CICLOS DE VIDA

No existe un único Modelo de Ciclo de Vida que defina los estados por los que pasa cualquier producto software. Dado que existe una gran variedad de aplicaciones para las que se construyen productos software (software de tiempo real, de gestión, de ingeniería y científico, empotrado, de sistemas, de computadoras personales, etc.) y que dicha variedad supone situaciones totalmente distintas, es natural que existan diferentes Modelos de Ciclo de Vida. Por ejemplo, en aquellos casos en que el problema sea perfectamente conocido, el grupo de desarrollo tenga experiencia en sistemas del mismo tipo, el usuario sea capaz de describir claramente sus requisitos, un ciclo de vida tradicional, en cascada o secuencial sería el adecuado. Por el contrario, si el desarrollo conlleva riesgos (sean técnicos o de otro tipo), un ciclo de vida en espiral será el más apropiado. Sin embargo, si se está ante el caso en que es necesario probarle el producto al usuario para demostrarle la utilidad del mismo, se estará ante un ciclo de vida con prototipado, etc.

Un ciclo de vida debe:

- Determinar el orden de las fases del Proceso Software
- Establecer los criterios de transición para pasar de una fase a la siguiente

A continuación, se repasan los diferentes modelos de ciclo de vida existentes: cascada, gradual, espiral, prototipado operativo, prototipo de usar y tirar, etc. No existe un Modelo de Ciclo de Vida que sirva para cualquier proyecto, esto debe quedar claro. Cada proyecto debe seleccionar un ciclo de vida que sea el más adecuado para su caso. El ciclo de vida apropiado se elige en base a: la cultura de la corporación, el deseo de asumir riesgos, el área de aplicación, la volatilidad de los requisitos, y hasta qué punto se entienden bien dichos requisitos. El ciclo de vida elegido ayuda a relacionar las tareas que forman el proceso software de cada proyecto.

Ciertos modelos de ciclo de vida existen desde los primeros días de la Ingeniería de Software. El ciclo de vida del software clásico o modelo *en cascada* y el de *refinamiento sucesivo,* están ampliamente tratados en casi todos los libros sobre Ingeniería de Software. Los *estándares militares* también han marcado ciertas formas de ciclo de vida clásico en la práctica exigida para contratistas del Ministerio de Defensa de EE.UU. El modelo de *desarrollo incremental* está estrechamente relacionado con las prácticas industriales de producción de software donde aparece con mayor frecuencia. El *prototipado* es uno de los últimos ciclos de vida aparecidos que se han extendido tan rápidamente que, hoy en día, puede considerarse clásico. Esta rápida expansión del prototipado es debida al aumento en la complejidad de los sistemas software que se construyen, que hace necesario el desarrollo de prototipos antes de poder pasar a la construcción del sistema a escala real.

Dado que todos estos modelos se han usado durante algún tiempo es por lo que se consideran tradicionales. Es decir, poseen la solvencia suficiente y están lo suficientemente explicados como para ser usados rutinariamente. Los ciclos de vida que hemos llamado alternativos *(desarrollo con reutilización, síntesis automática de sistemas y desarrollo de sistemas basados en conocimientos),* resultan a menudo demasiado novedosos y poco contrastados como para aventurarse a usarlos en un desarrollo real. No obstante, en pocos años, seguramente muchos de ellos ya estarán suficientemente avalados como para ser usados.

1. APROXIMACIÓN TRADICIONAL

1. 1. Modelo de ciclo de vida en cascada

Este modelo fue presentado por primera vez por Royce en 1970. Se representa, frecuentemente, como un simple modelo con forma de cascada de las etapas del software, como muestra la Figura 4. En este modelo la evolución del producto software procede a través de una secuencia ordenada de transiciones de una fase a la siguiente según un orden lineal. Tales modelos semejan una máquina de estados finitos para la descripción de la evolución del producto software. El modelo en cascada ha sido útil para ayudar a estructurar y gestionar grandes proyectos de desarrollo de software dentro de las organizaciones.

Este modelo permite iteraciones durante el desarrollo, ya sea dentro de un mismo estado, ya sea de un estado hacia otro anterior, como muestran las flechas ascendentes de la Figura 4. La mayor iteración se produce cuando una vez terminado el desarrollo y cuando se ha visto el software producido, se decide comenzar de nuevo y redefinir los requisitos del usuario.

El uso del modelo en cascada:

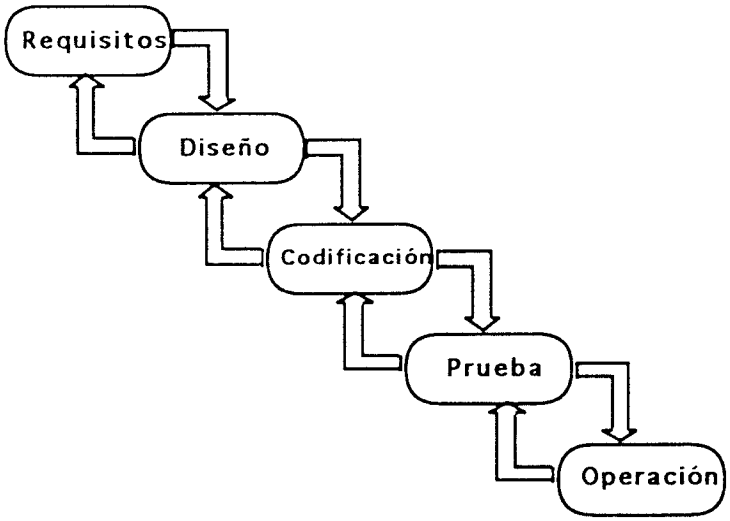

Figura 4. Modelo “en cascada’

- Obliga a especificar lo que el sistema debe hacer (o sea, definir los requisitos) antes de construir el sistema (esto es, diseñarlo).
- Obliga a definir cómo van a interactuar los componentes (o sea, diseñar) antes de construir tales componentes (o sea codificar).
- Permite al jefe del proyecto seguir y controlar los progresos de un modo más exacto. Esto le permite detectar y resolver las desviaciones sobre la planificación inicial.
- Requiere que el proceso de desarrollo genere una serie de documentos que posteriormente pueden utilizarse para la validación y el mantenimiento del sistema.

El modelo en cascada ha sido objeto de numerosas transformaciones y ha sido el más utilizado hasta ahora, aunque incorporando infinidad de variaciones que eliminan el carácter simplista del mismo. Aún hoy en día se asume que:

- Para que un proyecto tenga éxito, en cualquier caso, todos los estados señalados en el modelo en cascada deben ser desarrollados.
- Cualquier desarrollo en diferente orden de los estados dará un producto de inferior calidad. Sin embargo, que se siga este orden no significa (como se pensó durante mucho tiempo) que se deba seguir la filosofía del ciclo de vida en cascada: realizar cada fase de principio a fin y no pasar a la siguiente hasta haber acabado completamente con la anterior.

Sin embargo, a menudo, durante el desarrollo, se pueden tomar decisiones que den lugar a diferentes alternativas. El modelo en cascada no reconoce esta situación. Por ejemplo, dependiendo del análisis de requisitos se puede

implementar el sistema desde cero, o adoptar uno ya existente, o comprar un paquete que proporcione las funcionalidades requeridas. Es decir, resulta demasiado estricto para la flexibilidad que necesitan algunos desarrollos.

Entre otras limitaciones que se argumentan para este modelo se pueden señalar las siguientes:

- El modelo en cascada asume que los requisitos de un sistema pueden ser congelados antes de comenzar el diseño.

Este es el punto más débil de este modelo, puesto que esta asunción es sólo cierta para un pequeño número de problemas: aquellos no complejos, bien definidos y bien comprendidos. Para el resto de los sistemas, es decir para la mayoría, las necesidades del usuario están en constante evolución. Por tanto, el sistema que se construye persigue un objetivo que no es fijo, sino que varía. Esta es una de las principales razones para los retrasos en las entregas del software: El equipo intenta que el sistema cumpla requisitos nuevos que han surgido durante el desarrollo y para los cuales el sistema no estará diseñado. La movilidad del objetivo que se persigue es también una de las razones para no satisfacer las expectativas de los usuarios.

El equipo de desarrollo ha congelado y por tanto no ha reconocido los cambios inevitables que se han producido en los deseos del usuario durante el desarrollo. Cuando se entrega el sistema éste obviamente no satisface las expectativas actuales del cliente; en el mejor de los casos satisfará las expectativas que tenía tiempo atrás, cuando se comenzó el desarrollo.

La figura 5 muestra la constante evolución de las necesidades del usuario. Se han representado en un espacio de tiempo/funcionalidad: según pasa el tiempo, aumentan las expectativas de funcionalidades que el usuario espera que tenga el sistema. La evolución está simplificada, pues ni mucho menos es lineal ni continua, pero para hacerse una idea es suficiente.

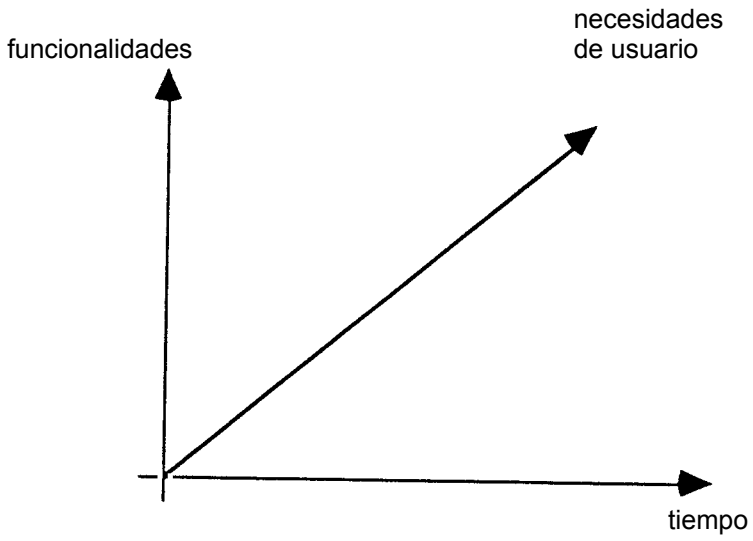

Figura 5. Evolución constante de las necesidades del usuario

La figura 6 lo que ocurre en los desarrollos que siguen el modelo en cascada: Cuando empieza el desarrollo, en t_0, el usuario tiene ciertas necesidades que seguramente el equipo de desarrollo no comprende bien. En t_1 el equipo ha producido un software operacional que no solamente no satisface las necesidades del usuario en el momento t_1, sino que ni siquiera alcanza a satisfacer las que tenía en t_0 el producto sufre entre t_1 y t_3 una serie de mejoras que ocasionalmente puede hacerle alcanzar las necesidades que el usuario tenía en t_0. En algún momento posterior a t_3, digamos t_4, el coste de mejora es tan alto que se opta por comenzar a construir un nuevo sistema (de nuevo basándose en requisitos no comprendidos en su totalidad); este nuevo desarrollo finaliza en t_5, y el ciclo comienza otra vez.

Además, hoy por hoy se asume que intentar describir a través de una descripción escrita de requisitos, lo que el usuario piensa, cree y siente sobre el futuro software, jamás puede ser tan efectivo como un prototipo donde el usuario vea de lo que se habla (y más especialmente para el interfase).

El fijar los requisitos al inicio del proyecto también suele traducirse en que el equipo de desarrollo emplea un gran esfuerzo en optimizar esa solución puntual. Eso lleva a que tal solución es difícil de modificar o aumentar, pues el haber congelado los requisitos llevó a no pensar en los posteriores cambios que siempre existen en el usuario, su entorno, sus procedimientos y su organización.

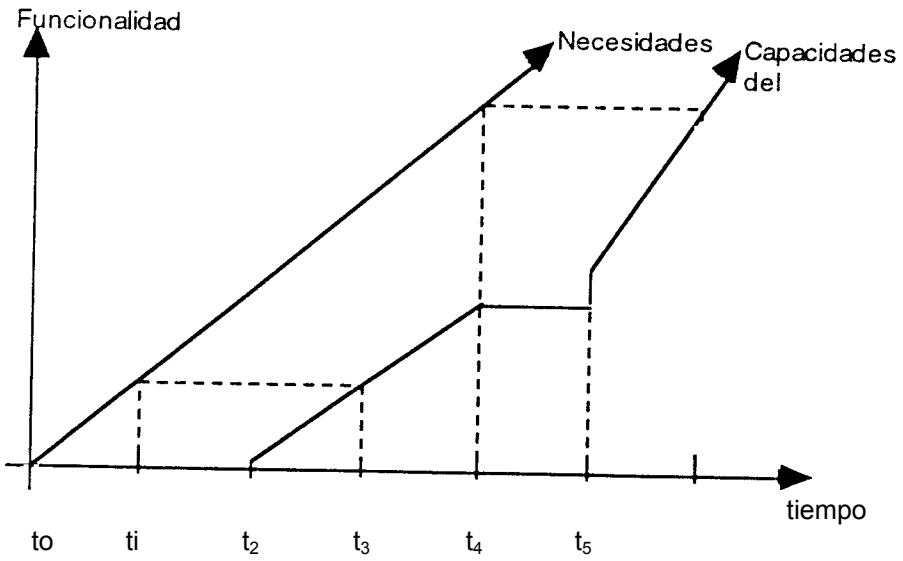

Figura 6. Los productos software no consiguen satisfacer todas las necesidades actuales del usuario

Envía al cliente el primer producto solamente después de que se han consumido el 99% de los recursos para el desarrollo. Esto significa que la mayor parte del "feedback" del cliente sobre sus necesidades se obtiene una vez que se han consumido los recursos. Compárese esta opción con la que se verá mas adelante en que: se envía al cliente un prototipo *chapucero* en las primeras fases del desarrollo; se obtienen los comentarios; se escribe una especificación de requisitos; y sólo entonces se acomete el desarrollo a su escala real. En este caso, sólo se han consumido el 20% de los recursos cuando el cliente ve el producto por primera vez.

Sin embargo, el ciclo de vida en cascada tiene tres propiedades muy positivas:

- Las etapas están organizadas de un modo lógico. Es decir, si una etapa no puede llevarse a cabo hasta que se hayan tomado ciertas decisiones de más alto nivel, debe esperar hasta que esas decisiones estén tomadas. Así, el diseño espera a los requisitos, el código espera a que el diseño esté terminado, etc.

- Cada etapa incluye cierto proceso de revisión, y se necesita una aceptación del producto antes de que la salida de la etapa pueda usarse. Este ciclo de vida está organizado de modo que se pase el menor número de errores de una etapa a la siguiente.

- El ciclo es iterativo. A pesar de que el flujo básico es de arriba hacia abajo, el ciclo de vida en cascada reconoce, como ya se ha comentado, que los problemas etapas inferiores afectan a las decisiones de las etapas superiores.

Existe una visión alternativa del modelo de ciclo de vida en cascada, mostrada en la Figura 7, que enfatiza en la validación de los productos, y de algún modo en el proceso de composición existente en la construcción de sistemas software.

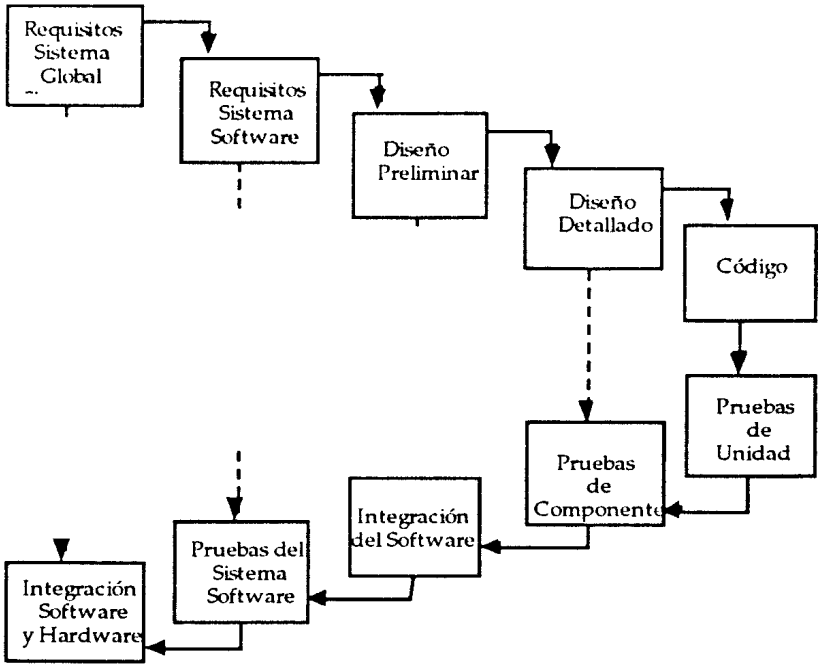

Figura 7. Visión alternativa del ciclo de vida en cascada

El proceso de análisis o descomposición subyacente en la línea superior del modelo de la Figura 7 consiste en los requisitos del sistema global se dividen en requisitos del hardware y requisitos del software. Estos últimos llevan al diseño preliminar de múltiples funciones, cada una de las cuales se expande en el diseño detallado, que, a su vez, evoluciona aún en un número mayor de programas unitarios. Sin embargo, el ensamblaje del producto final fluye justo en sentido contrario, dentro de un proceso de síntesis o composición. Primero se aceptan los programas unitarios probados. Entonces, éstos se agrupan en módulos que, a su vez, deben ser aceptados una vez probados. Los módulos se agrupan para certificar que el grupo formado por todos ellos incluyen todas las funcionalidades deseadas. Finalmente, el software es integrado con el hardware hasta formar un único sistema informático que satisface los requisitos globales.

La Figura 7 destaca que el producto obtenido en cada etapa no solo determina que debe hacerse en la fase siguiente del proceso, sino que también establece los criterios para determinar si el productos compuesto y ensamblado satisface los objetivos de la etapa. Esta idea está mostrada en la Figura 7 mediante las líneas punteadas. En otras palabras, el modelo de ciclo de vida de la Figura 7 está estructurado de tal modo que, en cada etapa, se define qué debe hacerse en el próximo paso de descomposición, pero también se documentan los criterios para determinar si el producto compuesto que resulta satisface las intenciones que se tenían hacia él.

Finalmente, en la Tabla 1 se muestran el porcentaje del costo que representa cada fase, sin tener en cuenta el mantenimiento. Evidentemente, el costo varía en función del tipo de software que se esté desarrollando. Las cifras que dio Boehm en 1975, todavía tienen validez, aunque el progreso que ha habido en los métodos de diseño hacen desplazar el costo hacia las fases de definición, disminuyéndolo en las fases de codificación y prueba.

Tipo de Sistema	Requisitos Diseño	Implementado	Prueba
Sistemas de Control	46	20	34
Sistemas Especiales	34	20	46
Sistemas Operativos	33	1 7	50
Sistemas Científicos	45-53	27-36	15-23
Sistemas de Gestión	44	28	28

Tabla 1. Costos de cada etapa

2. MODELO DE CICLO DE VIDA INCREMENTAL

El primero que habló de este nuevo modelo fue Hirsch en 1985. Se trata de una filosofía radicalmente distinta del modelo de cascada. El desarrollo incremental es el proceso de construir una implementación parcial del sistema global y posteriormente ir aumentando la funcionalidad del sistema. Esta aproximación reduce el gasto que se tiene antes de alcanzar cierta capacidad inicial. Además produce un sistema operacional más rápidamente; esto reduce la posibilidad de que las necesidades del usuario cambien durante el desarrollo.

Al seguir un tipo de desarrollo incremental, el software se construye de modo que deliberadamente sólo satisfaga unos pocos requisitos de todos los que tiene el usuario. Sin embargo, debe construirse de tal modo que facilite la incorporación de nuevos requisitos. Puede decirse, por tanto que el software así construido tiene una adaptabilidad mayor. Este tipo de aproximación tiene dos efectos:

1. Se reduce el tiempo de desarrollo inicial (hasta tener algo que funcione) debido al nivel reducido de funcionalidad.

2. Es más fácil de mejorar el software y, además, puede seguir mejorándose durante más tiempo.

La figura 8 muestra el modelo de ciclo de vida incremental en comparación con el modelo clásico en cascada. Nótese que: el tiempo de desarrollo inicial es menor que para el modelo en cascada; que ~ funcionalidad inicial (A) es menor que para la aproximación clásica (B); que la mayor adaptabilidad que posee el software así desarrollado, se indica por la mayor pendiente de la curva A-C con respecto al modelo en cascada (línea 8-O). El aspecto en forma de escalera indica una serie de construcciones del sistema (mejoras) discretas, planificadas y bien definidas.

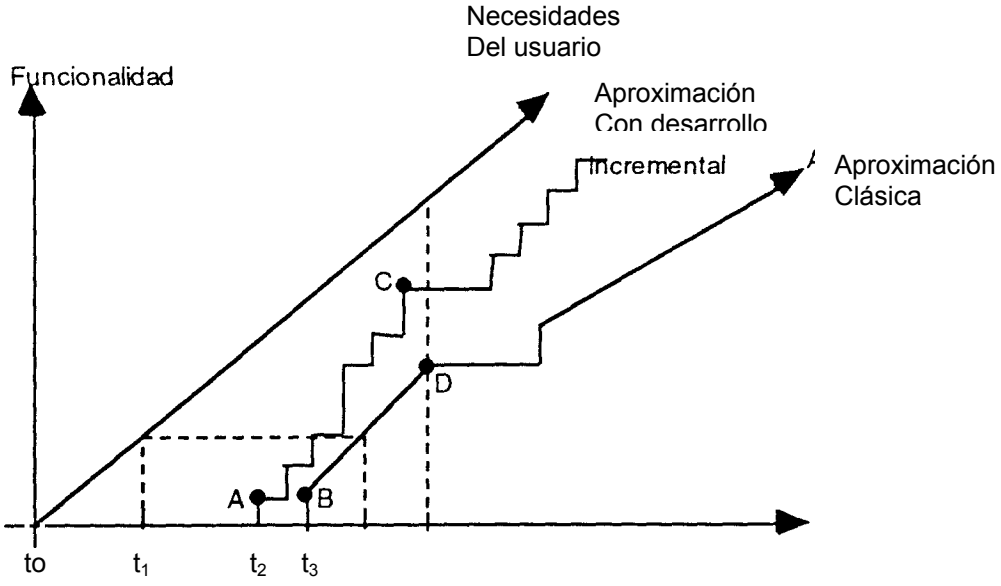

Figura 8.

En este modelo los puntos críticos del proyecto, o mojones a lo largo del camino de desarrollo, no son tanto los productos de cada etapa (especificaciones, diseño, código, etc.) sino las emisiones de incrementos en la capacidad del sistema. Los nuevos contenidos del sistema se determinan de la experiencia con las emisiones anteriores del sistema. Por tanto, el punto más crítico del sistema es la primera entrega: un software con capacidades suficientes como para servir para que el usuario se ejercite, para la evaluación y para evolucionar mejorando.

Los problemas que se le critican a este modelo de ciclo de vida son los siguientes:

- A menudo, la primera entrega se ha optimizado para que tenga éxito como demostración y como software de exploración. Por ejemplo, almacena toda la información en memoria principal para que el tiempo de respuesta sea adecuado. Posteriormente, cuando se quiere aumentar el sistema para ser usado a gran escala, las soluciones iniciales puntuales, y seguramente la arquitectura del sistema no servirán para versiones aumentadas.

- Las versiones iniciales no suelen prestar mucha atención a temas como: la seguridad, la tolerancia a fallos, el procesamiento distribuido, etc., centrándose en la funcionalidad del sistema y, quizás, en la interfase. El usuario puede pensar que el resto de las capacidades del sistema, que no se han tenido en cuenta en la primera versión, se entreguen tan rápidamente como las primeras. Esto puede meter en problemas al proyecto, puesto que la arquitectura inicial del sistema puede no ser válida cuando se consideran todas las capacidades que el sistema debe tener. En ese caso, alguna versión intermedia, la que trate la remodelación de sistemas, puede ser muy lenta de desarrollar.

- A veces, cuando se usa este modelo de ciclo de vida, los ingenieros olvidan realizar una buena etapa de análisis que les lleve a comprender la necesidad del usuario. Esta etapa de análisis debería conducirles a una primera versión que no se aleje mucho de los deseos del usuario. De este modo, el usuario practicará con la primera versión y ésta cumplirá el objetivo que tenía. El equipo de desarrollo a menudo, sacrifica la etapa de análisis en favor de una obtención rápida de la primera versión, pensando que en las siguientes versiones se irán acercando a los deseos del usuario, si en esta primera no se han entendido bien. Este comportamiento se traduce en una primera entrega tan alejada de la necesidad del usuario, que éste no se moleste en utilizar, avocando el proyecto completo al fracaso.

Por tanto, este modelo propone desarrollar sistemas produciendo en primer lugar las funciones esenciales de operación y, a continuación, proporcionar a los usuarios mejoras y versiones más capaces del sistema a intervalos regulares. Este modelo combina el ciclo de vida clásico del software con mejoras iterativas a nivel del desarrollo del sistema global. También proporciona una manera para distribuir periódicamente actualizaciones del mantenimiento de software comercial. Es, por lo tanto, un modelo popular de la evolución del software usado por firmas comerciales.

3. PROTOTIPADO

Suele ocurrir que, en los proyectos software, el cliente no tiene una idea muy detallada de lo que necesita, o que el ingeniero de software no está muy seguro de la viabilidad de la solución que tiene en mente. En estas condiciones, la mejor aproximación al problema es la realización de un prototipo.

El modelo de desarrollo basado en prototipos tiene como objetivo contrarrestar el problema ya comentado del modelo en cascada: la congelación de requisitos mal comprendidos. La idea básica es que el prototipo ayude a comprender los requisitos del usuario. El prototipo debe incorporar un subconjunto de la función requerida al software, de manera que se puedan apreciar mejor las características y posibles problemas.

La construcción del prototipo sigue el ciclo de vida estándar, sólo que su tiempo de desarrollo será bastante más reducido, y no será muy rigurosa la aplicación de los estándares.

El problema del prototipo es la elección de las funciones que se desean incorporar, y cuáles son las que hay que dejar fuera, pues se corre el riesgo de incorporar características secundarias, y dejar de lado alguna característica importante. Una vez creado el prototipo, se le enseña al cliente, para que “juegue” con él durante un período de tiempo, y a partir de la experiencia apodar nuevas ideas, detectar fallos etc.

Cuando se acaba la fase de análisis del prototipo, se refinan los requisitos del software, y a continuación se procede al comienzo del desarrollo a escala real. En realidad, el desarrollo principal se puede haber arrancado previamente, y avanzar en paralelo, esperando para un tirón definitivo ala revisión del prototipo.

El ciclo de vida clásico queda modificado de la siguiente manera por la introducción del uso de prototipos:

1. Análisis preliminar y especificación de requisitos
2. Diseño, desarrollo e implementación del prototipo
3. Prueba del prototipo
4. Refinamiento iterativo del prototipo
5. Refinamiento de las especificaciones de requisitos
6. Diseño e implementación del sistema final

Existen tres modelos derivados del uso de prototipos:

- **Maqueta**. Aporta al usuario ejemplo visual de entradas y salidas. La diferencia con el anterior es que en los prototipos desechables se utilizan datos reales, mientras que las maquetas son formatos encadenados de entrada y salida con datos simples estáticos.

- **Prototipo desechable**. Se usa para ayudar al cliente a identificar los requisitos de un nuevo sistema. En el prototipo se implantan sólo aquellos aspectos del sistema que se entienden mal o son desconocidos. El usuario, mediante el uso del prototipo, descubrirá esos aspectos o requisitos no captados. Todos los elementos del prototipo serán posteriormente desechados.

- **Prototipo evolutivo**. Es un modelo de trabajo del sistema propuesto, fácilmente modificable y ampliable, que aporta a los usuarios una representación física de las partes claves del sistema antes de la implantación. Una vez definidos todos los requisitos, el prototipo evolucionará hacia el sistema final. En los prototipos evolutivos, se implantan aquellos requisitos y necesidades que son claramente entendidos, utilizando diseño y análisis en detalle así como datos reales.

La construcción de una maqueta no tiene mayor dificultad, pues se trata, simplemente, de implementar el dialogo entre la computadora y el usuario para confirmar las necesidades del usuario. Esto puede ser suficiente en problemas simples, pero para problemas más complejos el dialogo no es suficiente para comprender los requisitos. Es entonces cuando se 0pta por uno de los otros dos tipos de prototipo.

3.1. Ciclo de vida con prototipo desechable

Esta aproximación al desarrollo de software fue popularizada por Gomoa. El modelo de prototipado desechable pretende asegurar que el producto software que se está proponiendo cumple realmente las necesidades del usuario. La aproximación consiste en construir una implementación rápida y no cuidada parcial del sistema antes o durante la etapa de requisitos. El usuario utilizará este prototipo durante un tiempo y proporcionará retroalimentación a los desarrolladores sobre los puntos fuertes y débiles del prototipo. Esta retroalimentación se usa para modificar la especificación de requisitos de modo que refleje las verdaderas necesidades del usuario. Llegados a este punto, los desarrolladores pueden proceder con el diseño e implementación del sistema completo, teniendo la confianza que están construyendo el sistema adecuado (excepto para aquellos numerosos casos en que las necesidades del usuario evolucionen durante el desarrollo del sistema global). Una extensión de este modelo de ciclo de vida es usar más de un prototipo desechable.

La figura 9 muestra la comparación entre el desarrollo con prototipo desechable y el desarrollo secuencial tradicional en cascada. El uso del prototipo desechable al inicio del ciclo de vida aumenta la probabilidad de que tanto los clientes como los desarrolladores tengan un entendimiento mejor de las necesidades del usuario que existen en t_0. Por tanto, su uso no afecta radicalmente al modelo de ciclo de vida (la forma es semejante a la forma de la aproximación clásica), pero aumenta el impacto del sistema resultante. Esto se ve en la figura 9 donde la línea vertical es t_1 (la funcionalidad proporcionada por el sistema a la entrega) es más larga que en la aproximación tradicional. En la figura también puede verse al prototipo mismo, como una pequeña línea vertical que proporciona capacidad limitada y

experimental muy cerca del momento de inicio del desarrollo, t_0. No hay una razón especial para creer que el tiempo durante el que el sistema puede ser mejorado sin incurrir en costos prohibitivos sea mayor con prototipo desechable que en el caso tradicional. Por eso en la figura 9 ese tiempo ($t_3 - t_1$) es el mismo en ambas aproximaciones.

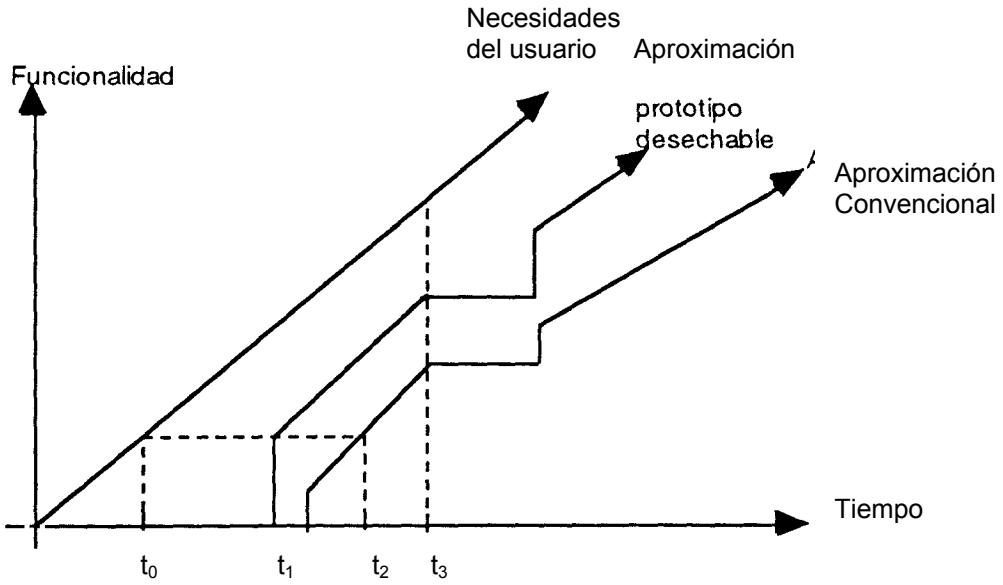

Figura 9. Prototipo desechable frente al desarrollo convencional

3.2. Ciclo de vida con prototipo evolutivo

En este tipo de ciclo de vida (establecido por Jackson) los desarrolladores construyen una implementación parcial que satisface los requisitos conocidos. Entonces el usuario lo utiliza para llegar a comprender mejor la totalidad de requisitos que desea. Mientras el desarrollo incremental implica que se comprenden la mayor parte de los requisitos desde el principio y se elige implementarlos en subconjuntos de capacidad incremental o aumentada, el prototipo evolutivo implica que no se conocen desde el principio todos los requisitos, y que necesitamos experimentar con un sistema operacional para aprender cuáles son. Nótese que en el prototipo desechable lo lógico es implementar sólo aquellos aspectos del sistema que se entienden mal, mientras que en el prototipo evolutivo lo lógico es empezar con aquellos aspectos que mejor se comprenden y seguir construyendo apoyados en los puntos fuertes, no en los débiles. En el caso de aplicaciones complejas, no es razonable esperar que la construcción del prototipo sea rápida. Por tanto la idea de prototipo de construcción rápida y poco cuidadosa se corresponde siempre con el desechable, nunca con el evolutivo. En el prototipo evolutivo incluso los requisitos de fiabilidad y rendimiento se implementan desde el principio. Esto es debido a que este tipo de requisitos, hoy por hoy, no se pueden incluir o ajustar con posterioridad, sino que para cumplirse deben contemplarse al mismo tiempo que se desarrollan las funcionalidades. No obstante, la idea del prototipo evolutivo se centra en funcionalidad y contenido.

Puede considerarse que, en cierto modo, el prototipado evolutivo es una extensión del desarrollo incremental. En aquél, el número y la frecuencia de prototipos operacionales es mayor que el número de versiones del sistema en la aproximación incremental. El énfasis, en el prototipado evolutivo, se pone en la evolución, de un modo más continuo, hacia una solución; en lugar de mediante un número discreto de versiones del sistema.

Mediante tal aproximación, surge un prototipo inicial, antes que con el modelo clásico debido a lo reducido de sus funcionalidades, normalmente el prototipo demuestra la funcionalidad en aquellos requisitos bien entendidos (en contraste con el prototipo desechable, donde normalmente se implementan primero los aspectos peor entendidos). Este primer prototipo proporciona un marco para el resto del desarrollo. Cada sucesivo prototipo explora una nueva área de necesidades de usuario, además de refinar las funciones del prototipo anterior. Como resultado de este modo de desarrollo, la solución software evoluciona acercándose cada vez más a las necesidades del usuario (ver figura

10). Pasado cierto tiempo, también el sistema software así construido deberá ser rehecho o sufrir una profunda reestructuración con el fin de seguir evolucionando.

Al igual que el desarrollo incremental, la pendiente (línea A-C) es más pronunciada que en la aproximación en cascada (línea B-D) puesto que el prototipo evolutivo fue diseñado para ser mucho más adaptativo. Obsérvese que en este caso la línea A-C no tiene forma de escalera (como en el caso del desarrollo incremental) puesto que los subdesarrollos bien planificados y bien definidos del modelo incremental han sido sustituidos, en la aproximación evolutiva, por una continua afluencia de funcionalidades nuevas, y a veces experimentales.

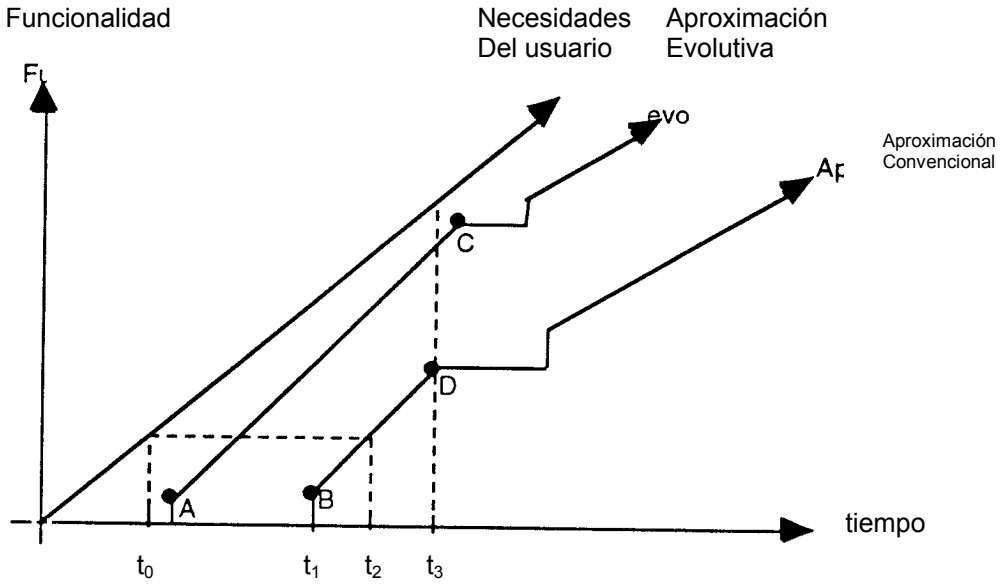

Figura 10. Modelo evolutivo frente a modelo convencional

4. MODELO EN ESPIRAL

El modelo en espiral para el desarrollo de software representa un enfoque dirigido por el riesgo para el análisis y estructuración del proceso software. Fue presentado por primera vez por Boehm en 1986. El enfoque incorpora métodos de proceso dirigidos por las especificaciones y por los prototipos. Esto se lleva a cabo representando ciclos de desarrollo iterativos en forma de espiral, denotando los ciclos internos del ciclo de vida análisis y prototipado precoz, y los externos, el modelo clásico. La dimensión radial indica los costes de desarrollo acumulativos y la angular el progreso hecho en cumplimentar cada desarrollo en espiral. El análisis de riesgos, que busca identificar situaciones que pueden causar el fracaso o sobrepasar el presupuesto o plazo, aparecen durante cada ciclo de la espiral. En cada ciclo, el análisis del riesgo representa groseramente la misma cantidad de desplazamiento angular, mientras que el volumen desplazado barrido denota crecimiento de los niveles de esfuerzo requeridos para el análisis del riesgo como se ve en la Figura 11.

La primera ventaja del modelo en espiral es que su rango de opciones permiten utilizar los modelos de proceso de construcción de software tradicionales, mientras su orientación al riesgo evita muchas dificultades. De hecho, en situaciones apropiadas, el modelo en espiral proporciona una combinación de los modelos existentes para un proyecto dado. Otras ventajas son:

- Se presta atención a las opciones que permiten la reutilización de software existente.
- Se centra en la eliminación de errores y alternativas poco atractivas.
- No establece una diferenciación entre desarrollo de software del sistema.

- Proporciona un marco estable para desarrollos integrados hardware-software.

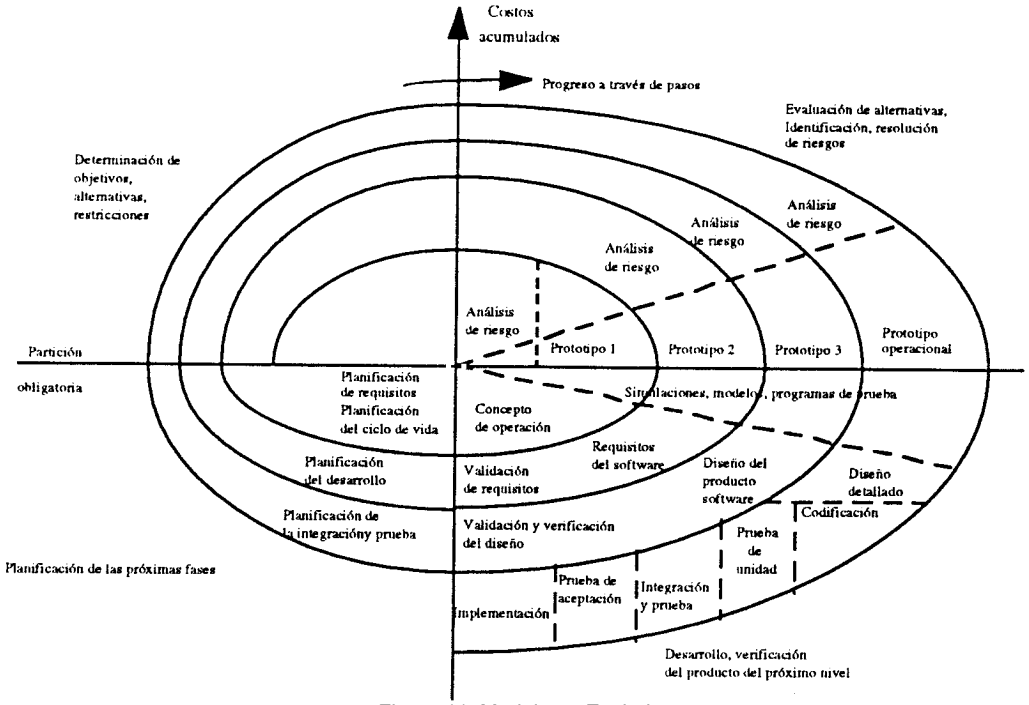

Figura 11. Modelo en Espiral

En realidad el modelo en espiral no significa una visión radicalmente distinta de los modelos tradicionales, incremental o prototipado. Cualquiera de los tres modelos pueden verse con una representación espiral. Este modelo de Boehm es más bien una formalización o representación de los modelos de ciclo de vida más acertada que la representación en forma de cascada, pues permite observar mejor todos los elementos del proceso (incluido riesgos, objetivos, etc.). No obstante, no pasa de ser eso: una mejor representación de los modelos de ciclo de vida, no un ciclo de vida en sí mismo.

Referencias Bibliográficas

- *"Ingeniería de Software", de Natalia Juristo Juzgado, Universidad Politécnica de Madrid.*