punto de vista del producto: Interfaces de tal o cual manera, rendimiento, etc.
    - Desde el punto de vista organizativo: Coste, tiempo, personal, etc.
- **Riesgos**: Lista de riesgos identificados.
- **Resolución de riesgos**: La técnica más usada es la construcción de prototipos.
- **Resultados**: Son lo que realmente ha ocurrido después de la resolución de riesgos.
- **Planes**: Lo que se va a hacer en la siguiente fase.
- **Compromiso**: Decisiones de gestión sobre como continuar.

Al terminar una iteración se comprueba que lo que se ha hecho efectivamente cumple con los requisitos establecidos, también se verifica que funciona correctamente. El propio cliente evalúa el producto. No existe una diferencia muy clara entre cuando termina el proyecto y cuando empieza la fase de mantenimiento. Cuando hay que hacer un cambio, este puede consistir en un nuevo ciclo.

Ventajas

- No necesita una definición completa de los requisitos para empezar a funcionar.
- Al entregar productos desde el final de la primera iteración es más fácil validar los requisitos.
- El riesgo en general es menor, porque si todo se hace mal, solo se ha perdido el tiempo y recursos invertidos en una iteración (las anteriores iteraciones están bien).
- El riesgo de sufrir retrasos es menor, ya que al identificar los problemas en etapas tempranas hay tiempo de subsanarlos.

Inconvenientes

- Es difícil evaluar los riesgos.
- Necesita de la participación continua por parte del cliente.
- Cuando se subcontrata hay que producir previamente una especificación completa de lo que se necesita, y esto lleva tiempo.

Donde es adecuado

- Sistemas de gran tamaño.
- Proyectos donde sea importante el factor riesgo.
- Cuando no sea posible definir al principio todos los requisitos.

Y en cuanto a los parámetros comentados en el apartado anterior, se podría utilizar este ciclo de vida cuando dichos parámetros tengan las siguientes valores:

- **Definición del problema.** Mal o medio definido. Si puede hacer una fase de requisitos y sobre esa ya intentar que el cliente se defina mejor.
- **Comprensión del problema.** Mal o medio comprendido. Se hace lo que se ha entendido mostrando eso al cliente antes de hacer el siguiente incremento.
- **Estabilidad de las necesidades (volatilidad).** Variable, ya que permite el cambio de requisitos, es decir, se puede incrementar los requisitos.
- **Equipo de desarrollo.**
    i) **Experiencia en el dominio.** Si
    ii) **Experiencia en la técnicas de desarrollo.** Si

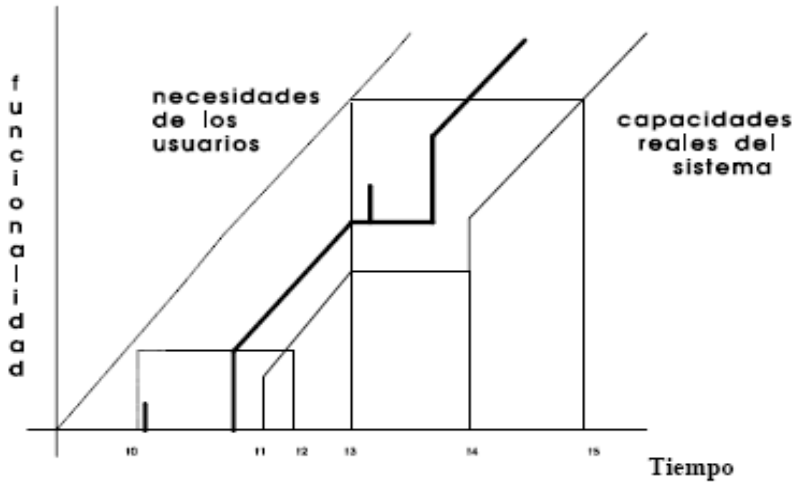
3.4 Prototipado desechable

Este modelo es básicamente prueba y error ya que si al usuario no le gusta una parte del prototipo significa que la prueba fallo por lo cual se debe corregir el error que se tenga hasta que el usuario quede satisfecho.

Además el prototipo debe ser construido en poco tiempo, usando los programas adecuados y no se debe utilizar mucho dinero pues a partir de que este sea aprobado nosotros podemos iniciar el verdadero desarrollo del software. Pero eso si al construir el prototipo nos asegura que nuestro software sea de mejor calidad, además de que su interfaz sea de agrado para el usuario. Un prototipo podrá ser construido solo si con el software es posible experimentar.

Sus desventajas son que debido a que el usuario ve que el prototipo funciona piensa que este es el producto terminado y no entienden que recién se va a desarrollar el software. Otro problema es que el prototipo deber ir acompañado de otro modelo pasa su desarrollo.

Si reflejamos el ciclo de vida en la gráfica que hemos comentado anteriormente (funcionalidad/tiempo) quedaría de la siguiente forma:



Y en cuanto a los parámetros comentados en el apartado anterior, se podría utilizar este ciclo de vida cuando dichos parámetros tengan las siguientes valores:

- **Definición del problema.** Mal o medio definido. Si puede hacer una fase de requisitos y sobre esa ya intentar que el cliente se defina mejor.
- **Comprensión del problema.** Mal o medio comprendido. Se hace lo que se ha entendido mostrando eso al cliente antes de hacer el siguiente incremento.
- **Estabilidad de las necesidades (volatilidad).** Nada o poco volátil.
  i) **Experiencia en el dominio.** si
  ii) **Experiencia en la técnicas de desarrollo.** Si

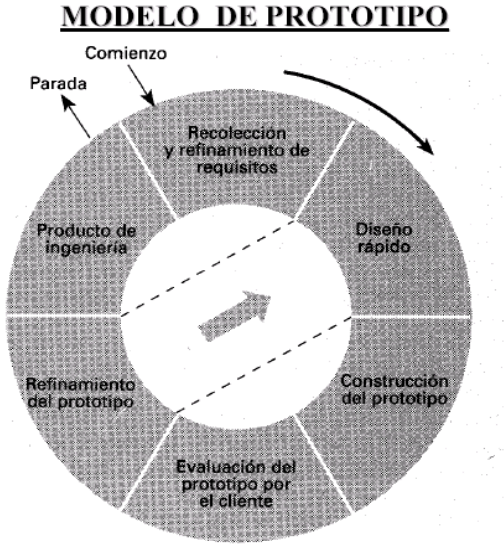
Dentro de los prototipos que se puede desarrollar se encuentran:

- maquetas o interfaz sin proceso.
- Interfaz con cierta funcionalidad.

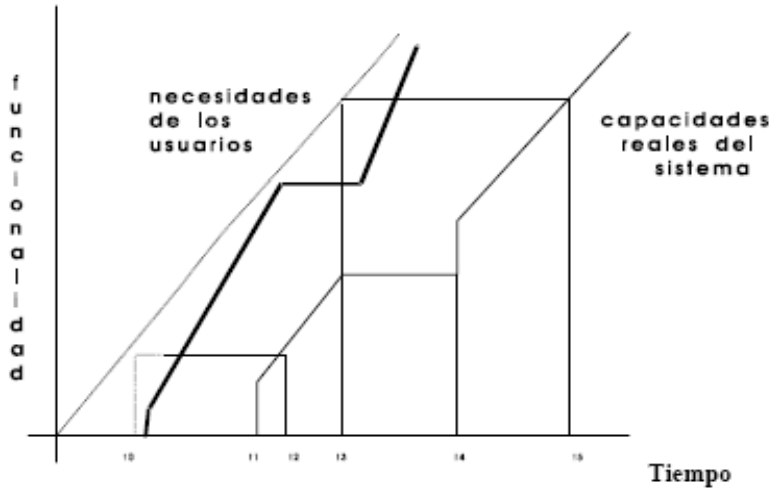
3.5 Prototipado evolutivo

Es un modelo parcialmente construido que puede pasar de ser prototipo a ser software pero no tiene una buena documentación y calidad. Sus características son:

- Construcción de una implementación parcial que cubre los requisitos conocidos, para ir aprendiendo el resto y, paulatinamente, incorporarlos al sistema.
- Reduce el riesgo y aumenta la probabilidad de éxito.
- No se conocen niveles apropiados de calidad y documentación.
- Problemas de gestión de configuración.



Si reflejamos el ciclo de vida en la gráfica que hemos comentado anteriormente (funcionalidad/tiempo) quedaría de la siguiente forma:



Y en cuanto a los parámetros comentados en el apartado anterior, se podría utilizar este ciclo de vida cuando dichos parámetros tengan las siguientes valores:

- **Definición del problema.** Mal definido.
- **Comprensión del problema.** Mal comprendido.
- **Estabilidad de las necesidades (volatilidad).** Volátil.
- **Equipo de desarrollo.**
  i) **Experiencia en el dominio.** no
  ii) **Experiencia en la técnicas de desarrollo.** no

3.6 Modelo genérico para el desarrollo orientado a objetos

En general la tecnología orientada a objetos propone seguir un ciclo de vida iterativo e incremental. Esto se establece las siguientes características:

- La tecnología orientada a objetos elimina las fronteras entre fases.
- El uso de la reutilización de bibliotecas de clases y otros componentes .

3.7 Metodologías ágiles

Son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno. En esencia, las empresas que apuestan por esta metodología consiguen gestionar sus proyectos de forma eficaz reduciendo los costes e incrementando su productividad. Ejemplos de estas metodología son SCRUM, KANBAN, DEVOPS,etc.

4 Ejercicios

Para cada enunciado determina que ciclo de vida es el más apropiado justificando la respuesta elegida:

i. Se pretende desarrollar una aplicación relativa a la gestión de pedidos de una empresa. En este caso el cliente no tiene una idea muy clara de que es lo que quiere. Además el personal informático va a utilizar una tecnología que le es completamente nueva.
ii. Se trata de un desarrollo informático de poca envergadura el cliente es informático y conoce en buena medida cuales son los requisitos de la aplicación que desea solicitar al equipo de desarrollo.
iii. En este caso de pretende desarrollar un sistema de información para una empresa de producción láctea. Por las características económicas y financieras de esta empresa, el desarrollo y la implantación del nuevo sistema de información supone asumir una serie de riesgos de índole técnico y económico.
iv. Se pretende desarrollar un sistema distribuido, sus requisitos no están claramente comprendidos y son propensos al cambio, cada uno de los componentes de este nuevo sistema se ejecutan en diferentes sucursales de la empresa, el sistema no estará disponible hasta que todos los componentes hayan sido implantados en la sucursal correspondiente.
v. En el sistema que se pretende desarrollar, el cliente sabe exactamente que debe hacer el sistema y comunica claramente los requisitos funcionales al equipo de desarrollo. Se sabe además que los requisitos son bastante estables, sin embargo el usuario no tiene muy clara la manera en que la nueva aplicación informática debería funcionar para satisfacer sus necesidades. Tampoco tiene claro que necesite una aplicación informática para solucionar su problemática.
vi. El desarrollo informático que se pretende emprender presenta ciertos problemas de liquidez económica. El cliente dispone de un presupuesto inicial para comenzar con el desarrollo del sistema que necesita y el resto del dinero que se estima necesario esta sujeto a la aprobación de diferentes créditos bancarios.