Ciclo de Vida[1]

Introducción:

- Es el conjunto y la disposición de las actividades que suceden desde la concepción del sistema hasta que el mismo es desinstalado de la última máquina del usuario.
- Tiene como función establecer el criterio bajo el cual proceder de un tipo de tareas a otro.
- Dependiendo del ciclo de vida que uno elija, es posible mejorar la velocidad del desarrollo, mejorar la calidad, facilitar el seguimiento, reducir la exposición a riesgos o mejorar el contacto con el cliente. La elección errónea puede producir reducción en la productividad, retrabajo y frustración.

[1]Basado en un extracto del capítulo 7 – "Lifecycle Planning" del libro "Rapid Development" de Steve McConell.

Ciclo de Vida – Code and Fix

Este ciclo de vida es raramente útil pero sin embargo altamente utilizado. Si no se ha explícitamente elegido un ciclo de vida probablemente sea éste el que esté siendo usado.

El ciclo comienza con una idea general sobre lo que debe ser construido. Es posible que exista una especificación, sin ser la misma necesaria para comenzar a construir. Luego se comienza a usar cualquier combinación de metodologías de diseño informal, codificación y testing hasta que el producto está listo para ser liberado.

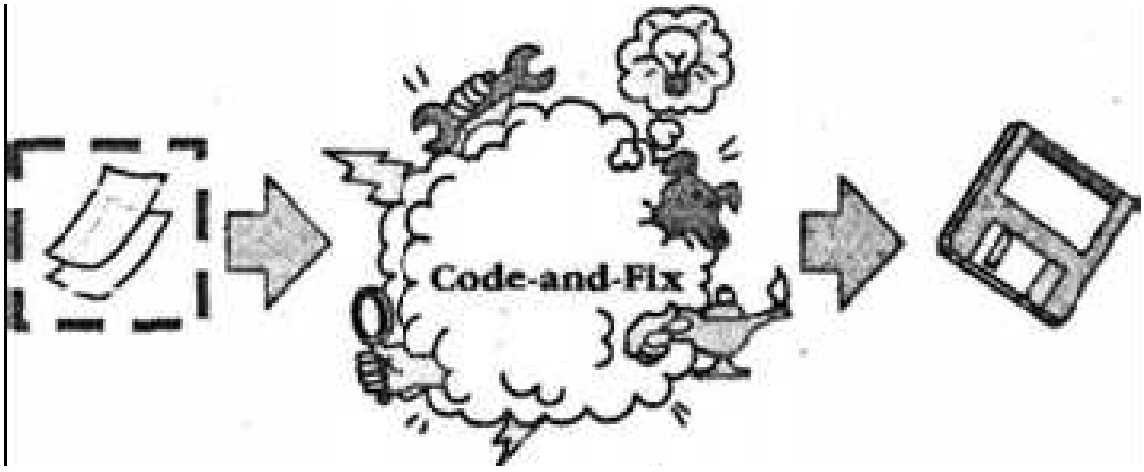
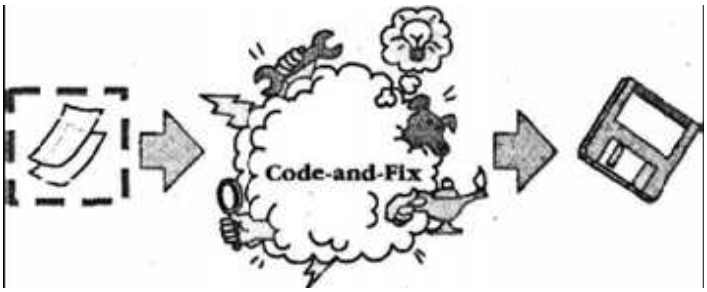


Figura 1: Code & Fix Model.



El modelo tiene dos ventajas. Primero, no posee overhead: uno salta directamente a codificar, uno cree que posee inmediatamente signos de progreso. Segundo, no requiere ningún tipo de conocimiento excepto saber programar: cualquiera puede usarlo.

Para pequeños proyectos que uno sabe que serán descartados rápidamente luego de ser usados, este modelo puede llegar a ser útil (programas para pruebas de concepto (proof-of-concept), demos de vida acotada, o prototipos desechables.).

Para cualquier otro proyecto este modelo es peligroso. Puede ser que no tenga overhead pero tampoco tenemos forma de asegurar que existe progreso alguno. Uno codifica hasta que el producto se considera listo ("listo" no tiene métrica asociada tampoco). No tenemos forma de controlar la calidad ni de identificar riesgos. Es muy probable aquí que se alcancen puntos en el proyecto donde sea necesario descartar absolutamente todo lo realizado justamente por no estar el objetivo bien definido y comprendido.

Ciclo de Vida – Pure Waterfall

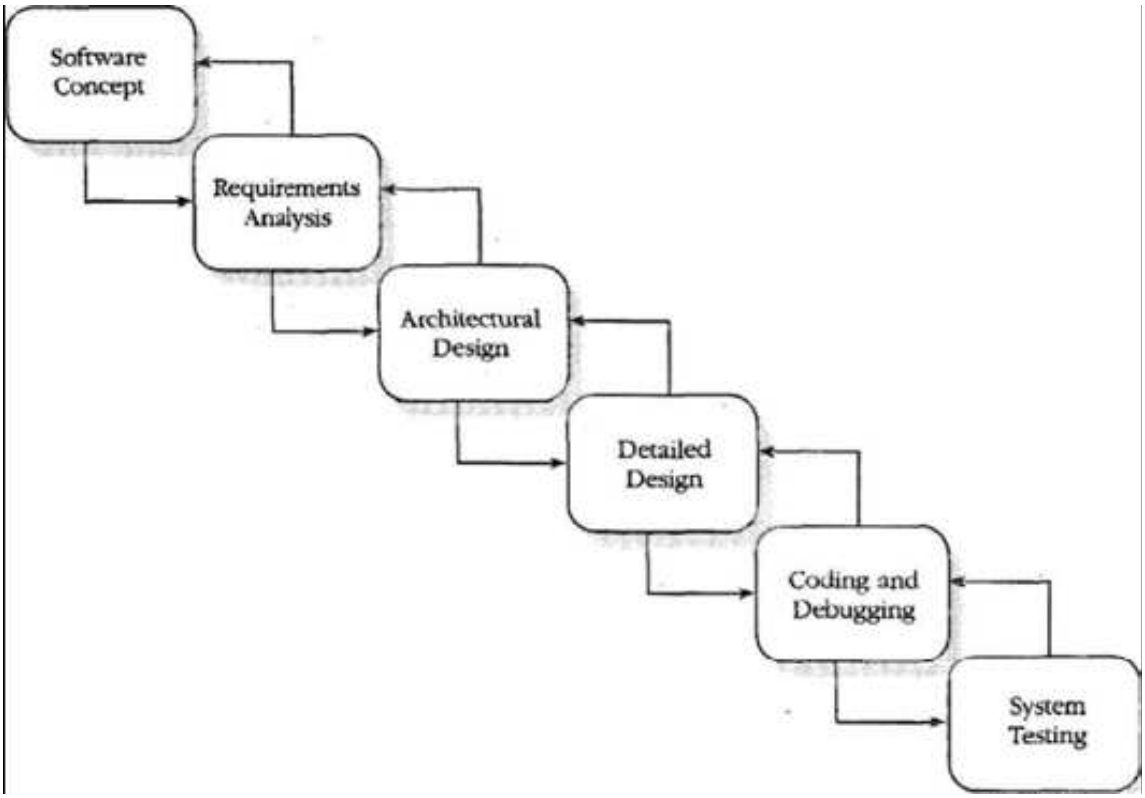
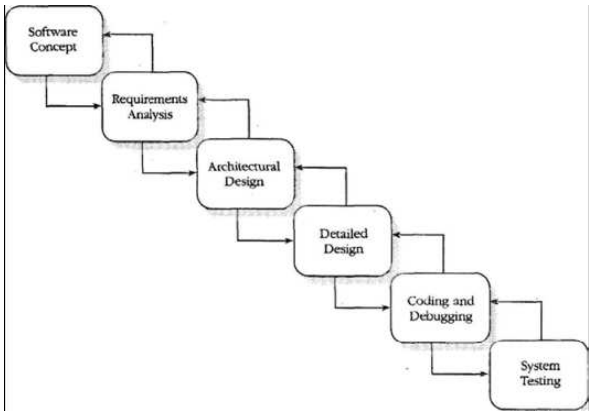


Figura 2: The pure waterfall model.

Bajo este ciclo de vida el proyecto progresa a través de una secuencia ordenada de pasos desde la concepción inicial del Software.

El proyecto contempla una revisión al final de cada paso para determinar si se pasará al siguiente. Esto hace que sea posible permanecer por un tiempo indeterminado sobre uno de los pasos si dicha revisión no es alcanzada.

El modelo waterfall es orientado a la documentación lo que significa que la gran mayoría de los productos de software que son intercambiados entre etapas son documentación. Las etapas no se solapan en este modelo.



En proyectos donde hay alta estabilidad funciona bien. La estabilidad se relaciona con un gran conocimiento de los requerimientos y de los métodos que serán usados.

En estos casos no sólo es un modelo eficiente sino que es el correcto a usar puesto que tenemos la oportunidad de encontrar errores de alto impacto en etapas tempranas y de bajo costo.

El modelo contribuye a minimizar el overhead en la planificación porque es posible hacer la actividad de planificación al inicio. Por otra parte, no tenemos resultados tangibles hasta el fin del proyecto. El avance debe ser entonces medido a partir de la documentación generada durante las etapas.

Funciona bien cuando los requerimientos de calidad dominan el costo y el cronograma.

Las desventajas del modelo radican en la dificultad de conocer los requerimientos al 100% al comienzo del proyecto, antes de comenzar a diseñar y antes de que cualquier código sea escrito.

Ciclo de Vida – Pure Waterfall - Salmon's Model

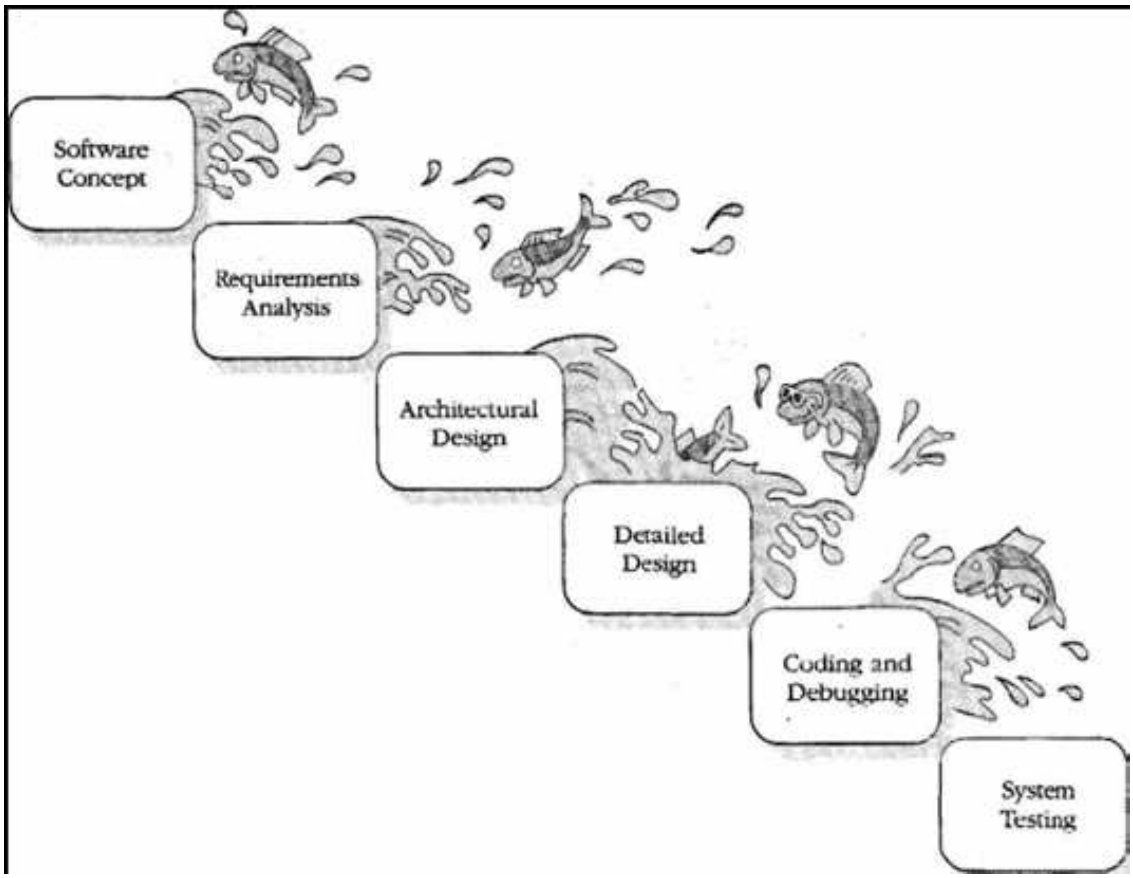


Figure 3: Salmon's Waterfall Model.

"Es posible retroceder en las fases pero esto puede costarnos el proyecto"

Ciclo de Vida – Sashimi (Waterfall + Overlapping Phases)

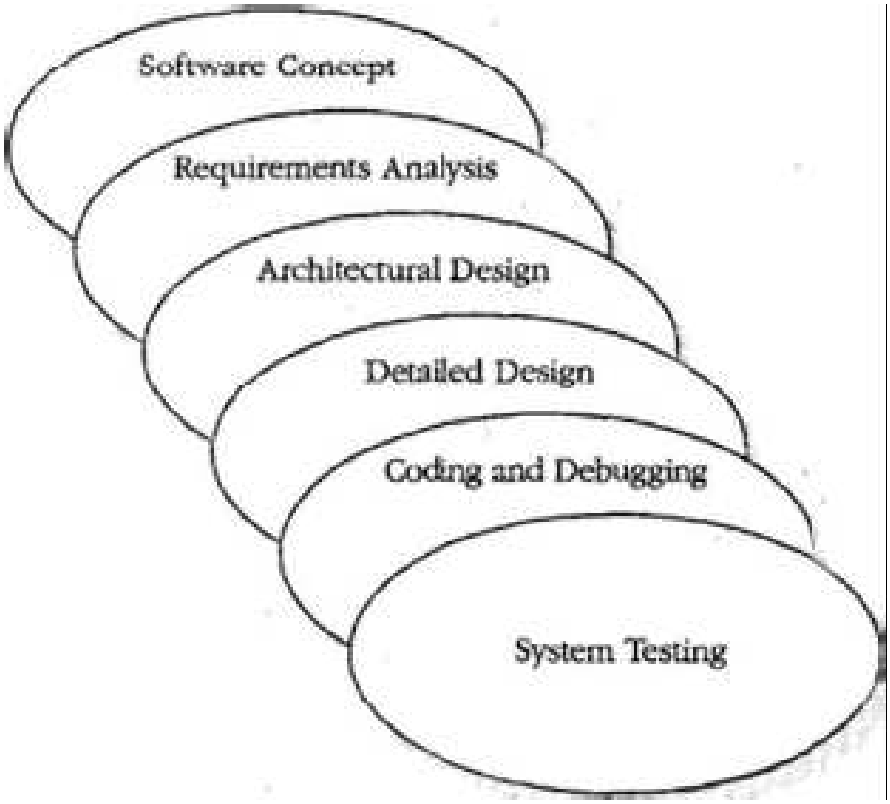
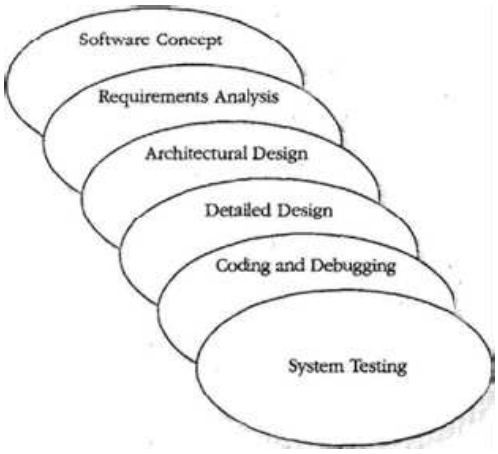


Figure 4: Shashimi Model.

En el modelo puro Waterfall, la documentación ideal es la que es pasada "de mano en mano" entre los equipos de trabajo que operan en las diferentes fases. La pregunta es "¿Por qué?".

Si el equipo de trabajo es el mismo que opera en cada una de las fases, entonces no es necesaria la documentación creada para transmitir el conocimiento entre fases. Este cambio reduce la documentación a la necesaria para el posterior mantenimiento del Software.



Como existe solapamiento entre fases, los hitos (milestones) son más ambiguos y esto hace más difícil realizar el seguimiento con cierto nivel de seguridad.

Efectuar actividades en paralelo requiere que las actividades solapadas estén bien comunicadas, estamos expuestos a que cambios o novedades en actividades "viejas" no sean comunicadas a las actividades "nuevas" y por ende exista inconsistencia a lo largo del ciclo.

Ciclo de Vida – Waterfall with Subprojects

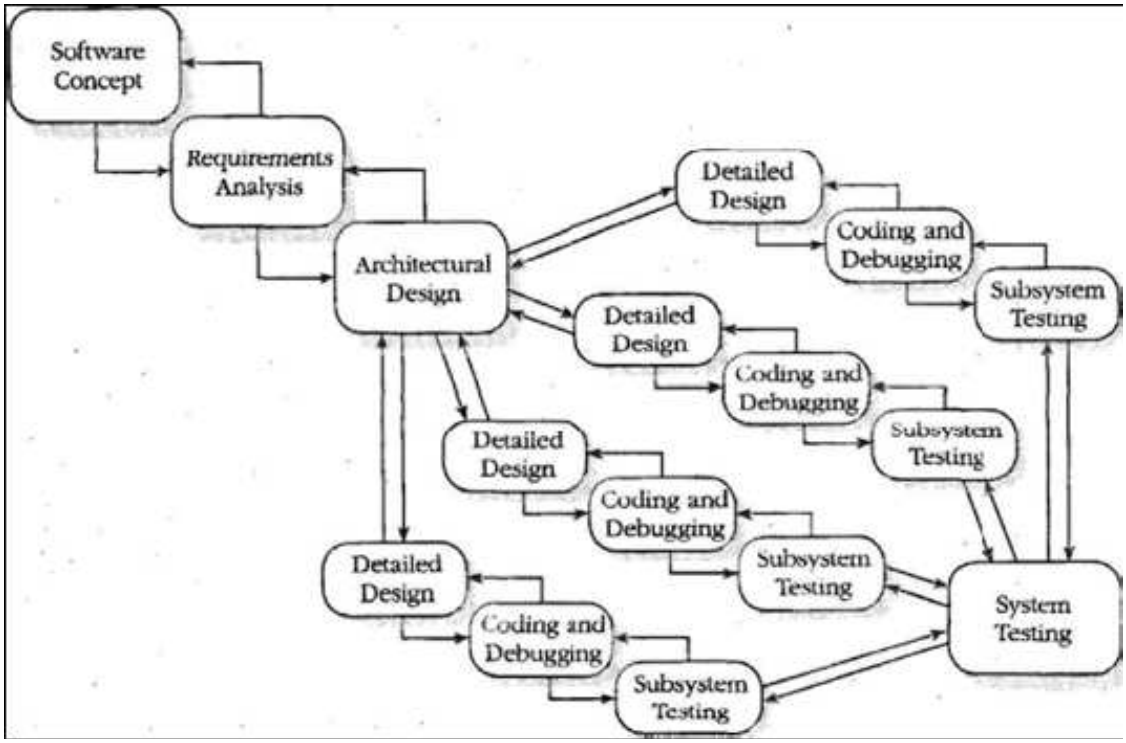
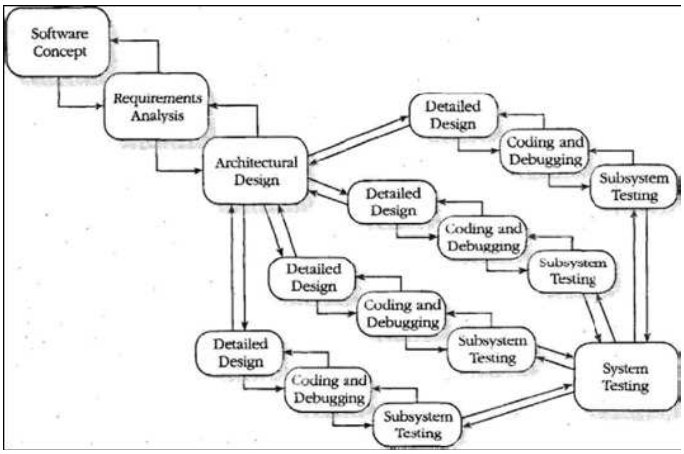


Figure 5: Waterfall Model with Subprojects.

Otro problema con el modelo waterfall puro es que se supone que debemos completar el 100% del diseño arquitectónico antes de comenzar con el diseño detallado, y que a su vez debemos completar el 100% del diseño detallado antes de comenzar a codificar.

Los sistemas tienen ciertas áreas donde existen sorpresas de diseño, pero también existen áreas donde la tarea es simple e incluso en algunos casos conocida. ¿Por qué entonces retrasar el comienzo de los subsistemas simples a causa de la complejidad de parte de la arquitectura?

Si es posible partir la arquitectura en subsistemas lógicamente independientes se pueden obtener subproyectos que sean tratados independientemente y en paralelo hasta llegar al momento de integrarlos.



El mayor riesgo de esto es que existan interdependencias no previstas que generen problemas de integración. Este riesgo es posible mitigarlo con una actividad de arquitectura más intenso que en el caso del pure waterfall, pero nunca podremos eliminarlo por completo.

Ciclo de Vida – Waterfall with Risk Reduction

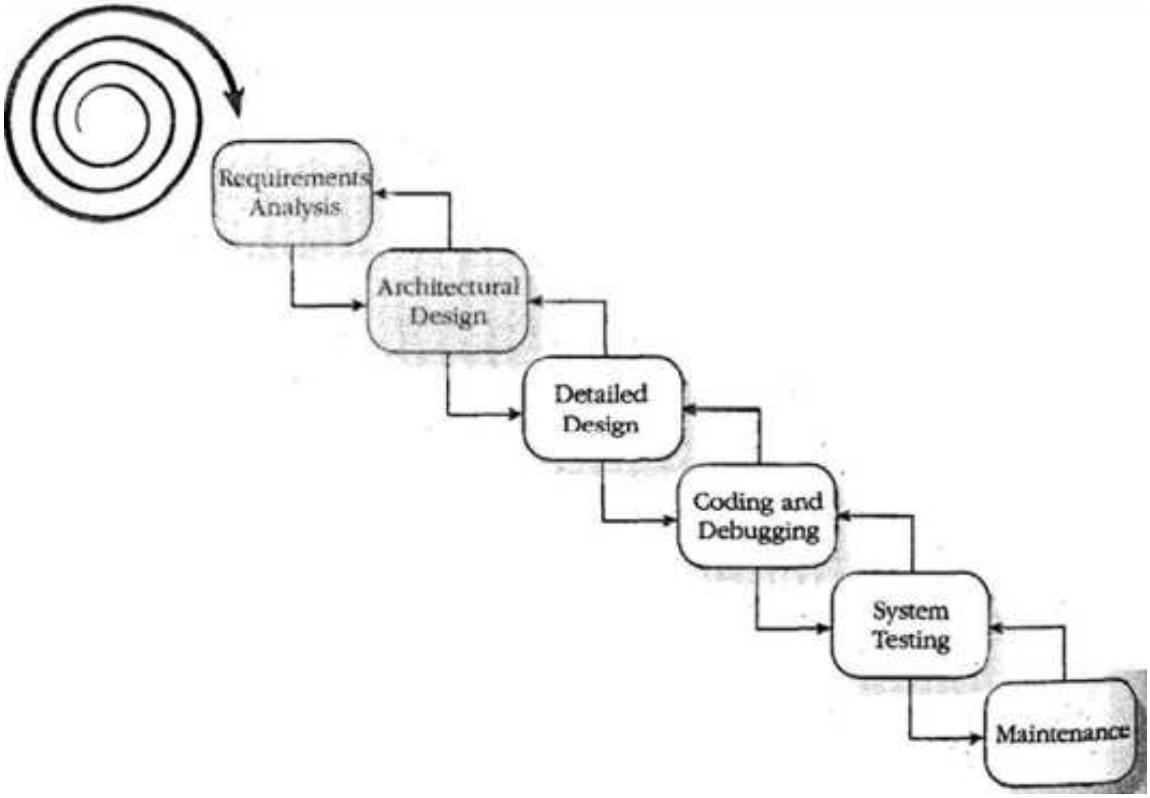
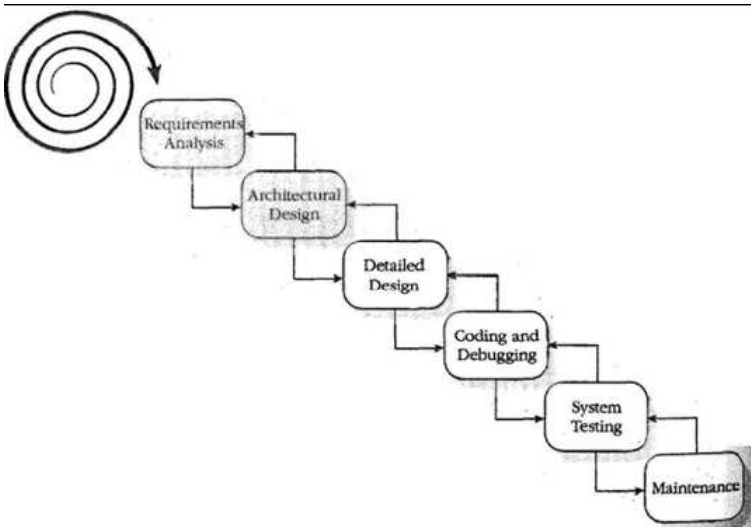


Figure 6: Waterfall with Risk Reduction

(Activities in grey indicate they are addressed during the risk reduction)

Otro de los problemas del waterfall puro es que es necesario tener una precisa definición de los requerimientos antes de comenzar con el diseño arquitectónico. Esta necesidad no solo se encuentra en los requerimientos sino que es necesario para ingresar en una fase contar con gran estabilidad en la anterior y con el total de los productos requeridos para ingresar en la nueva fase.

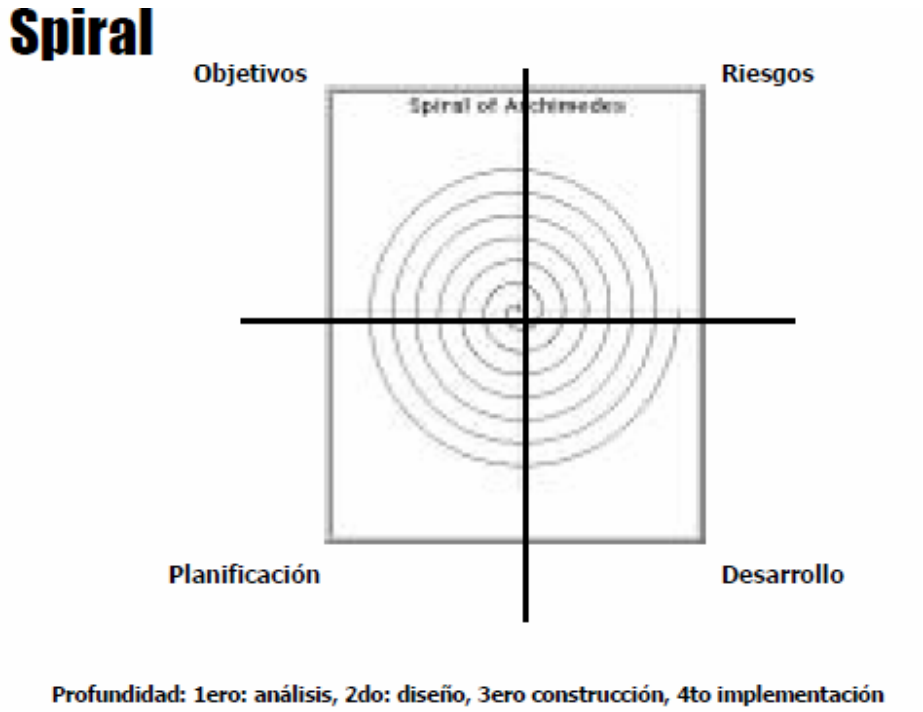
Una posible modificación leve al pure waterfall es la propuesta por "Risk Reduction", donde se incorpora una actividad de prototipado de interfaz o de aquellos componentes de mayor riesgo por la incertidumbre natural al desarrollo de Software. Esta actividad no necesariamente debe centrarse en los requerimientos sino que es posible extenderla al diseño e incluso a la codificación.



La actividad de prototipado es generalmente beneficiosa aunque existe la posibilidad de no poder recolectar mayor información incurriendo en gastos que no tienen rédito en el proyecto.

Dado que luego de la actividad de prototipado el ciclo es idéntico al pure waterfall estamos frente a las mismas dificultades que en dicho modelo.

Ciclo de Vida – Spiral



El ciclo de vida spiral es un modelo orientado a riesgos, donde el proyecto es dividido en mini-proyectos. Cada uno de los mini-proyectos atiende a uno a más riesgos "importantes" hasta que finalmente todos los riesgos con alta exposición han sido atendidos.

El concepto de riesgo, se refiere a requerimientos pobremente entendidos, a arquitectura pobremente definida o comprendida, problemas potenciales de performance, problemas en la tecnología subyacente, y demás temas del proyecto donde sea requerido mayor conocimiento.

Una vez que los riesgos han sido atendidos el ciclo de vida finaliza como el pure waterefall.

La idea detrás del modelo es que uno comienza con un alcance reducido en el centro de la espiral, explora los riesgos, construye el plan para atender los riesgos encontrados, y luego planea el siguiente ciclo. Cada iteración mueve el proyecto hacia un alcance más extenso.

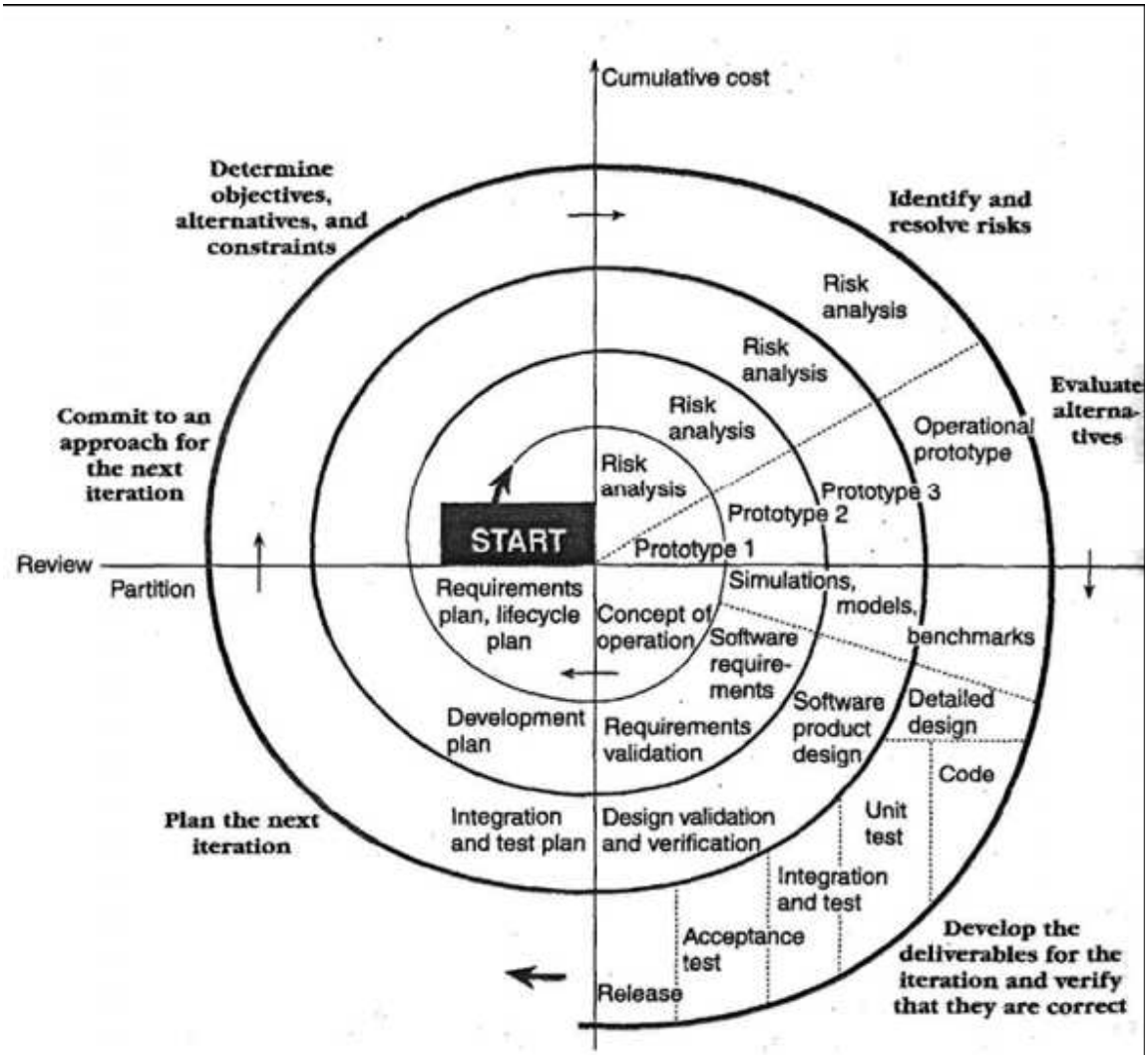


Figure 7: Spiral Model.

Cada iteración comprende 6 pasos
1. Determinar objetivos, alternativas y restricciones.
2. Identificar y resolver riesgos.
3. Evaluar alternativas.
4. Desarrollar los entregables de la iteración y verificar que son correctos.
5. Planificar la siguiente iteración.
6. Si se decide continuar, obtener compromiso para la siguiente iteración.

En el modelo espiral las etapas tempranas son las más económicas. Uno gasta menos desarrollando el concepto de operación del Software de los que gasta en la ingeniería de requerimientos, y menos en los requerimientos de lo que gasta en el diseño, desarrollando el producto y probándolo.

Una de las ventajas más importantes del modelo es que el costo aumenta a medida que el riesgo decrece. A mayor tiempo y dinero invertido, menor la exposición al riesgo. Esto es justamente uno de los atributos más buscados en un proyecto de Software.

El modelo espiral provee igual o mayor control en la gestión del proyecto de la provista por el modelo tradicional pure waterfall. Uno cuenta con puntos de control al finalizar cada iteración. Si el proyecto es inviable debido a razones técnicas o funcionales, es descubierto ésto en forma temprana.

La única desventaja del modelo radica en su complejidad.

Requiere de quién gestiona el proyecto conciencia, atención y conocimiento en gestión.
Puede llegar a ser difícil definir milestones objetivos y verificables, que indiquen cuando uno está en condiciones de agregar una nueva iteración.

En algunos casos el producto alcanzado es suficiente, y los riesgos a los que estamos expuesto moderados lo suficiente como para que no sea requerido continuar "pensando en riesgos", con lo que el modelo orientado a riesgos propuesto por el ciclo de vida espiral se vuelve redundante.

Ciclo de Vida – Evolutionary Prototyping

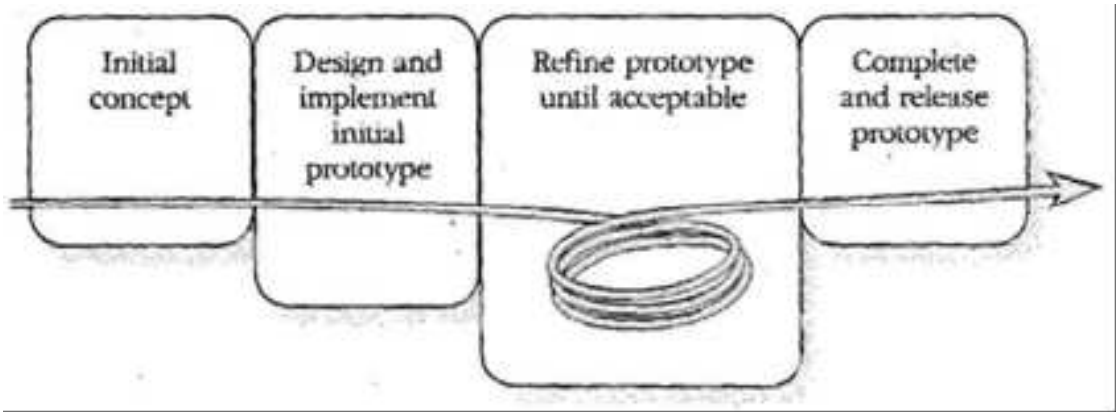


Figure 8: Evolutionary Prototyping Model.

En este modelo uno desarrolla el concepto del sistema a medida que avanzo sobre el proyecto.

Usualmente comenzamos desarrollando los aspectos más visibles del sistema. El resultado es mostrado al cliente y el producto evoluciona en base a la información obtenido por parte de éste. En determinado momento el cliente indica que el prototipo es "suficientemente bueno", se completan las tareas remanentes, especialmente las relacionadas con performance, y el prototipo se convierte así en el release.

Este modelo es especialmente útil cuando los requerimientos son dinámicos o pobremente definidos. También es útil cuando el cliente no tiene la capacidad o voluntad para comprometerse con un requerimiento mejor definido, o no existe nadie que conozca bien el dominio de problema.

La mayor desventaja de este modelo radica en que no es posible saber en qué momento el producto convergerá a la solución. Uno desconoce cuantas iteraciones deberán ser necesarias para obtener finalmente el producto.

Otra desventaja es que el modelo fácilmente se convierte en Code & Fix. Distinguimos "evolutionary delivery" de "code & fix" principalmente porque la actividad de especificación y diseño existen en el primer modelo.

Ciclo de Vida – Staged Delivery

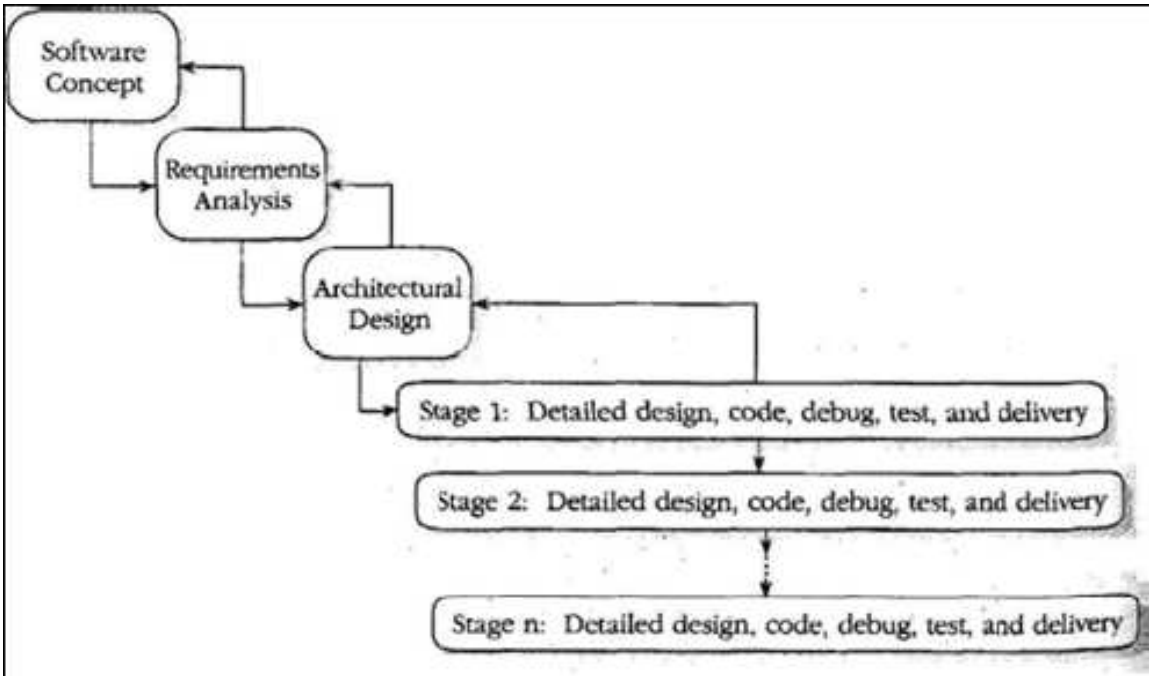


Figure 9: Staged Delivery Model.

A nivel de gerenciamiento, se debe estar seguro que las etapas tienen sentido con respecto al uso que el cliente le dará, y que el entregable de cada una de ellas está "autocontenido"

A nivel técnico, se debe asegurar que las dependencias entre los entregables de las etapas no impiden que el producto sea construido independientemente.

La ventaja principal del modelo radica en que nos permite entregar la funcionalidad principal al principio del proyecto. Si las etapas son planeadas con cuidado, podemos reducir el riesgo en los puntos centrales del Software entregándolos primero y pudiendo así obtener feedback operacional antes del fin del proyecto, cuando aún podemos tomar medidas correctivas. Otra ventaja es que provee signos tangibles de avance desde etapas tempranas, lo que facilita el control sobre presión que pueda ejercer el cliente.