# Metodoloxías de Desenvolvemento

## Metodologías Clásicas

---

Javier Parapar
javierparapar@udc.es

Revised: Pedro Cabalar

Updated: 11 de octubre de 2017

Information Retrieval Lab
Computer Science Department
University of A Coruña

# Organización del desarrollo

- Los **componentes** de cualquier entorno de desarrollo afectan al **modo** en el que se construye el software
- La **metodología** que se aplique influye tanto en la **productividad** como en la **calidad** del producto.
- Los procedimientos de **gestión** enfocan y regulan el empleo de las **técnicas** y **herramientas** más **apropiadas** para las arquitectura que ha de definirse
- La determinación de la **metodología** a emplear, teniendo en cuenta las características del entorno de desarrollo, es **esencial** para el buen fin de los proyectos

# Organización del desarrollo

En la **selección** de la metodología hay dos **alternativas**:

- Desarrollar una **metodología *ad hoc*** para el entorno y el proyecto, teniendo en cuenta la variedad de métodos, técnicas y herramientas, y evaluando de todas ellas las más oportunas y las que posibilitan establecer una pauta metódica
- Evaluar la propuestas de **metodología existentes**, tratando de determinar cual y porqué una de ellas es la más apropiada

# Organización del desarrollo

Al seleccionar una metodología ha de tenerse en cuenta el modo y rigor con el contempla las siguientes cuestiones:

- **Cobertura** del ciclo de desarrollo, y el modo en el que se puede aplicar en cada fase del mismo
- **Reglas** y patrones predefinidos
- **Políticas de verificación** de productos intermedios y finales, y política de liberación y gestión de versiones
- Técnicas de **planificación y control** de tiempos y costes
- Técnicas de **gestión** y asignación de **recursos**, del trabajo en equipo, y de valoración de rendimientos
- Garantías de **calidad**

# Organización del desarrollo

Al seleccionar una metodología ha de tenerse en cuenta el modo y rigor con el contempla las siguientes cuestiones:

- Necesidades de **formación** y posibilidades de rápida asimilación
- Documentación y **comunicación eficiente**
- Capitalización de la **experiencia** y empleo generalizable para otros proyectos
- **Herramientas** de apoyo y de generación de código
- Métodos de **mantenimiento** y actualización
- **Estructuración** de los productos y preparación de componentes re-utilizables

# Generaciones

- Desarrollo **Convencional**
- Desarrollo **Estructurado**
- Desarrollo **Orientado a Objetos**

# Desarrollo convencional

- Los **cambios** en la organización, la gestión, la estrategia y los objetivos afectan muy **negativamente** al proceso de desarrollo
- La continuidad, evolución y adaptación del software depende de **individualidades**
- **No** es posible establecer un **control** suficiente del proyecto ni determinar una **planificación**
- La **documentación** es **escasa** y singular
- Los resultados finales son **impredecibles**, en alcance, idoneidad y calidad

# Desarrollo estructurado

- La llegada de la definición de **jerarquías y de la estructuración** de las labores de construcción de software tuvo la siguiente progresión
  - Se inicia con Programación Estructurada: **normaliza** el software
  - Se complementa muy pronto con el **Diseño** Estructurado. que organiza las **especificaciones** técnicas.
  - Se cierra con el **Análisis** estructurado, que se extiende a los **requisitos** y **especificaciones** funcionales, las organiza por partes, controla y minimiza la redundancia y las explicita mediante **modelos gráficos** muy expresivos
- Surgen metodologías para el desarrollo ***top-down***, trabajando la abstracción desde le nivel más alto de **comprensión** general del problema, hasta el más sencillo de síntesis y **codificación** de la lógica
- En estas metodologías se representan los **procesos** y **flujos** de procesos, la **estructura** de datos, y su combinación

# Desarrollo orientado a objetos

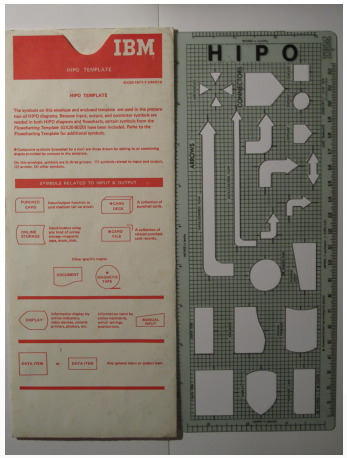
- El paradigma de la **Orientanción a Objetos** llega a la construcción del software de la forma siguiente:
  - Se inicia también con la Programación Orientada a Objetos, que facilita la **re-utilización de componentes** de software y **encapsula** datos y procesos
  - Se complementa muy pronto, utilizando los conceptos del paradigma en el **Análisis y Diseño**, donde adquieren mayor importancia y garantizan el acercamiento de los **modelos** y sistemas de información a los sistemas y contextos **reales**
  - Se completa con la propuestas de un **Lenguaje Unificado de Modelado**, que pronto supone un estándar y adquiere un uso universal
- Es el origen de múltiples herramientas y metodologías que responden a diversas filosofías y orientaciones (RUP, SCRUM, XP y otras)

# Clasificación

- Estructuradas
  - Orientadas a Procesos
  - Orientadas a Datos
    - Jerárquicas
    - No jerárquicas
- Orientadas a Objetos
  - Iniciales (Puras)
    - Modelo estático
    - Modelo dinámico
    - Modelo de procesos
  - Evolutivas
    - Basadas en UP (encorsetadas)
    - Ágiles
    - Extremas
- Sistemas en tiempo real
- Sistemas globales.

# Estructuradas: Orientadas a Procesos

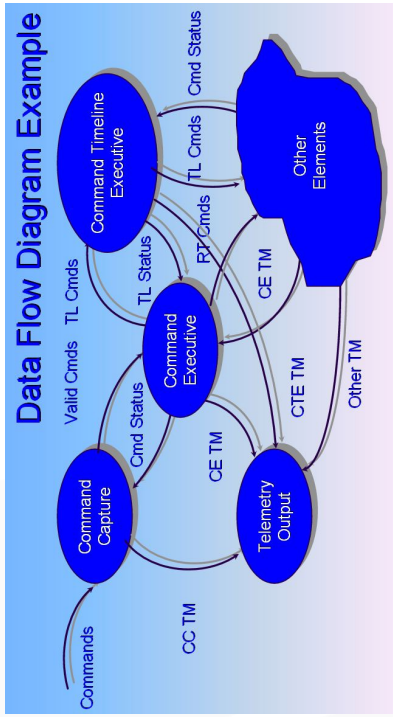
- Se fundan en la abstracción de modelado básica, de **entrada**, **proceso** y **salida** de un sistema (por ejemplo los diagramas HIPO de IBM)
- El fundamento del método es la **descomposición descendente y funcional**, para definir requerimientos, requisitos y especificaciones del sistema
- Se apoya en técnicas gráficas y **diagramas** que respetan la estructura **jerárquica** para expresar el desglose tanto de los procesos como, en cierta manera, de los datos
- Todo ello da lugar al concepto de **Especificación Estructurada**, que se describe como un modelo fraccionado y descendente



# Estructuradas: Orientadas a Procesos

La Especificación estructurada contiene

- **DFD** (Diagramas de Flujo de Datos): los procesos y los datos que entre ellos se intercambian, con **diferentes niveles** detalle. Los procesos complejos se refinan **descomponiéndolos** en procesos más simples, pero se mantiene la **trazabilidad**
- **Diccionarios de Datos**: se declaran en ellos todos los datos usados en los flujos de información de los DFDss. También se pueden **jerarquizar** por su nivel de agregación (diccionarios locales, superdiccionarios etc.)
- **Especificaciones de Proceso**: detallan el alcance, la lógica, los condicionantes y cualquier otra cosa que permita describir de forma clara y explícita los que sucede **en cada *burbuja*** de un DFD



# Estructuradas: Orientadas a Procesos

Según el nivel de abstracción (de más a menos) hablaremos de:

- **Modelo Conceptual** es el más cercano a la representación del mundo real. Describe los conceptos **independientemente** la plataforma o el **paradigma** de modelado que se utilice después. Estructura la percepción del problema y relaciones entre los elementos que formarán el sistema solución
- **Modelo Lógico** que se deriva del refinamiento y depuración del Modelo Conceptual, y se **adapta al paradigma seleccionado**, por ejemplo, representaciones de las BBDD con el Modelo de E/R. El modelo lógico es la base y contiene los elementos suficientes para conformar y declarar un Modelo Físico
- **Modelo Físico** = derivación del Modelo Lógico considerando los detalles de **implementación**, y depende de la plataforma elegida, p. ej., el gestor BBDD a emplear y sus características para la declaración de espacios, tablas, atributos etc.

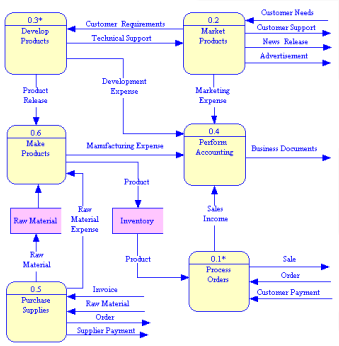
En **DeMarco** se proponen las siguientes fases:

1. **Estudio del entorno** físico del sistema actual, considerando sus procedimientos y representándolo con un conjunto organizado de **DFD**s, que conforman el Modelo Físico Actual
2. Deducir y **elaborar el Modelo Lógico Actual**, también como un conjunto de DFDs, e incluso se esbozan diccionarios de datos, y especificaciones de proceso, pero se obvian aspectos físicos y se incide en los descriptivos
3. Derivar un **nuevo Modelo Lógico**, teniendo en cuenta nuevos **requisitos** y necesidades. Este Modelo consta de DFDs, Diccionarios de Datos y especificaciones de Proceso, suficientemente explícitas

# Estructuradas: Orientadas a Procesos ▷ Gane y Sarson

Gane-Sarson (IBM) es parecida a de DeMarco pero los contenidos de la persistencia o almacenes de datos se describen en 3FN

1. Construir directamente el Modelo Lógico (**DFD Lógico Actual**)
2. Elaborar el **Modelo Lógico** del **nuevo** sistema, definiendo especificaciones estructuradas y organizando en Tercera Forma Normal las entidades de datos. Se pueden plantear alternativas
3. Seleccionar el **Modelo Lógico Final**, en función de la justificación y discusión de las alternativas
4. Crear el **Modelo Físico** correspondiente con el Modelo Lógico
5. Recopilar la **documentación** necesaria y especificaciones



Yourdon/Constantine es otra alterantiva presentada en los 80

1. Elaborar los DFDs del sistema
2. Realizar el **Diagrama de Estructuras**, basándose en los DFDs, efectuando los análisis de
   - **Transformación**:
     entrada ⇒ proceso (transformación) ⇒ salida
   - **Transacción**:
     entrada ⇒ centro de transacción) ⇒ acción(es)
3. Evaluar el diseños, midiendo la calidad de la estructura modular por el estudio del **acoplamiento** y la **cohesión**
4. Crear los **Cuadernos de Carga** (Unidades Físicas: programas y módulos), dividiendo el diseño según las características y organización de la implementación

# Estructuradas: Orientadas a Datos

Aplican el modelo básico de **Input–Process–Output** (IPO), por lo que se orienta a la estructuración de las entrada salida:

1. Se **define** la estructura de **datos**
2. A partir de esta **estructura** se **particiona** y acota la división en componentes de procedimiento
3. Se elabora una **estructura de control** *factorizada* y jerárquica para los programas
4. Se **mezcla** esta estructura de datos con la estructura jerárquica de los programas
5. Se ordena la **lógica de los procedimientos** para que se ajuste a esta estructura

El Diseño Lógico debe preceder y estar claramente **separado** del Diseño Físico, evitando influencias de la implementación

# Estructuradas: Orientadas a Datos

Consta de 4 etapas:

1. **Planificación** donde se establece una estructura de la información y una estrategia que soporte los objetivos de la organización
2. **Análisis** donde se profundiza en el entendimiento de las diferentes áreas de negocio y se determinan los requisitos del sistema, basándose en esta comprensión de la empresa y en la calificación y establecimiento de prioridades
3. **Diseño** donde se define el sistema en base a los servicios que desea el usuario y condicionan su funcionamiento, y se asegura que sea viable y alcanzable con la tecnología a emplear
4. **Construcción** donde se implementa y despliega un sistema que satisfaga lo descrito en las etapas anteriores

# Orientadas a Objetos

Existen diferencias fundamentales en la OO

- Las metodologías anteriores, estudian los sistemas desde el marco **funcional**, determinando las **tareas** que han de soportar, y se avanza en la definición descomponiendo sucesivamente estas tareas para determinar los módulos del sistema, las aplicaciones y componentes de las mismas
- Las metodologías de la OO tienen la pretensión esencial de capturar la realidad entendiendo los **objetos del dominio del problema**, y desarrollar los modelos basándose en la conexión e interactuación o colaboración, de tales objetos

# Orientadas a Objetos

Existen diferencias fundamentales en la OO

- En las metodologías tradicionales se produce de hecho una **separación** excesiva entre **función/proceso y datos**
- En las metodologías de la OO los **comportamientos** y **conocimientos** se encapsulan como **características** de cada **objeto**, que participan así en una *sociedad de objetos* significativos en el dominio del problema, que colaboran y que son una imagen o representación muy precisa de los objetos y actores del mundo real

# Tiempo Real

> *Se denomina **Sistema en Tiempo Real**, aquel que gestiona y controla un ambiente en el que se ejecuta, procesando los datos de entrada y devolviendo los resultados con una inmediatez suficiente como para influir en el comportamiento de dicho ambiente en ese momento*

- Las metodologías para sistemas de tiempo real se orientan fundamentalmente al **control** y este control está guiado por eventos externos
- Características:
  - Existe **comunicación** y cohesión entre tareas/tratamientos
  - Son **concurrentes**
  - Se establecen **prioridades** de proceso
  - Posibilitan el acceso **simultáneo** a aquellos datos de la persistencia que se definan como comunes

# Tiempo Real

Así pues para definir los requisitos de estos sistemas es necesario considerar:

- El manejo de las **interrupciones**
- La comunicación y **sincronización** entre tareas
- La gestión de procesos **concurrentes**
- La oportunidad de las **respuestas** a los eventos externos
- La concepción discreta o continua de los **datos**

Es necesario diferenciar entre los Sistemas en Tiempo Real y los Sistemas en Línea:

- Los actores que interactúan con los sistemas on-line son personas
- Los actores que interactúan con los Sistemas en Tiempo Real, son personas y componentes físicos, repositorios, y otros subsistemas del exterior

# Globalizadas

- Del mismo modo que las metodologías OO están evolucionando para abarcar el desarrollo tanto de Sistemas en Tiempo Real como en línea, también han de adaptarse a todas las características y artefactos necesarios para asumir la **arquitectura y despliegues de la globalización de servicios**
- Por otra parte, y como opinión, quizás sea necesario en algunos casos pensar en **metodologías mixtas**, que combinen diferentes orientaciones y se apliquen según los subsistemas a construir
- También existen propuestas de definir, sobre principios generales y usando estándares, lenguajes de modelado y herramientas de planificación y gestión, **metodologías *ad hoc***
- Por ultimo, es evidente que las metodologías también tendrán que contemplar con más detalle las características derivadas de la **socialización de sistemas**, los **BPMs**, la **integración**, y la implantación de herramientas de **BI**

# Diseño Incremental

- Los modelos de desarrollo en **cascada** presentaban como desventajas principales que no contempla nada más que una secuencia **lineal**, y además el proceso de desarrollo es **lento** y el software no se puede **liberar** hasta que se supere la última prueba
- Para solucionar estos problemas surge el modelo de desarrollo incremental, cuya base es el avance **iterativo**, con resultados **entregables**, y de modo que la iteraciones no sólo se **consoliden**, sino que sirvan como experiencia y **base** de la iteración siguiente

# Diseño Incremental

- En contraposición los modelos de desarrollo en **espiral** compensan estas limitaciones dado que, a medida que se avanza, el equipo de desarrollo y el cliente **identifican** mejor los **riesgos** y **reaccionan** tempranamente a ellos
- Así se puede **comenzar** con una implementación **simple** de los requerimientos del sistema y en las posteriores versiones resultantes de las iteraciones, ir **incorporando** la **mejora** y las nuevas funcionalidades hasta lograr el sistema final

# Etapas comunes

- En el modelo incremental se contemplan:
  - La etapa de **inicialización**
  - La lista de **control** del proyecto
  - La etapa de **iteración**
- El objetivo de la etapa de **inicialización** es crear una **versión preliminar**, con los aspectos esenciales del sistema que el usuario pueda manejar y que sirva como fuente para retro-alimentar el proceso. El **alcance** tiene que **acotarse** con precisión para que el resultado pueda ser comprendido e implementado fácilmente
- El objetivo de la lista de **control** del proyecto es **organizar** dinámicamente todas las **tareas** que deben ser abordadas y que contemplan tanto los rediseños necesarios como las nuevas funcionalidades. Esta lista se **actualiza** constantemente, con los resultados progresivos del análisis

# Etapas comunes

- El objetivo de la etapa de **iteración** es diseñar, rediseñar o implementar **una tarea** de la lista de control, y analizando la **versión actual** del sistema
- Las iteraciones han de ser **acotadas**, **sencillas**, directas y **modulares** para poder abordar el trabajo de la etapa como una tarea alcanzable en ella, y que esté bien y completamente **definida**
- Una iteración **parte** de las funcionalidades ya logradas
- El **usuario** ha de estar involucrado

# Análisis e implementación

1. Ha de considerarse la necesidad de **rediseñar** o recodificar ante cualquier **dificultad** que surja en el desarrollo de una modificación, desde el diseño de la misma hasta la prueba
2. La **modificaciones** deben estar bien encuadradas en los **módulos aislados** o independientes. En caso contrario quizás haga falta revisar el diseño
3. Si para una modificación es necesario cambiar tablas en las BBDD, estos **cambios** han de ser **fáciles de incorporar**. Si no fuera así, quizás sea necesario revisar el diseño
4. Según se **avanza** en la iteraciones y versiones, las modificaciones han de ofrecer **menos dificultades**. Si esto no ocurriera puede haber debilidades serias en el análisis y el diseño, y un exceso de parches en la codificación

# Análisis e implementación

5. Los **parches** sólo se deben mantener en una iteración, cuando se decide incorporarlos para evitar el rediseño inmediato en una fase de implementación. La iteración siguiente debería considerar su **eliminación**
6. Los resultados de la **implementación** deben ser **evaluados continuamente** para establecer el grado de ajuste del sistema a los requerimientos objetivo
7. Las **opiniones** del usuario deben ser **estudiadas** cuidadosamente

# Pros

- △ Los usuarios **no necesitan esperar** hasta la terminación completa del desarrollo para **utilizar** el sistema
- △ Con cada **entrega**, el usuario puede hacer explícitos **mejor** los **requisitos** que no tiene claros o están adecuadamente definidos
- △ El **riesgo** del proyecto se **gestiona** mucho mejor, ya que se acota por cada iteración incremental
- △ Las partes más importantes del sistema tienen asignada su **prioridad** para ser desarrolladas e implementadas en las primeras iteraciones, lo que mejora la secuencia de pruebas y minimiza los fallos y su impacto

# Cons

- La **interacción** continua y el **feedback** del usuario puede hacer más **lento** el avance. Esto es especialmente relevante cuando los desarrollos son largos y precisan muchos recursos
- Los **incrementos** para la secuencia de versiones, si no son suficientemente **sencillos**, y no aumentan adecuadamente la **funcionalidad**, no consiguen limitar el riesgo
- La ***trazabilidad*** de las versiones incrementales con los requisitos y requerimientos a veces es **difícil** de determinar
- A veces **no es trivial** determinar las partes genéricas o principales del sistema y **priorizar** su desarrollo
- La implicación de los **usuarios** finales tiene un **coste** operativo **añadido** para la compañía pues merma su rendimiento en las labores de su puesto