2.1 Introducción

Un **ciclo de vida** es una sucesión de etapas por las que pasa el software desde que se concibe hasta que se deja de utilizar. Cada **etapa del ciclo** lleva asociada una serie de tareas y de documentos de **salida** que serán la **entrada** de la fase siguiente.

La **elección de un ciclo de vida** se realiza de acuerdo con la naturaleza del proyecto, los métodos a usar, y los controles y entregas requeridos.

2.2 Ciclo de vida en Cascada

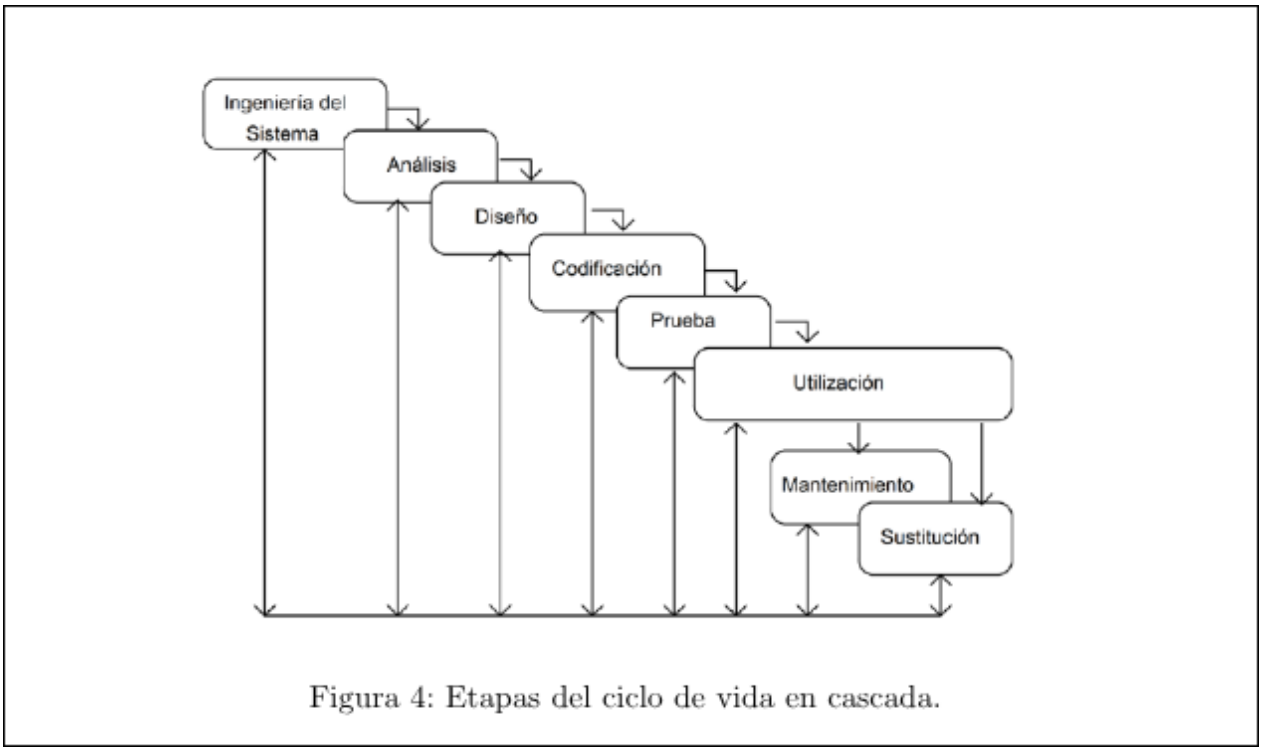

Figura 4: Etapas del ciclo de vida en cascada.

Figure 1: Pasted image 20251103134023.png

Es el ciclo más antiguo. Exige un enfoque sistemático y **secuencial** del desarrollo de software. Se dice que el modelo en cascada está guiado por documentos, ya que **nunca empieza la siguiente fase antes de que se presente el documento de la anterior.**

2.1.1 Fases

1. **Ingeniería y análisis del sistema:** define las interralaciones del software con otros elementos del sistema más complejo en el que ese englobado; es decir, se asignan funciones del sistema al software. Por tanto, comprende los requisitos globales a nivel de sistema, mediante un análisis y diseño a alto nivel.
2. **Análisis de requisitos del software:** análisis detallado de los componentes del

Proceso	Documentos Producidos
Especificaciones del sistema	Especificación funcional. Arquitectura del sistema
Análisis de requisitos	Documento de requisitos
Diseño de la arquitectura del software	Especificación de la arquitectura
Diseño de interfaces	Especificación del diseño
Codificación	Código de programa
Prueba de unidades	Informe de pruebas de unidad
Prueba de módulos	Informe de pruebas de módulo
Prueba de integración	Informe de prueba de integración y manual de usuario final
Prueba del sistema	Informe de prueba del sistema
Prueba de aceptación	Sistema final más la documentación

Figure 2: Pasted image 20251103134147.png

sistema implementados mediante software, incluyendo **datos** a manejar, **funciones** a desarrollar, **interfaces** y **rendimiento** esperado.

3. **Diseño:** en cuanto a estructura de los datos, arquitectura de las aplicaciones, estructura interna de los programas y las interfaces. Es un proceso que traduce los requisitos en una representación del software que permita conocer la arquitectura, funcionalidad e incluso la calidad antes de codificar.
4. **Codificación:** traducción del diseño a un formato que sea legible para la máquina
5. **Prueba:** verificar que no se hayan producido errores en alguna de las fases de traducción anteriores. Deben probarse todas las sentencias de todos los módulos, y no solo los casos habituales.
6. **Utilización:** entrega del software al cliente y comienzo de su vida útil. Etapa que se solapa con las posteriores.
7. **Mantenimiento:** El software sufrir+a cambios a lo largo de su vida útil debido a que el cliente detecte errores, que se produzcan cambios en alguno de los componentes del sistema, o que el cliente requiera modificaciones funcionales. Estos cambios requieren volver atrás en el ciclo de vida (etapas de codificación, diseño o incluso análisis), en función de la magnitud del cambio.
8. **Sustitución:** Cualquier aplicación acaba siendo sustituida. Es una tarea que debe planificarse cuidadosamente y, si es posible, por fases:
    1. Trasvase de la información que maneja el sistema viejo al formato del nuevo.
    2. Mantención de los dos sistemas funcionando en paralelo para comprobar que el nuevo funciona correctamente.
    3. Sustitución completa del sistema antiguo.

2.1.2 Aportaciones

Es el modelo más simple, conocido y fácil de usar. Los procesos se **formalizan** en normas, por lo que sabes lo que tienes que hacer en cada momento. Permite generar software eficientemente, y de acuerdo a las especificaciones, evitando así sobrepasar

fechas de entrega o costes esperados. Los interesados pueden comprobar el estado del proyecto al final de cada fase.

2.1.3 Problemas

En realidad **los proyectos no siguen un ciclo de vida estrictamente secuencial**, sino que siempre hay iteraciones. Un claro ejemplo de ello es la fase de mantenimiento, aunque es frecuente detectar errores en cualquiera de las otras fases.

No siempre se pueden establecer los requisitos desde el primer momento, sino que lo normal es que se vayan aclarando y refinando a lo largo de todo el proyecto. Es habitual que los clientes no tengan el conocimiento de la importancia de la fase de análisis, o bien que no hayan pensado en detalle qué quieren del software.

Hasta que se llega a la fase final, no se dispone de una versión operativa del programa. Esto no resulta óptimo, dado que la mayor parte de los fallos suelen detectarse una vez el cliente puede probar el programa; al final del proyecto, es cuando más costosos son de corregir, y cuando más prisas hay.

Se producen estados de bloqueo, dado que una fase no puede comenzar sin los documentos de salida de la anterior.

2.3 Construcción de prototipos

Este ciclo de vida palia directamente las deficiencias del ciclo de vida en cascada de que: - Sea difícil tener claros todos los requisitos al inicio del proyecto - No se disponga de una versión operativa del programa hasta las fases finales.

2.3.1 Sistemas que resultan ser buenos candidatos

Resulta idóneo en **proyectos con un nivel alto de incertidumbre**, donde los requisitos no están nada claros en primera instancia, ya que se provee un **método para obtener los requisitos de forma fiable** a través del feeback del usuario. Por lo tanto, debería omitirse antes problemas bien comprendidos.

También es de especial interés en aplicaciones que presenten mucha interacción con el usuario, o algoritmos evolutivos. La aplicación no debe contar con una gran complejidad, o las ventajas de la construcción de prototipos se verán superadas por el esfuerzo de un prototipo que habrá que deshechar o modificar mucho. El cliente debe estar dispuesto a probar un prototipo.

2.3.2 Beneficios de los prototipos

- Permiten probar la eficiencia en condiciones similares a las que existirán durante la utilización del sistema.
- Permiten mostrar al cliente la E/S de datos en la aplicación, para detectar problemas en la especificación.

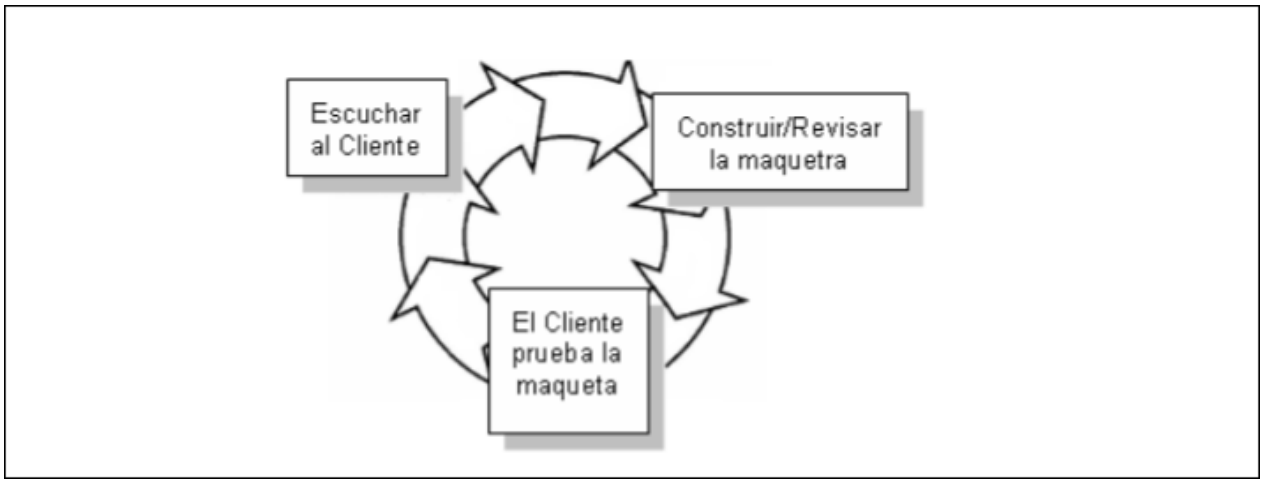

Figure 3: Pasted image 20251103213813.png

- Gran parte del trabajo realizado puede ser utilizado en el diseño final. El diseño se parecerá cada vez más al del producto final, mientras que los componentes codificados deberán ser generalmente optimizados.
- Los prototipos permiten analizar **alternativas y viabilidades**, refinando el resultado final, o abortándolo antes de invertir demasiado dinero en el mismo en caso de no considerarlo viable.

2.3.3 Formas de prototipos

- El papel o ejecutable, que describa la interacción con el usuario
- Que implemente subconjuntos de la funcionalidad requerida, para evaluar el rendimiento o las necesidades del sistema.
- Que realice todo o en parte, pero con características que todavía deban ser mejoradas o incluso simulando la funcionalidad del back-end de la aplicación

2.3.4 Construcción de un prototipo

1. Realizar un modelo del sistema a partir de los requisitos conocidos. No es necesario realizar una definición completa de estos.
2. Diseñar rápidamente el prototipo, centrándose en la arquitectura del sistema y en las interfaces, antes que en aspectos procedimentales.
3. Construir el prototipo, mediante una codificación rápida y que facilite el desarrollo incremental. Es irrelevante la calidad obtenida.
4. Presentar el prototipo al cliente para que lo pruebe y sugiera modificaciones
5. Con estos comentarios, proceder a construir un nuevo prototipo, y así sucesivamente, hasta que los requisitos queden completamente formalizados y se pueda comenzar a desarrollar el producto final.

2.3.5 Problemas

- En muchas ocasiones, el prototipo, que carece de calidad, pasa a ser parte del sistema final por presiones, o bien porque los técnicos se hayan acostumbrado a él.
- Es **posible que nunca se llegue a la fase de construcción final** por falta de recursos, lo cual da lugar a los problemas característicos de un software construido sin seguir procesos de ingeniería al emplear el prototipo como sistema final.
- Es imposible una **predicción de costes fiable**

2.4 Ciclo de vida Incremental

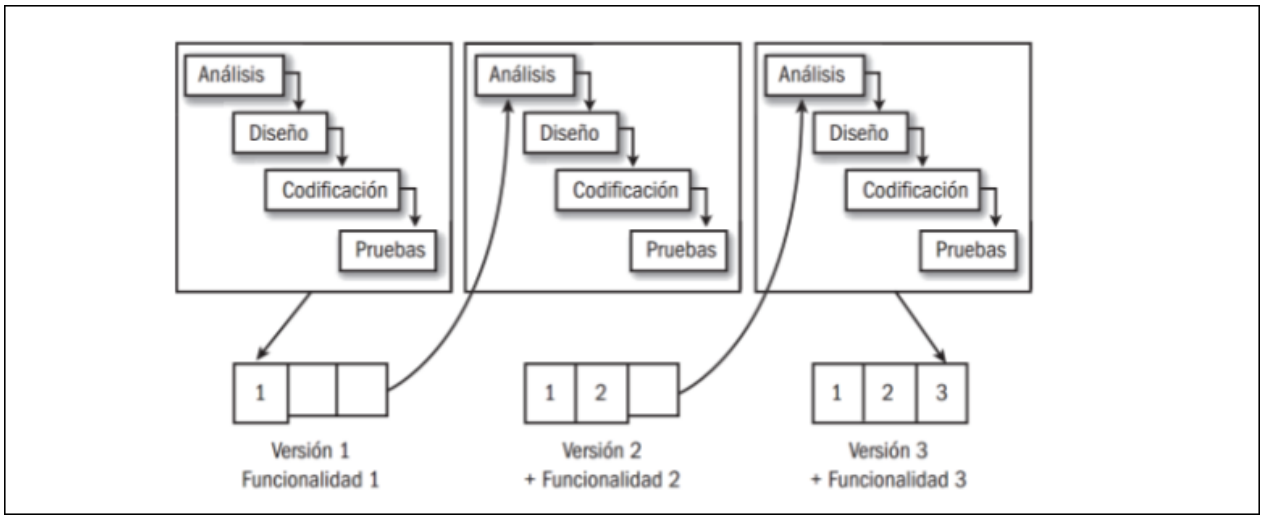

Figure 4: Pasted image 20251103214927.png

Combina elementos del **modelo lineal** en cascada con la **filosofía iterativa** construcción de prototipos. Para ello, se va creando el sistema software mediante incrementos, aportando nuevas funcionalidades o requisitos. De este modo el software ya no se ve como un producto con una fecha de entrega, sino como una **integración de sucesivos refinamientos**.

En comparación al modelo por prototipos, el incremental se centra en obtener un producto operativo en cada iteración, en lugar de simples prototipos; los primeros incrementos simplemente son versiones incompletas del producto final.

Incrementos: componentes funcionales del sistema. Cada incremento es una parte completa y funcional del sistema por sí misma.

2.4.1 Sistemas que resultan ser buenos candidatos

- Entornos con incentidumbre (prototipos si hay mucha incetidumbre)

2.4.2 Ventajas

- Soluciona parte de los problemas del modelo en cascada, al mejorar la **comunicación con el cliente**, y al existir **detecciones de errores** con más antelación.
- Favorece la modulación del software al ser un modelo iterativo

2.4.3 Desventajas

- Si la parte de requisitos no se encuentra en la fase iterativa, será difícil determinar si son válidos, y se seguirían detectando los errores relativamente tarde, al haberse construir ya productos operativos en cada iteración.
- Y, aunque se realice de todos modos una fase de análisis de requisitos en cada iteración en un incremento no hay contacto con el cliente hasta darlo por finalizado, así que los requisitos mal interpretados siguen siendo detectados relativamente tarde (aunque será menos grave que si el análisis se realizase tan solo al comienzo del proyecto).

2.5 Técnicas de cuarta generación

Son un conjunto diverso de métodos y herramientas con el objetivo de facilitar el desarrollo de software, al permitir especificar características del software a alto nivel, y generar código fuente automáticamente a partir de estas especificaciones: - Acceso a bases de datos utilizando lenguajes de consulta de alto nivel sin ser necesario conocer la estructura de estas. - Generación de código. - Generación de pantallas y entornos gráficos - Generación de informes

Con esta automatización, se reduce la duración de las fases del ciclo de vida clásico, especialmente de la codificación. A pesar de ello, estas técnicas no han obtenido los resultados previstos cuando se comenzaron a desarrollar, con intenciones de eliminar la necesidad de la codificación manual y de incluso analistas. Por lo menos, consiguen eliminar las tareas más repetitivas y tediosas.

2.5.1 Críticas más habituales

- No son más fáciles de utilizar que los lenguajes de tercera generación
- El código fuente que producen es ineficiente
- Generalmente, solo son aplicables al software de gestión

2.6 Modelo en Espiral

Es un **modelo iterativo que combina las principales ventajas del modelo de ciclo de vida en cascada y el modelo de construcción de prototipos**. Su principal característica es incorporar el **análisis de riesgos** en el propio ciclo de vida, de modo que los prototipos son usados para reducir el riesgo, incluso permitiendo finalizar el proyecto antes de embarcarse en el desarrollo final so mp se considera viable.

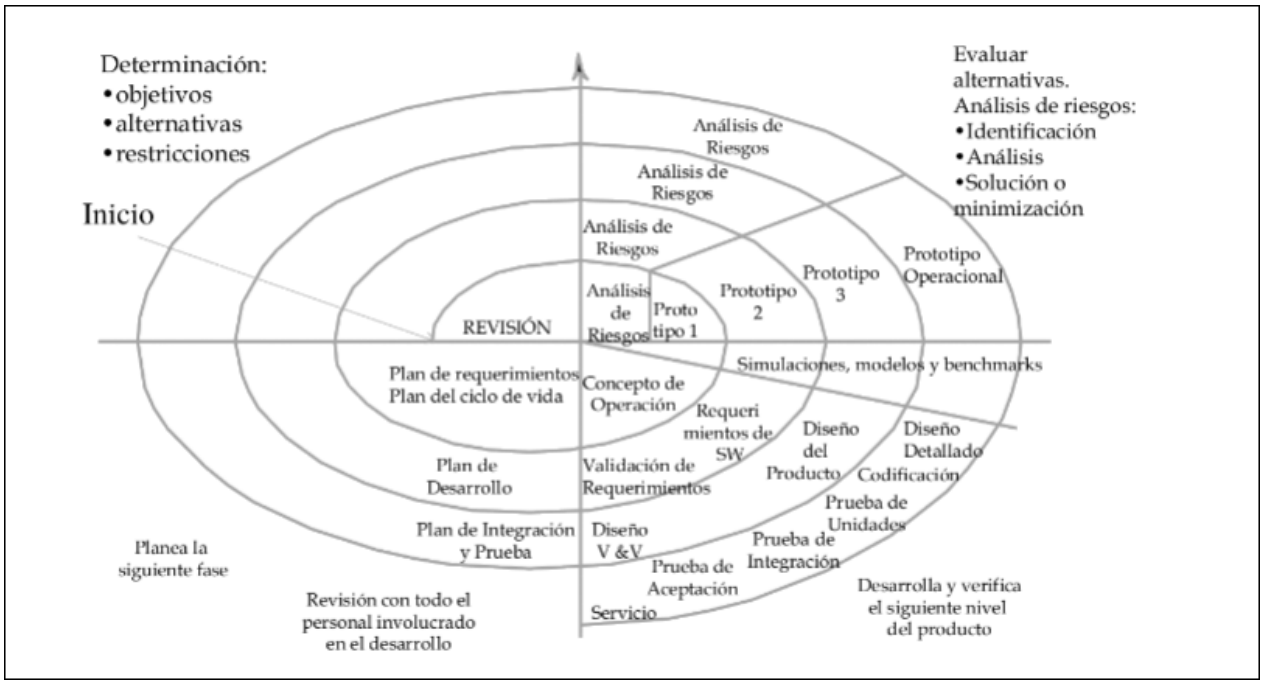

Figure 5: Pasted image 20251103224556.png

2.6.1 Descripción del modelo

El proceso se representa como una espiral, en donde cada ciclo representa una fase del proceso, en función de lo que mejor se ajuste al proyecto; por ejemplo, el ciclo más interno puede relacionarse con la factibilidad, el siguiente con la definición de requisitos, el siguiente con el diseño, etc. Siempre se actúa en función a los riesgos del proyecto presente, y por ello se incluyen las actividades de la gestión de riesgos para reducirlos.

Se definen un total de cuatro sectores: 1. **Definición de objetivos:** objetivos de la fase actual y restricciones con las que deben alcanzarse. Finaliza con la identificación de diferentes alternativas que permitirán alcanzar estos objetivos. 2. **Análisis de riesgos:** se identifican los riesgos de las diferentes alternativas, y cómo tratarlos, con el objetivo de escoger una de las alternativas 3. **Desarrollo y validación:** se realiza el análisis de viabilidad de las alternativas y, si se ha reducido el riesgo a niveles aceptables, se realizan las actividades de ingeniería que construyen el producto; es decir, se generan entregable de los procesos clásicos, siguiendo, por ejemplo, un ciclo en cascada. 4. **Revisión y planificación:** todas las personas implicadas, incluso clientes, revisan los productos desarrolladores y se planificaría la ejecución del siguiente ciclo. Se finaliza con la toma de decisión de finalizar o continuar el proyecto.

2.6.2 Ventajas

- Es mucho más realista que el ciclo de vida clásico
- Permite la utilización de prototipos en cualquier etapa de la evolución del proyecto.
- Existe un reconocimeinto explícito de las alternativas para realizar el proyecto.

- Existe una identificación de los riesgos, junto con las diferentes maneras de tratarlos, eliminando las ilusiones de avance.
- Se adapta a cualquier tipo de actividad, incluso alejadas de un ciclo de vida tradicional, como la consulta de asesores externos.

2.6.3 Desventajas

- El modelo en espiral se adapta bien a los desarrollos internos, pero necesita un ajuste para la contratación de software, al no contar con la flexibilidad para ajustar el desarrollo etapa por etapa.
- Requiere expertos en la evaluación de riesgos, para realizar evaluaciones apropiadas y no abocarse al desastre.
- Resulta difícil de controlar, y de convencer al cliente de que es manejable

2.6.4 Análisis de riesgos