Ingeniería del Software I

2º

TEMA 2

Procesos, modelos de proceso y ciclos de vida.

Universidad Francisco de Vitoria
UFV Madrid

Grado en Ingeniería Informática
Escuela Politécnica Superior

T2. Procesos, modelos de proceso y ciclos de vida

Universidad Francisco de Vitoria UFV Madrid

Objetivos

- Idea de **proceso de software**: conjunto coherente de **actividades** para la producción de software.
  - Comprender conceptos y modelos sobre **procesos de software**.
  - **Tres modelos de proceso de software genérico** y saber cuándo usarlos.
  - **Principales actividades del proceso de ingeniería** de requerimientos de software, desarrollo, pruebas y evolución
  - **Por qué deben organizarse los procesos** para enfrentar cambios en requerimientos y diseño.

T2. Procesos, modelos de proceso y ciclos de vida

Universidad Francisco de Vitoria UFV Madrid

Índice

Introducción

2.1 Modelos de proceso de software

2.2 Actividades del proceso

2.3 Cómo enfrentar el cambio

Introducción

- Un **proceso de software** es una serie de actividades relacionadas que conduce a la elaboración de un producto de software.
  - Las actividades pueden incluir software desarrollado desde cero como Java o C.
  - El nuevo software empresarial se desarrolla extendiendo sistemas existentes, componentes del sistema
- Existen muchos tipos de procesos de software, pero todos deben incluir cuatro actividades fundamentales para la ISW:
  - **Especificación del software**. Tienen que definirse tanto la funcionalidad del software como las restricciones de su operación.
  - **Diseño e implementación**. Debe desarrollarse el software para cumplir con las especificaciones.
  - **Validación del software**. Hay que validar el software para asegurarse de que cumple lo que el cliente quiere.
  - **Evolución del software**. El software tiene que evolucionar para satisfacer las necesidades cambiantes del cliente.

- Estas actividades forman parte de todos los procesos software.
- En la práctica, son actividades complejas e incluyen subactividades.
- También existen actividades de soporte al proceso como la documentación y el manejo de la configuración del software.
- Las descripciones de los procesos deben incluir:
  - **Productos**: resultados de una actividad del proceso.
  - **Roles**: responsabilidades de las personas que intervienen en el proceso.
  - **Precondiciones y postcondiciones**: declaraciones válidas antes y después de que se realice una actividad del proceso o que se cree un producto.
- Para sistemas críticos se requiere un proceso de desarrollo muy estructurado. Para los sistemas empresariales con requerimientos rápidamente cambiantes, es mejor un proceso más flexible y menos formal.
- Los procesos de software pueden ser definidos por un plan o plan-driven, procesos ágiles.
  - Todas las actividades del proceso se planean por anticipado y el avance se mide según el plan.
  - Se estudiarán en detalle en el tema 3.
  - El plan es incremental y es más sencillo modificar el proceso para reflejar los requerimientos del cliente.

T2. Procesos de software. 2.1 Modelos de procesos de sofware

Universidad Francisco de Vitoria UFV Madrid

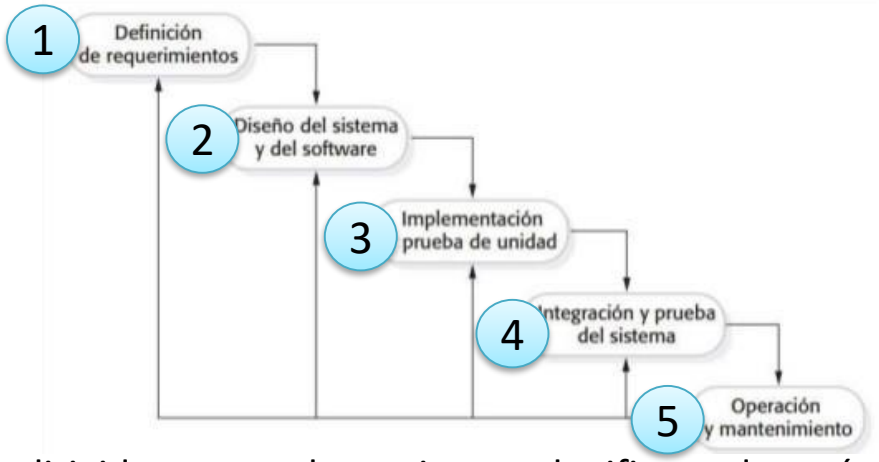
2.1. Modelos de procesos de software

- Modelo de proceso de software es una representación simplificada del proceso.
- Los modelos representan una perspectiva particular del proceso. Por eso, ofrece información parcial sobre él. A veces son genéricos, abstracciones para explicar los diferentes enfoques del desarrollo del software. Son marcos del proceso que luego se adaptan para crear procesos más específicos.
- Los modelos de proceso son:
  - **Modelo en cascada. Waterfall**. Toma las actividades fundamentales del proceso y las representa como fases separadas del mismo.
  - **Desarrollo incremental**. Vincula actividades de especificación, desarrollo y validación. El sistema desarrolla versiones o incrementos y cada versión añade funcionalidad a la versión anterior.
  - **Ingeniería del software orientada a la reutilización**. Existencia de un número alto de componentes reutilizables. El proceso de desarrollo del sistema se enfoca en la integración de estos componentes.
- Estos modelos no son excluyentes, se suelen usar juntos para grandes sistemas.

T2. Procesos de software. 2.1.1 Modelo Cascada

Universidad Francisco de Vitoria UFV Madrid

2.1.1. Modelo en cascada



- Debido al paso de una fase a otra, este modelo se llama **"modelo en cascada" o ciclo de vida del software**.
- El modelo en cascada es un ejemplo de un proceso dirigido por un plan: primero planificar y después trabajar con ellas.
- Las principales etapas del modelo en cascada reflejan las actividades fundamentales del desarrollo:
  - **Análisis y definición de requerimientos**. Los servicios, las restricciones y las metas del sistema se establecen mediante consulta a los usuarios del sistema. Después, se definen con detalle y sirven como una especificación del sistema.
    - Requisitos del sistema. Servicios que se espera que ofrezca el **sistema** y de sus **restricciones**.
    - Requisitos de usuario. Servicios **exactos** que se proporcionarán. Estos requisitos sirven como **contrato** con el cliente.
  - **Diseño del sistema del software**. Asigna los requerimientos para sistemas de hardware o de software al establecer una arquitectura de sistema global. El diseño implica identificar y describir las abstracciones fundamentales del sistema de software y sus relaciones.

T2. Procesos de software. 2.1.1 Modelo Cascada

Universidad Francisco de Vitoria UFV Madrid

- **Implementación y prueba de unidad**. El diseño del software se realiza como un conjunto de programas o unidades del programa. La prueba de unidad consiste en verificar que cada unidad cumpla con su especificación.
- **Integración y prueba del sistema**. Las unidades del programa o los programas individuales se integran y se prueban como un sistema completo para asegurarse de que se cumplan los requerimientos de software. Después de probarlo, se libera el sistema de software al cliente.
- **Operación y mantenimiento**. Esta suele ser la fase más larga del ciclo de vida, donde el sistema se instala y se pone en práctica. El mantenimiento incluye:
  - corregir los errores que no se detectaron en etapas anteriores del ciclo de vida,
  - mejorar la implementación de las unidades del sistema
  - incrementar los servicios del sistema conforme se descubren nuevos requerimientos.

- El resultado de cada fase consiste en uno o más documentos que se autorizaron, es decir, que se firmaron. La siguiente fase no debe comenzar sino hasta que termine la fase previa.
- En la práctica estas etapas se nutren mutuamente de información
  - Ejemplo: Durante el diseño se identifican problemas con los requerimientos.

T2. Procesos de software. 2.1.1 Modelo Cascada

Universidad Francisco de Vitoria UFV Madrid

- El proceso de software **es un modelo secuencial**. **Una etapa ha de terminar para comenzar la siguiente**. También implica retroalimentación.
  - Los documentos han de modificarse de una fase a otra para reflejar los cambios.
  - Estas iteraciones conviene analizarlas bien para que **no se realicen cambios constantes**.
  - Durante la fase final del ciclo de vida. Operación y mantenimiento. El software se pone en servicio, se descubren errores, falta de requerimientos iniciales, etc.
    - Por tanto, el sistema debe evolucionar para mantenerse útil. Estos cambios de software pueden implicar repetición de etapas anteriores del proceso.
- El modelo en cascada en cada fase produce **documentación**.
- Debe usarse cuando la **especificación sea conocida**. Es decir, cuando los requerimientos de entrada se entiendan bien y sea improbable un cambio radical durante el desarrollo del sistema.
- Los procesos basados en transformaciones formales se usan, por lo general, sólo en el desarrollo de sistemas críticos para protección o seguridad.

T2. Procesos de software. 2.1.2. Desarrollo incremental

Universidad Francisco de Vitoria UFV Madrid

2.1.2. Desarrollo incremental (evolutivo)

- El desarrollo de software es **incremental**, que es una parte fundamental de los **enfoques ágiles**, es mejor que un enfoque en cascada para la mayoría de los sistemas empresariales de comercio electrónico y personales.
  - Rara vez se trabaja por adelantado una solución completa del problema, se avanza unos pasos hacia una solución y se retrocede cuando hay errores.
- Al desarrollar el software de manera incremental, resulta **más barato y fácil** realizar cambios en el software conforme éste se diseña.
- Por lo general, los **primeros** incrementos del sistema incluyen **la función más importante o la más urgente**.
- Comparado con el modelo en cascada, el desarrollo incremental tiene tres beneficios importantes:
  - Se reduce el costo de adaptar los requerimientos cambiantes del cliente. La cantidad de análisis y la documentación que ha de reelaborarse es mucho menor.
  - Es más sencillo obtener **retroalimentación** del cliente sobre el trabajo desarrollado que se realizó. Los clientes pueden comentar las demostraciones del software y darse cuenta de cuánto se ha implementado. A partir de documentos, es más difícil.
  - Es posible que sea más rápida la entrega e implementación de software útil a cliente (aunque no se haya incluido toda la funcionalidad).

T2. Procesos de software. 2.1.2. Desarrollo incremental

Universidad Francisco de Vitoria UFV Madrid

- El desarrollo incremental ahora es en cierta forma el enfoque más común para el desarrollo de sistemas de aplicación.
- Desde una perspectiva administrativa, el enfoque incremental tiene dos problemas:
  - El proceso no es visible. Los administradores necesitan entregas regulares.
  - La estructura del sistema tiende a degradarse conforme se tienen nuevos incrementos. El cambio regular tiende a corromper su estructura.
- Los problemas del desarrollo incremental son graves para sistemas grandes y de larga duración donde distintos equipos desarrollan distintas partes del sistema.
  - Esto debe planificarse por adelantado en vez de desarrollarse de forma incremental.

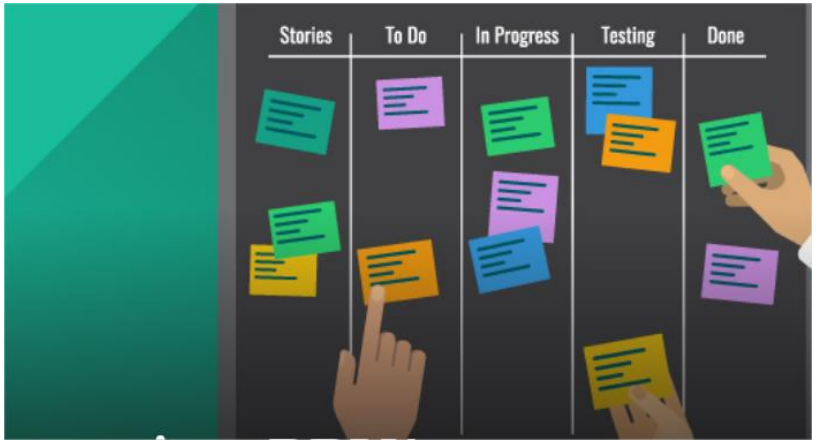
T2. Procesos de software. 2.1.2. Desarrollo incremental

Universidad Francisco de Vitoria UFV Madrid

- Metodologías ágiles.
  - Apple. Steve Jobs. Entrevista que hizo en 2010 en la que nos cuenta cómo Apple se organiza como una startup y mediante una metodología ágil
    - https://www.youtube.com/watch?v=f60dheI4ARg&feature=youtu.be

  - Caso de éxito BBVA. Scrum.
    - Transformación cultural BBVA https://www.youtube.com/watch?v=9t0eJM-2zaQ

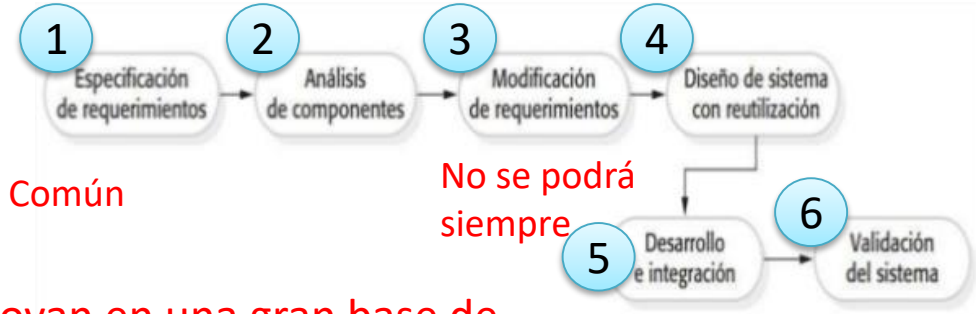
- Desarrollo basado en componentes
  - Caso éxito: ING – Polymer JS https://www.youtube.com/watch?v=2_BQz-ywx0o
    - Otros: React JS, Node, etc.



T2. Procesos de software. 2.1.3. Ingeniería del software orientada a la reutilización

Universidad Francisco de Vitoria UFV Madrid

2.1.3. Ingeniería del software orientada a la reutilización



* Común

No se podrá siempre

* Común

- Los enfoques orientados a la reutilización se apoyan en una gran base de componentes de software reutilizable y en la integración de marcos para la composición de dichos componentes.
- Sus etapas son:
  - **Análisis de componentes**. Se realiza una búsqueda de componentes para implementar dicha especificación.
  - **Modificación de requerimientos**. Se analizan los requerimientos usando la información de los componentes encontrados y luego se modifican. No siempre podrá tener lugar, dependerá el alcance definido por el cliente.
  - **Diseño de sistema con reutilización**. Se diseña un marco conceptual del sistema o se reutiliza uno existente.
  - **Desarrollo e integración**. Se diseña el software que no puede procurarse de manera externa y se integran los componentes y los sistemas.

T2. Procesos de software. 2.1.3. Ingeniería del software orientada a la reutilización

Universidad Francisco de Vitoria UFV Madrid

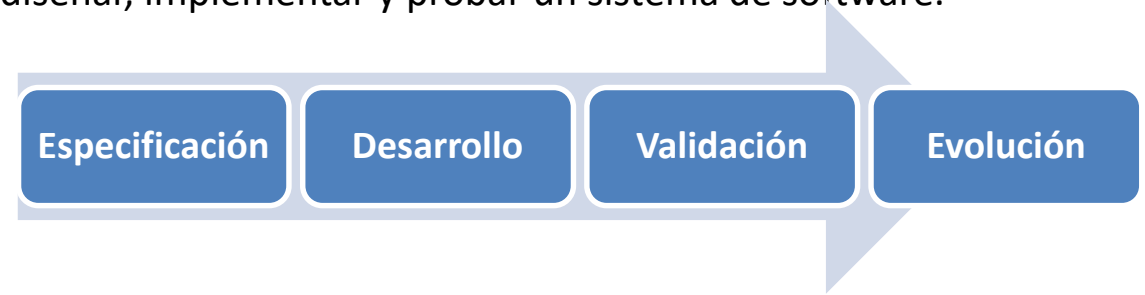
- Existen tres tipos de componentes de software que pueden usarse en un proceso orientado a reutilización:
  - **Servicios Web** que se desarrollan en concordancia para atender servicios estándares y que están disponibles para la invocación remota.
  - **Colecciones de objetos**: que se desarrollan como un paquete para su integración con un marco de componentes como .NET o J2EE o C#.
  - **Sistemas de software independientes**: que se configuran para usar en un entorno particular.

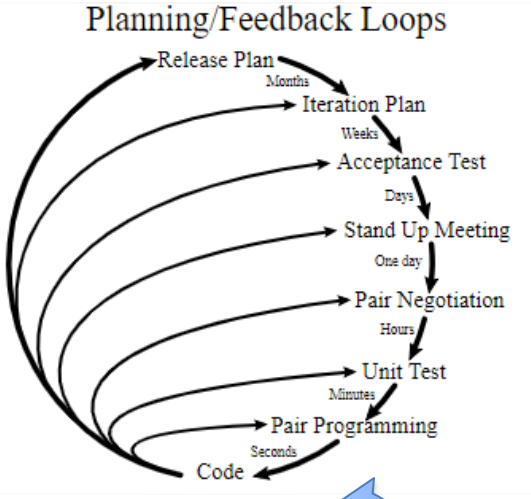
- La ISW orientada a la reutilización tiene la clara ventaja de reducir el **software** a desarrollar, disminuir **costes** y **riesgos**. Por tanto, **entregas más rápidas**.
  - Sin embargo, conlleva compromiso de requerimientos y conduciría hacia un sistema que no cubra las necesidades reales de los usuarios.
  - Se perdería el control sobre la evolución del sistema, conforme las nuevas versiones de componentes reutilizables o estén bajo control de la organización.
- La reutilización del software es muy importante. Tenéis más detalle en los últimos capítulos del libro: 16,17,18 y 19.

T2. Procesos de software. 2.2 Actividades del proceso

Universidad Francisco de Vitoria UFV Madrid

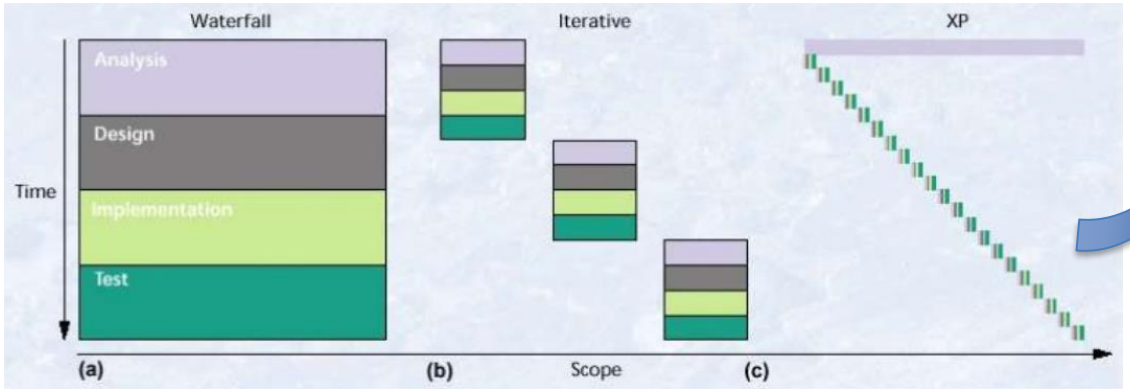
- Los procesos de software real son **secuencias entrelazadas** de actividades técnicas, colaborativas y administrativas con el objetivo final de especificar, diseñar, implementar y probar un sistema de software.



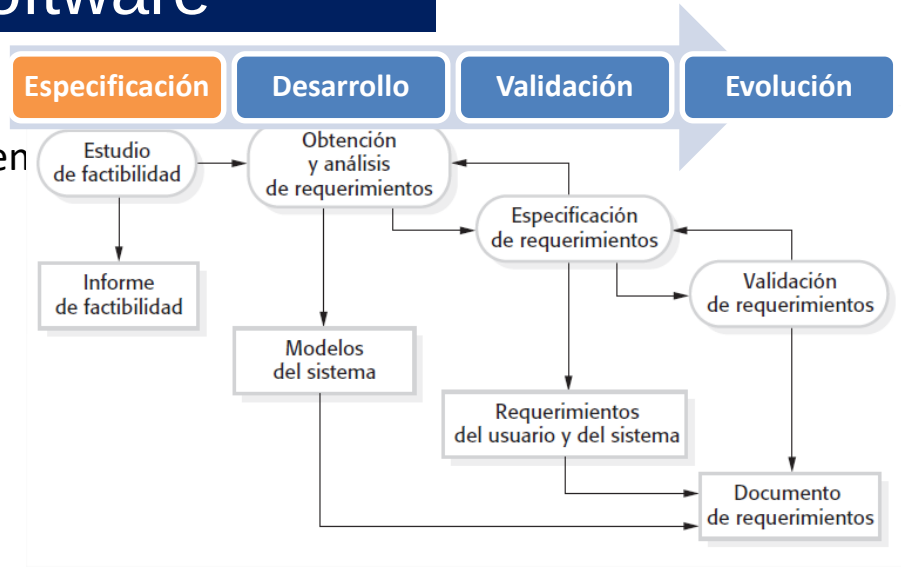


- Las cuatro actividades básicas de proceso se organizan en secuencia, mientras que se entrelazan en el desarrollo incremental.
  - La forma en que se llevan a cabo estas actividades, depende del tipo de software, del personal, y de la inclusión de estructuras organizativas.
  - Ejemplo: **programación extrema**, las pruebas son de tipo ejecutable y se desarrollan antes que el propio programa.

«Embracing change with extreme programming» de Kent Beck



T2. Procesos de software. 2.2 Actividades del proceso. 2.2.1 Especificación del software



2.2 Actividades del proceso

- La especificación del software o la ingeniería de requerimien consisten en el proceso de comprender y definir **qué servicios se requieren del sistema**.
- Es una **etapa crítica** del proceso del software porque los errores que se cometan aquí derivarán en problemas posteriores.

2.2.1 Especificación del software

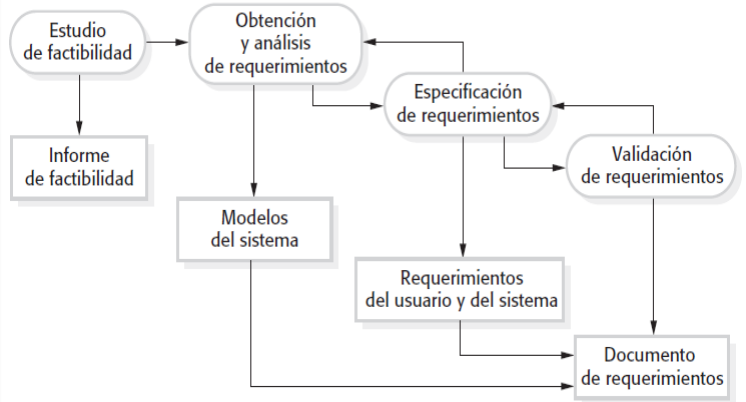
- Existen **cuatro actividades** principales en el proceso de ingeniería de requerimientos:
  1. Estudio de viabilidad: estimación sobre si las necesidades identificadas del usuario se cubren con las actuales tecnologías de software y hardware.
     - Un estudio de factibilidad debe ser **rápido y relativamente barato**. El resultado debe informar la decisión respecto a si se continúa o no continúa con un análisis más detallado.
  2. Obtención y análisis de requerimientos : proceso de derivar los requerimientos del sistema mediante observación de los sistemas existentes, discusiones con los usuarios y proveedores potenciales, análisis de tareas, etc. Puede incluir el desarrollo de uno o más modelos de sistemas y prototipos.

T2. Procesos de software. 2.2 Actividades del proceso. 2.2.1 Especificación del software

Universidad Francisco de Vitoria UFV Madrid

3. Especificación de requerimientos: actividad de transcribir la información recopilada durante la actividad de análisis, en un documento que define un conjunto de requerimientos.

4. Validación de requerimientos: verifica que los requerimientos se completos. Es inevitable descubrir errores en el documento de requ



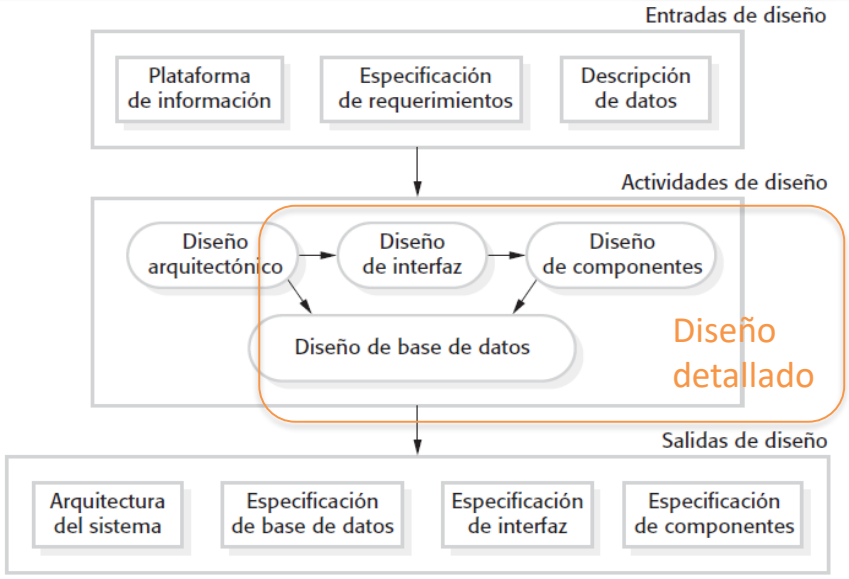
- Las cuatro actividades básicas de proceso se organizan en secuencia, mientras que se entrelazan en el desarrollo.
- El análisis de requerimientos continúa durante la definición y especificación, y a lo largo del proceso salen a la luz nuevos requerimientos, por lo tanto, las actividades de análisis, definición y especificación están vinculadas

T2. Procesos de software. 2.2 Actividades del proceso. 2.2.2 Diseño e implementación

Universidad Francisco de Vitoria UFV Madrid

2.2.2 Diseño e implementación del software

- Convertir una especificación del sistema en un sistema ejecutable.
- Siempre incluye procesos de diseño y programación de software. Involucra corrección en la especificación del software, si se utiliza un enfoque incremental.
- Un **diseño de software** se entiende como una descripción de la **estructura** del software que se va a implementar, los **modelos** y las **estructuras** de datos utilizados por el sistema, las **interfaces** entre componentes del sistema y, en ocasiones, los **algoritmos** usados.
- La mayoría del software tiene **interfaz** junto con otros sistemas de software. En ellos se incluyen **sistema operativo**, **base de datos**, **middleware** y otros **sistemas auxiliares**.
  Éstos constituyen la «plataforma de software»
  el entorno donde se ejecutará el software.
- Se dan **cuatro actividades** que podrían formar parte del proceso de diseño para sistemas de información:
  1. **Diseño arquitectónico**: se identifica la estructura global componentes, sus relaciones y cómo se distribuyen.
  2. **Diseño de interfaz**: se definen las interfaces entre los componentes de sistemas.
     Ha de ser muy precisa. Factible usar un componente sin que otros tengan que saber cómo se implementó.

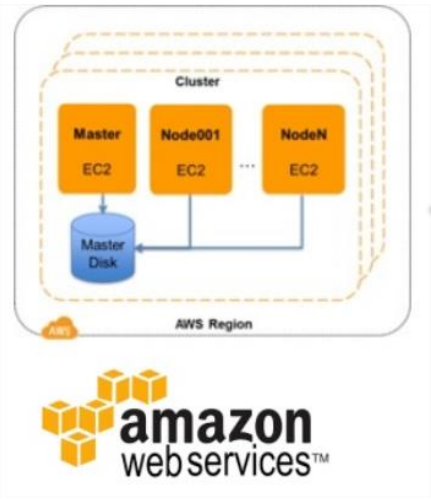


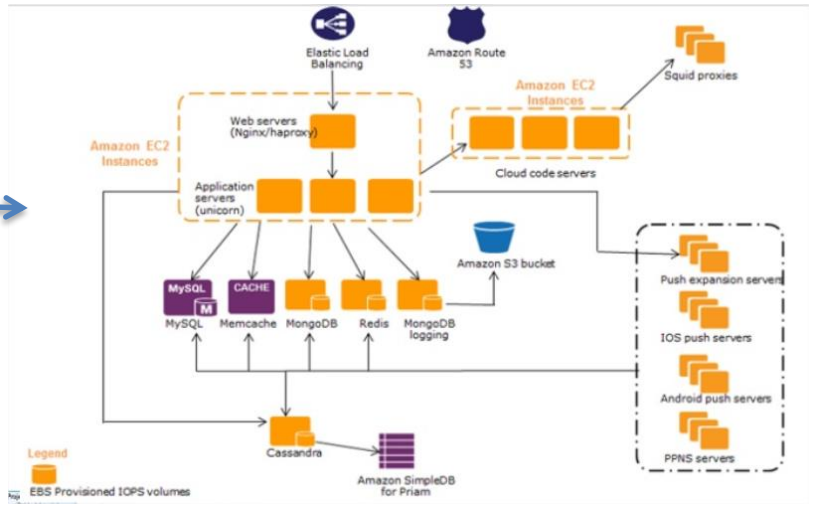
T2. Procesos de software. 2.2 Actividades del proceso. 2.2.2 Diseño e implementación

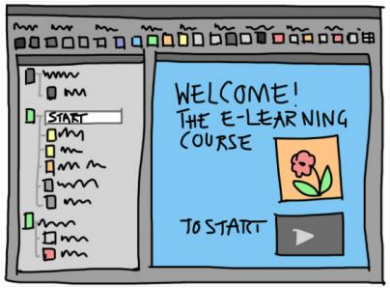
Universidad Francisco de Vitoria UFV Madrid

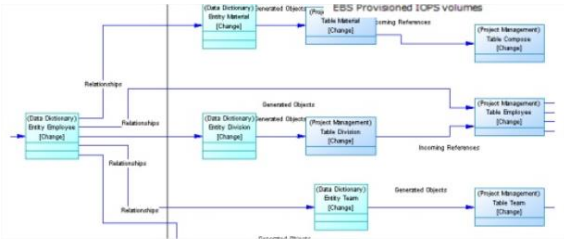
Especificación | Desarrollo | Validación | Evolución

3. **Diseño de componentes**: se toma cada componente del sistema y se diseña cómo funcionará. Esto puede ser un **simple dato** de la funcionalidad que se espera implementar, y al programador se le deja el diseño específico.
4. **Diseño de base de datos**: se diseñan las estructuras del sistema de datos y cómo se representarán en una base de datos. Se reutilizará, se creará una nueva…









- Donde se usen **métodos ágiles de desarrollo**, las **salidas** del proceso de diseño no podrían ser documentos de especificación separados, sino que tendrían que **representarse en el código** del programa.

T2. Procesos de software. 2.2 Actividades del proceso. 2.2.2 Diseño e implementación

Universidad Francisco de Vitoria UFV Madrid

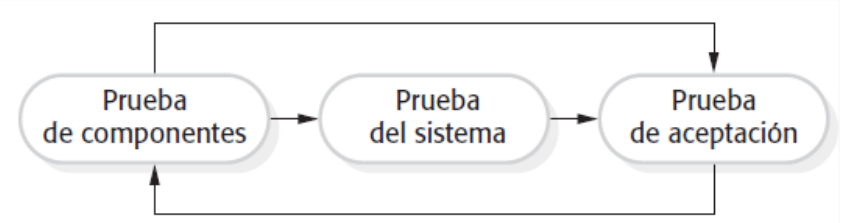
Especificación | Desarrollo | Validación | Evolución

- Los métodos estructurados se desarrollan con UML orientado a objetos .
- Las herramientas de desarrollo de software se usan para generar un programa esqueleto a partir del diseño.
- Por lo general, los programadores realizan algunas pruebas del código que desarrollaron. Suele revelar con frecuencia errores que requieren una depuración (debugging).
- La **prueba de defectos** y la **depuración** son procesos diferentes.
  - La primera establece la existencia de defectos, en tanto que la segunda se dedica a localizar y corregir dichos defectos.

T2. Procesos de software. 2.2 Actividades del proceso. 2.2.3 Validación de software

Universidad Francisco de Vitoria UFV Madrid

- La validación de software o, más generalmente, su verificación y validación (V&V), se crea para mostrar que un sistema cumple tanto con sus **especificaciones** como con las **expectativas** del cliente.
- Las pruebas del programa, donde el sistema se ejecuta a través de datos de prueba simulados, son la principal técnica de validación.
- Los sistemas no deben ponerse a prueba como una unidad monolítica sino por separado: componentes, sistema y con datos de cliente.
- El proceso es iterativo, con información retroalimentada desde etapas posteriores hasta las partes iniciales del proceso.
- Las etapas en el proceso de pruebas son:
  - **Prueba de desarrollo**: los desarrolladores del sistema ponen a prueba los componentes que constituyen el sistema. Cada componente se prueba de manera independiente. También se usan herramientas de automatización de pruebas (JUnit), definidas previamente y que se ejecutan de forma automática.



T2. Procesos de software. 2.2 Actividades del proceso. 2.2.3 Validación de software

Universidad Francisco de Vitoria UFV Madrid

Especificación | Desarrollo | Validación | Evolución

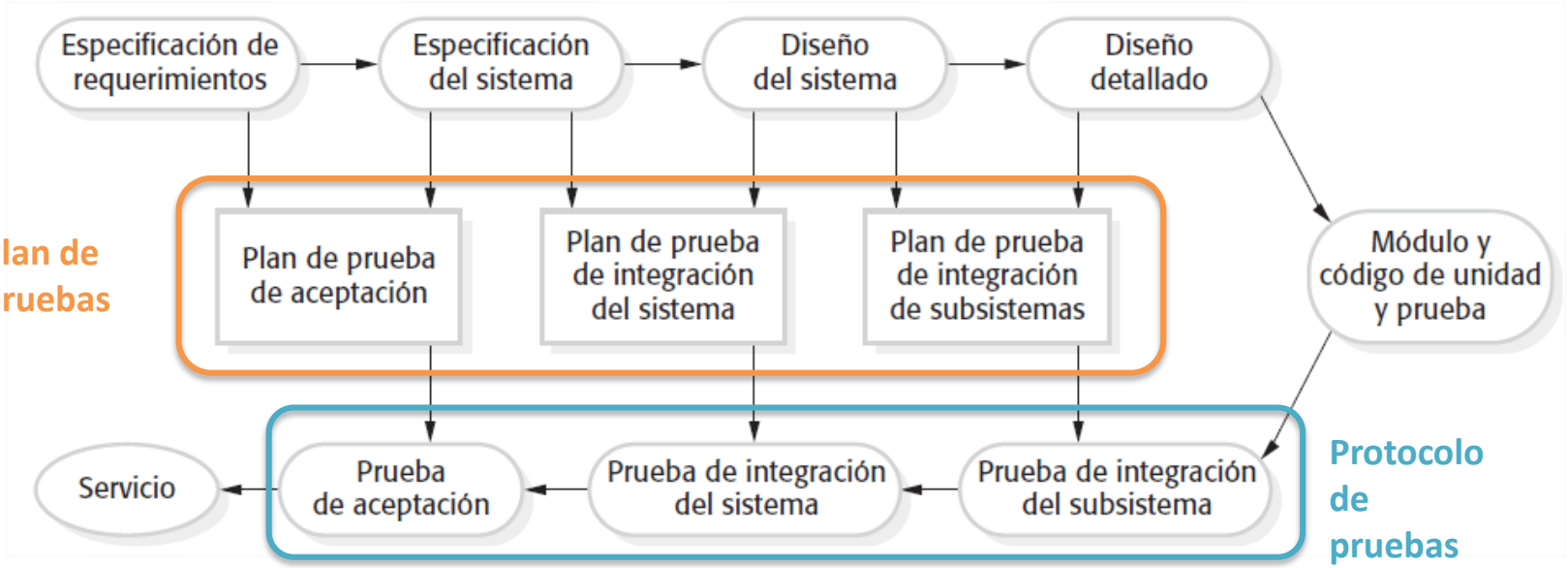
- **Pruebas del sistema. Pruebas FAT** - Factory Aceptance Test: los componentes del sistema se integran para **crear un sistema completo**. Finalidad: descubrir errores que resulten de interacciones no anticipadas entre componentes y problemas de interfaz de componente + comprobar sistema cubre sus requerimientos funcionales y no funcionales, y poner a prueba las propiedades del sistema.
- **Pruebas de aceptación. Pruebas SAT** - Site Acceptance Test: El sistema se pone a prueba con **datos suministrados por el cliente** del sistema, **en vez de datos de prueba simulados** .
  - Revelan los errores y las omisiones en la definición de requerimientos del sistema.
  - Revelan problemas de requerimientos funcionales definidos, donde las instalaciones del sistema no cumplan las necesidades del usuario o cuando haya mal rendimiento.

- Procesos de desarrollo y de pruebas de componentes están entrelazados. Los programadores construyen sus propios datos de prueba y experimentan el código de manera incremental conforme lo desarrollan
- El programador es el más indicado para generar casos de prueba.
- En **programación extrema**, las **pruebas se desarrollan junto con los requerimientos** antes de comenzar el desarrollo.

T2. Procesos de software. 2.2 Actividades del proceso. 2.2.3 Validación de software

Universidad Francisco de Vitoria UFV Madrid

- Se vinculan los planes de prueba entre las actividades de pruebas y desarrollo. A esto se le conoce en ocasiones como **modelo V de desarrollo.**
- **Pruebas de aceptación**: se suelen llamar pruebas Alfa, y cuando un sistema se marca como producto de software, pruebas Beta

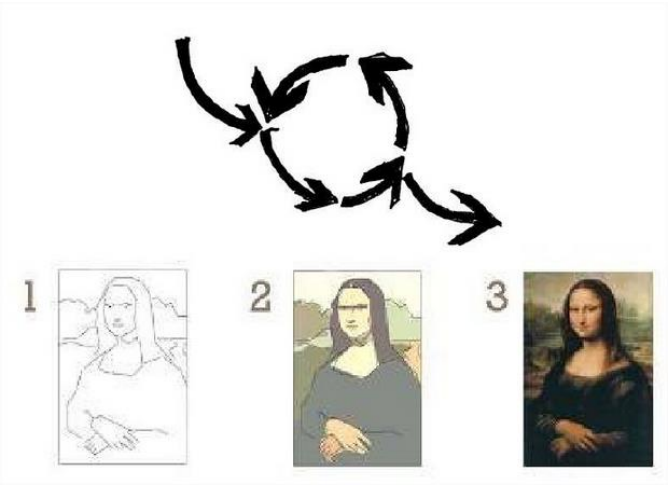


T2. Procesos de software. 2.2 Actividades del proceso. 2.2.4 Evolución del software

Universidad Francisco de Vitoria UFV Madrid

Especificación | Desarrollo | Validación | Evolución

- La **flexibilidad** de los sistemas de software es una de las razones principales por las que cada vez más software se incorpora en los sistemas grandes y complejos.
- En la fabricación de **hardware**, resulta **muy costoso hacer cambios** en su diseño.
- Durante o después del desarrollo del sistema, pueden hacerse **cambios al software**.
  - Incluso los cambios mayores son todavía **más baratos** que los correspondientes cambios al hardware del sistema.
- No obstante, consideran en ocasiones **el mantenimiento del software como insulso y poco interesante.**
- La distinción entre desarrollo y mantenimiento es cada vez más irrelevante.
  - Es muy difícil que cualquier sistema de software sea un sistema completamente nuevo, y tiene mucho más sentido ver el desarrollo y el mantenimiento como un continuo. **Desarrollo y mantenimiento como un todo: EVOLUCIÓN**.



T2. Procesos de software. 2.3 Cómo enfrentar el cambio

Universidad Francisco de Vitoria UFV Madrid

- El cambio es inevitable en todos los grandes proyectos de software.
  - Los requerimientos del sistema varían conforme la empresa procura que el sistema responda a presiones externas y se modifican las prioridades administrativas.
  - A medida que se ponen a disposición nuevas tecnologías, surgen nuevas posibilidades de diseño e implementación.
- Por tanto, cualquiera que sea el modelo del proceso de software utilizado, es esencial que **ajuste los cambios al software a desarrollar**.
- El cambio se agrega a los **costes del desarrollo** de software debido a que, por lo general, significa que **el trabajo ya terminado debe volver a realizarse**. A esto se le llama **REHACER**.

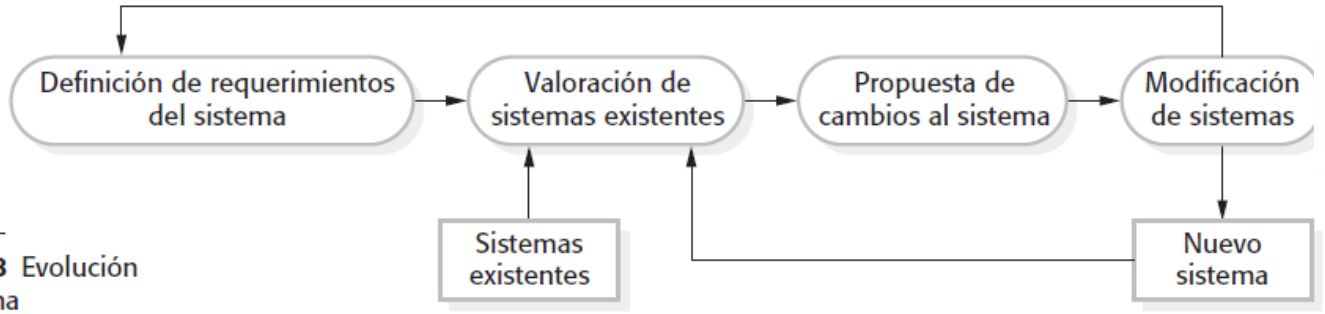


Figura 2.8 Evolución del sistema

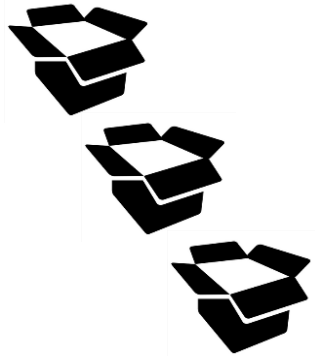
T2. Procesos de software. 2.3 Cómo enfrentar el cambio

Universidad Francisco de Vitoria UFV Madrid

- Existen dos enfoques relacionados que se usan para reducir los costes del rehacer:
  1. **Evitar el cambio**, donde el proceso de software incluye actividades que anticipan cambios posibles antes de requerirse la labor significativa de rehacer.
     - Por ejemplo, puede desarrollarse un sistema **prototipo**
  2. **Tolerancia al cambio**, donde el proceso se diseña de modo que los cambios se ajusten con un coste relativamente bajo. Esto comprende algunas formas del **desarrollo incremental**.

- Dos formas de enfrentar el cambio y los requerimientos cambiantes del sistema:
  1. **Prototipo de sistema**, donde rápidamente se desarrolla una versión del sistema o una parte del mismo, para comprobar los requerimientos del cliente y la factibilidad de algunas decisiones de diseño.
  2. **Entrega incremental**, donde los incrementos del sistema se entregan al cliente para su comentario y experimentación. Esto apoya tanto al hecho de evitar el cambio como a tolerar el cambio

- La noción de **refactorización**, restructuración de código sin modificar el comportamiento del sistema. Es la mejora de la estructura y organización de un programa, es también un mecanismo importante que apoya la tolerancia al cambio.

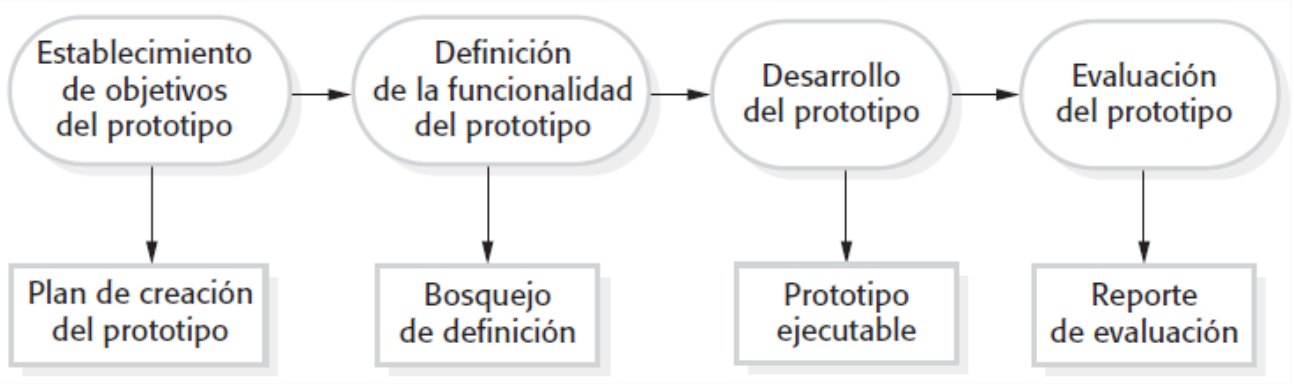


T2. Procesos de software. 2.3 Cómo enfrentar el cambio. 2.3.1 Creación del prototipo

Universidad Francisco de Vitoria UFV Madrid

2.3.1 Creación del prototipo

- Un **prototipo** es una versión inicial de un sistema de software que se usa para demostrar conceptos, tratar opciones de diseño y encontrar más sobre el problema y sus posibles soluciones.
- Un prototipo de software se usa en un proceso de desarrollo de software para contribuir a anticipar los cambios que se requieran:
    1. En el proceso de ingeniería de requerimientos, un prototipo ayuda con la selección y validación de requerimientos del sistema.
    2. En el proceso de diseño de sistemas, un prototipo sirve para buscar soluciones específicas de software y apoyar el diseño de interfaces del usuario.
- Los prototipos son mejores que las descripciones textuales o los diseños en papel.

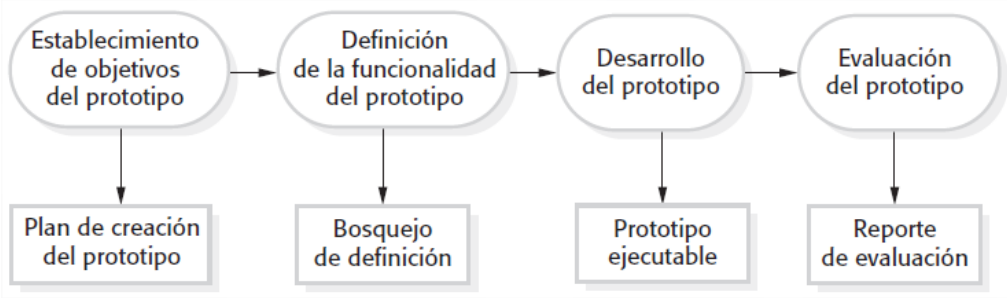


T2. Procesos de software. 2.3 Cómo enfrentar el cambio. 2.3.1 Creación del prototipo

Universidad Francisco de Vitoria UFV Madrid

2.3.1 Creación del prototipo

- La creación rápida de prototipos con la participación del usuario final es la única **forma sensible para desarrollar interfaces de usuario gráficas** para sistemas de software.
- **El mismo prototipo no puede cumplir con todos los objetivos**, ya que si éstos quedan sin especificar, los administradores o usuarios finales quizá malinterpreten la función del prototipo.
- La siguiente etapa del proceso consiste en **decidir qué poner** y, algo quizá más importante, qué dejar fuera del sistema de prototipo. Objetivo : reducir costes.
- El manejo y la gestión de errores pueden ignorarse, a menos que el objetivo del prototipo sea establecer una interfaz de usuario.
- La etapa final del proceso es la **evaluación del prototipo**. Hay que tomar provisiones durante esta etapa para la capacitación del usuario y usar los objetivos del prototipo para derivar un plan de evaluación.
- Un problema general con la creación de prototipos es que quizás el prototipo **no se utilice necesariamente en la misma forma que el sistema final.**

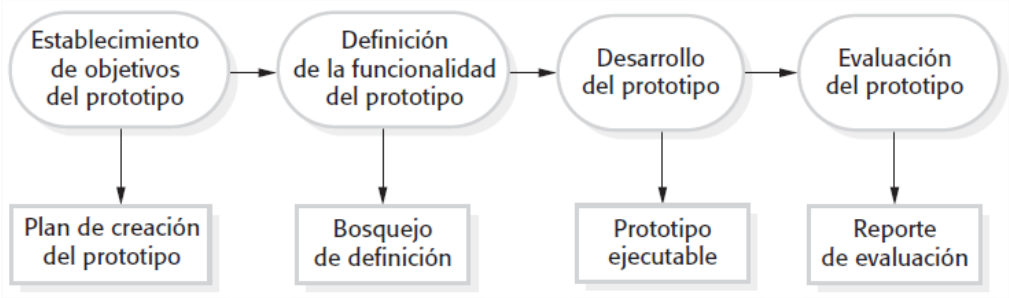


T2. Procesos de software. 2.3 Cómo enfrentar el cambio. 2.3.1 Creación del prototipo

Universidad Francisco de Vitoria UFV Madrid

2.3.1 Creación del prototipo

- En ocasiones, los desarrolladores están **presionados** por los administradores para **entregar prototipos** desechables, sobre todo cuando existen demoras en la entrega de la versión final del software.
  1. Puede ser imposible corregir el prototipo para cubrir requerimientos no funcionales, como los requerimientos de rendimiento, seguridad, robustez y fiabilidad.
  2. El cambio rápido durante el desarrollo significa claramente que el prototipo **no está documentado.**
  3. Probablemente los cambios realizados durante el desarrollo de prototipos **degradará la estructura** del sistema, y este último será difícil y costoso de mantener.
  4. Por lo general, durante el desarrollo de prototipos se hacen **más flexibles los estándares de calidad** de la organización.



T2. Procesos de software. 2.3 Cómo enfrentar el cambio. 2.3.2 Entrega incremental

Universidad Francisco de Vitoria UFV Madrid

2.3.2 Entrega incremental

- La entrega incremental es un enfoque al desarrollo de software donde algunos de los incrementos diseñados se entregan al cliente y se implementan para usarse en un entorno operacional.
- La asignación de servicios por incrementos depende de la prioridad del servicio, donde los servicios de más alta prioridad se implementan y entregan primero.
- La entrega incremental tiene algunas ventajas:
  1. Los clientes pueden usar los primeros incrementos como prototipos y adquirir experiencia.
  2. Los clientes deben esperar hasta la entrega completa del sistema, antes de ganar valor del mismo.
  3. El proceso mantiene los beneficios del desarrollo incremental en cuanto a que debe ser relativamente sencillo incorporar cambios al sistema.
  4. Puesto que primero se entregan los servicios de mayor prioridad y luego se integran los incrementos, los servicios de sistema más importantes reciben mayores pruebas.

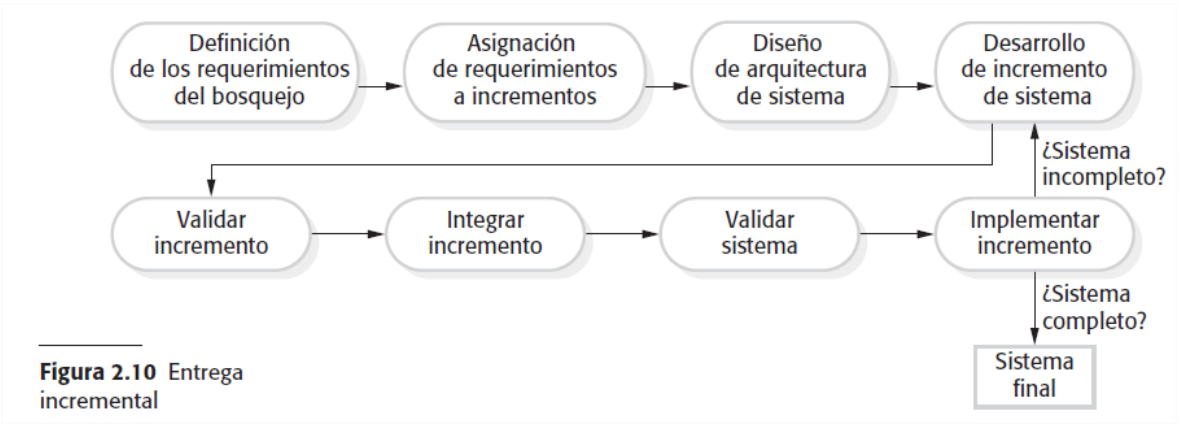


Figura 2.10 Entrega incremental

T2. Procesos de software. 2.3 Cómo enfrentar el cambio. 2.3.2 Entrega incremental

Universidad Francisco de Vitoria UFV Madrid

- Sin embargo, existen problemas con la entrega incremental:
  - Los requerimientos no están definidos con detalle sino hasta que se implementa un incremento, resulta **difícil identificar recursos comunes** que necesiten todos los incrementos.
  - El desarrollo iterativo resulta complicado cuando se diseña un **sistema de reemplazo**. Los usuarios requieren de toda la funcionalidad del sistema antiguo, ya que es común que no deseen experimentar con un nuevo sistema incompleto.
  - En el enfoque incremental, no hay especificación completa del sistema, sino hasta que se define el incremento final. Esto requiere una **nueva forma de contrato** que los grandes clientes, como las agencias gubernamentales, encontrarían difícil de adoptar.
- Existen algunos tipos de sistema donde el desarrollo incremental y la entrega **no son el mejor enfoque**.
  - Hay **sistemas muy grandes, complejos, que siguen un procedimiento muy detallado (requerimientos legales, o de seguridad)** El desarrollo incluye algunos sistemas embebidos o sistemas antiguos con los que es difícil integrarse, o incluso, el software depende del propio hardware existente.
  - Algunos **sistemas críticos** donde todos los requerimientos tienen que analizarse para comprobar las interacciones que comprometan la seguridad o protección del sistema.

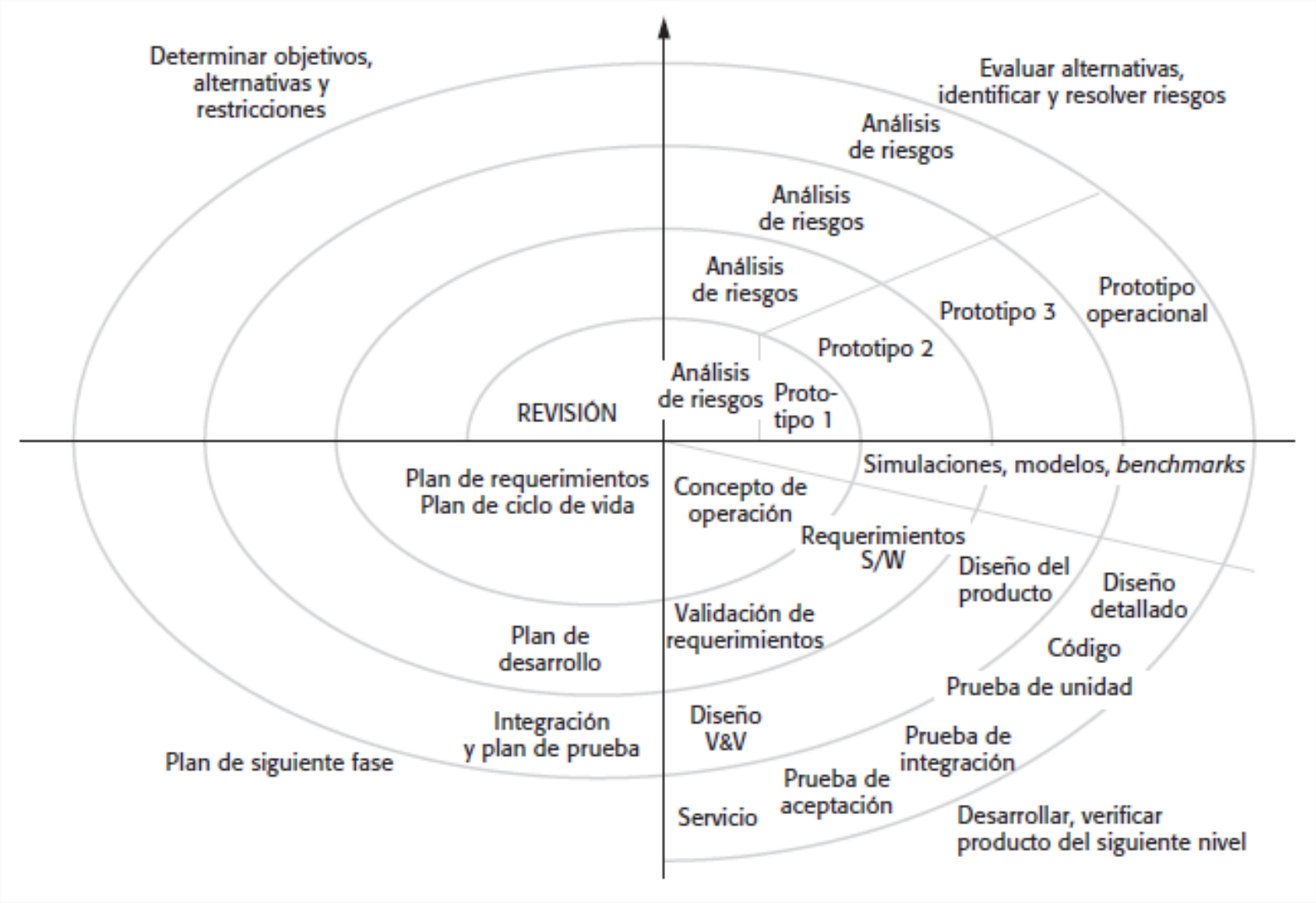
T2. Procesos de software. 2.3 Cómo enfrentar el cambio. 2.3.3 Modelo en espiral de Boehm

Universidad Francisco de Vitoria UFV Madrid

- Boehm (1988) propuso un marco del **proceso de software dirigido por el riesgo** (el **modelo en espiral**)
- El modelo en espiral combina el evitar el cambio con la tolerancia al cambio. Lo anterior supone que los cambios son resultado de riesgos del proyecto e incluye actividades de gestión de riesgos explícitas para reducir tales riesgos.
  1. **Establecimiento de objetivos** Se definen objetivos específicos para dicha fase del proyecto. Se identifican restricciones en el proceso y el producto, y se traza un plan de gestión detallado.
  2. **Valoración y reducción del riesgo** En cada uno de los riesgos identificados del proyecto, se realiza un análisis minucioso.
  3. **Desarrollo y validación** Después de una evaluación del riesgo, se elige un modelo de desarrollo para el sistema.
  4. **Planteamiento**. El proyecto se revisa y se toma una decisión sobre si hay que continuar con otro ciclo de la espiral.
- La diferencia principal entre el modelo en espiral con otros modelos de proceso de software es su reconocimiento explícito del riesgo. Un ciclo de la espiral comienza por elaborar objetivos como rendimiento y funcionalidad.
  - Se numeran formas alternativas de alcanzar dichos objetivos y de lidiar con las restricciones en cada uno de ellos. Cada alternativa se valora contra cada objetivo y se identifican las fuentes de riesgo del proyecto.

T2. Procesos de software. 2.3 Cómo enfrentar el cambio. 2.3.3 Modelo en espiral de Boehm

Universidad Francisco de Vitoria UFV Madrid



T2. Procesos de software. Puntos clave

Universidad Francisco de Vitoria UFV Madrid

- ✓ Los **procesos de software** son actividades implicadas en la producción de un sistema de software. Los **modelos de proceso** de software consisten en representaciones abstractas de dichos procesos.

- ✓ Los **modelos de proceso general** describen la organización de los procesos de software. Los ejemplos de estos modelos generales incluyen el modelo en **cascada**, el desarrollo **incremental** y el desarrollo **orientado a la reutilización**.

- ✓ La **ingeniería de requerimientos** es el proceso de desarrollo de una especificación de software. Las **especificaciones** tienen la intención de comunicar las necesidades de sistema del cliente a los desarrolladores del sistema.

- ✓ Los procesos de **diseño e implementación** tratan de transformar una especificación de requerimientos en un sistema de software ejecutable. Pueden usarse métodos de diseño sistemáticos como parte de esta transformación.

T2. Procesos de software. Puntos clave

Universidad Francisco de Vitoria UFV Madrid

- ✓ La **validación del software** es el proceso de comprobar que el sistema se conforma a su especificación y que satisface las **necesidades reales** de los usuarios del sistema.

- ✓ La **evolución del software** tiene lugar cuando cambian los sistemas de software existentes para satisfacer **nuevos requerimientos**. Los cambios son continuos y el software debe evolucionar para seguir siendo útil.

- ✓ Los procesos deben incluir **actividades para lidiar con el cambio**. Esto puede implicar una fase de **creación de prototipos** que ayude a evitar malas decisiones sobre los requerimientos y el diseño. Los procesos pueden estructurarse para desarrollo y entrega iterativos, de forma que los cambios se realicen sin perturbar al sistema como un todo.