

TECNOLÓGICO NACIONAL CEICOM

CARRERA DE SISTEMAS INFORMÁTICO

G

**PRÁCTICA 02**

**CAPITULO 2**

**PARADIGMAS Y SOFTWARE**

**Materia:** Análisis y diseño de Sistemas I

**Elaborado por:** Guarachi Camacho Harold Cristian

**Docente:** Ing. Baltazar Llusco Ever Jaime

**Fecha:** 18 de junio de 2018

Cochabamba - Bolivia

**CAPITULO 2**

**PARADIGMAS Y SOFTWARE**

**METODOLOGÍA Y PARADIGMA**

Comencemos diciendo que el proceso de creación del software es cíclico. Es así como todo software tiene su propio siclo y cada proyecto, contenido en uno o varios procesos, también. Notemos en primera instancia que diferenciamos el software, el proceso y el proyecto. En algunos casos, proyecto y proceso de creación puede tener ejecutándose uno mas procesos que interactúan o no entre si.

El proceso de software es la vista de más alta abstracción bajo lo cual organizaremos las tareas de ingeniería para obtener un software de calidad. Los proyectos de desarrollo incluyen e estos procesos y también a otros que pueden tener que ver las tareas relacionadas, como por ejemplo las económicas y los arreglos de contratos con clientes y trabajadores.

**EL PROCESO DE SOFTWARE**

El proceso de software es el conjunto definido, limitado y coherente de actividades que permiten el desarrollo de software. Como sabemos destacado en el capitulo 1, el software posee cierto complejidad que aumenta con todo los requisitos que atrae aparejados (cumplir con calendarios, respetar costos, tratar con personas). A lo largo de la historia del desarrollo se han presentado cuatro fases características de todo proceso:

* Requerimientos
* Construcción
* Validación
* Evolución

Estas etapas representan el ciclo de vida de un proyecto y pueden ser ordenadas de acuerdo aun modelo. Un modelo es una representación simplificada de la realidad y en este caso es el marco de los pasos que hay que seguir para que el desarrollo alcance buen término. En resumen, es un proceso de software, el modelo nos aporta lo siguiente:

* Define las tareas necesarias para el correcto funcionamiento por etapa y en general.
* Describe y delimita las fases.
* Permite la ubicación en proyecto.
* Facilita la estimación de tiempos y costos.
* Ubica a los recursos en un momento dado y en una tarea determinada.

**Modelo en cascada**

Cuando nos acercamos por primera vez a los desarrollos se nos enseña el modelo en cascada. Este existe desde hace más de 35 años y sigue siendo el pilar para tener noción sobre cuales son las etapas que se requiere cumplir antes de tener un entregable de mínima funcionalidad.

Especificación

Esta etapa es comúnmente denominada recolección de requisitos o especificación de requisitos, aunque en algunos entornos se le llama ingeniería de sistema. En esta fase necesitamos definir el dominio del software que vamos a desarrollar, comprender sus fines, utilidad y limitaciones, al menos en grandes rasgos. Para esto. Recolectaremos información que debe ser fácil de estructurar y almacenar porque será fundamental a lo largo de todo el desarrollo y los problemas o falencias en esta etapa pueden llevar al fracaso.

**Análisis**

En esta etapa se determinan los objetivos y límites a partir de la información adquirida anteriormente, intentando profundizar antes de pasar al diseño. Se deben determinar las tareas que realizará el sistema y sus estructuras, procesos críticos y secundarios. Una vez que tengamos una idea acabada de objetivo, comenzaremos a diagramar el sistema, aplicando alguna de las metodologías de análisis que consideramos adecuado al proyecto en cuestión. A medida que adquirimos experiencia, la etapa de análisis tendrá menos componentes teóricos Porque estos serán reemplazados por la carga de conocimientos que le pueda aportar el equipo. Por sus propiedades, las tareas del Análisis son denominadas.

* Conceptualización: el objetivo es obtener una vista del alto nivel de abstracción se focaliza en el sistema y sus partes teniendo en cuenta los elementos externos.
* Análisis funcional: se trata de Identificar y describir los procesos internos del sistema una vez establecido el proceso lo seleccionamos y realizamos su refinamiento obteniendo los flujos de datos ingresantes y salientes.
* Análisis de condiciones: especificación de las restricciones de nuestro sistema. Las limitaciones pueden ser de tipo técnico, físico y económico o humano.
* Aplica la metodología: luego de las etapas anteriores aplicaremos algún proceso de diagramación. Esta creación para poder modelar el análisis realizado incorporar a toda la información recopilada anteriormente, como así también todas las restricciones y detalle que aparezcan en el ejecución de esta fase.
* Validación: si nuestro análisis correcto deberemos pasar a la etapa siguiente, pero para asegurarnos tenemos que someter este análisis ha una completa validación para evitar las inconsistencias, las ausencias de información y la programación de errores o defectos.

**Diseño**

En el diseño aplicamos prácticas de ingeniería para definir de forma técnica los procesos, elementos y datos del sistema. El diseño profundiza y completa la visión de software, pero además le agrega los componentes técnicos. Toda la etapa se realiza utilizando un tipo de modelo definido. El resultado de la fase de diseño será la base para el comienzo de la construcción de software. La documentación será entendida por los técnicos y estaremos dejando de lado la visión del alto nivel proporcionada para el cliente en las entrevistas y análisis, para detallar cada uno de los componentes o partes integrales. Según las definiciones tradicionales, en el diseño se materializan los requerimientos los requerimientos del cliente.

**Codificación**

En la fase de diseño se definieron una o más lenguas de programación en los cuales se escribirá el sistema. En esta etapa debemos destinar recursos para el sistema que ya ha sido analizado y diseñado pueda comenzar a ser una realidad. Las tareas de codificación son amplias y se realizan siguiendo cronogramas o patrones de trabajos establecidos. Cabe destacar que la generación de los modelos de datos y su estructura están inclinados en lo que denominamos codificación, aunque estas tareas no son siempre llevadas a cabo por el mismo equipo de trabajo.

**Pruebas**

La prueba del Software es un conjunto de prácticas que se llevan a cabo para localizar, identificar y eliminar errores y mejorar el producto. En general, cuando nos planteamos desarrollos de gran envergadura, un plan de pruebas adecuado puede ser la diferencia entre un software robusto y un software deficiente. Hay que remarcar que el objetivo de la prueba no sólo debe ser la localización del error sino que debe servir para anticiparse a los potenciales requerimientos futuros, asegurar la fiabilidad y presentar datos objetivos sobre el rendimiento y la calidad.

**Mantenimiento**

Luego de que el sistema ha sido probado e implementado y su funcionamiento es el correcto, debemos intentar realizar tareas que permitan prolongar su vida útil. Las operaciones de mantenimiento proponen lograr que el sistema mantenga el nivel de ejecución satisfactorio que obtuvo en las pruebas. Dentro del mantenimiento podemos observar:

* Monitoreo: es el conjunto de procesos que evalúan la calidad del control en el tiempo y permiten al sistema reaccionar en forma dinámica.
* Mantenimiento correctivo: es la reparación de errores en el momento en que estos aparecen.
* Mantenimiento preventivo: también denominado mantenimiento planificado. Consiste en la correcci6n de fallas de forma programada o la ejecución de tareas para que estas no aparezcan.

**Modelos evolutivos**

Los modelos evolutivos suponen la confrontación de lo desarrollado con los deseos de los dientes. Obteniendo la respuesta del cliente mas las mejoras que aporta el equipo de desarrollo se refina y valida el sistema a fin de alanzar los objetivos propuestos. Dentro de lo que se denominan modelos evolutivos; encontramos distintas variaciones aunque el principio es el acercar las posiciones entre el cliente y el desarrollador sobre en a estructura para mejorar en la siguiente aproximación.

**Prototipos e incremental**

EI modelo de prototipado es un modelo evolutivo que establece iteraciones cortas de forma cal de mostrarle los avances al cliente. A fin de poder acelerar el desarrollo para contrastar resultados, muchos de los prototipos son construidos haciendo hincapié en características visibles (pero secundarias para la arquitectura) como las interfaces, los reportes o la carga de datos. La principal diferencia que puede tener con otros modelos evolutivos es que se acepta desechar (prototipo desechable) todo el producto construido a fin de progresar en la siguiente iteración. Este use de descarte es una medida muy útil cuando se quiere mejorar la calidad. Una vez alcanzada A madurez suficiente en el conocimiento de problema, se descarta el sistema y se construye con enfoque en prácticas de calidad. Alguna de las ideas de los modelos evolutivos será luego utilizada en metodologías menos tradicionales. A pesar de que el modelo por prototipos presenta suficientes ventaja. (Mayor conocimiento del sistema por parte de dientes y usuarios y menos problemas de interacción de fases erróneas), uno de los mayores riesgos es que los desarrolladores (durante tanto tiempo concentrados en cuestiones secundarias) utilicen los prototipos como sistemas finales, ya sea para cumplir con un calendario de entregas o un plan de costos.

**Modelo en espiral**

Es un modelo mixto propuesto por Barry Boehm que conjuga las prácticas del modelo clásico junto a tendencia evolutivas. El resultado es un modelo con el cual se desarrollan versiones de software con mayor funcionalidad por iteración. Se definen cuatro actividades:

* Planificación: se entrevista al cliente ya se obtiene la información importante para el resto de la iteración
* Análisis de riesgo: se evalúa en base a la composición del equipo de desarrollo, del proyecto y los nuevos requisitos, las posibles amenazas y debilidades del proyecto.
* Construcción: fase propiamente dicha de desarrollo, construcción del sistema en base al análisis y los objetivos para esta etapa.
* Evaluación: el cliente realiza un relevamiento crítico del entregable y en base a esto se inicia una nueva iteración

**Modelo de desarrollo de componentes**

El modelo de desarrollo de componentes es un intento de mejora sobre el modelo de construcción en espiral. Se puede materializar en la práctica gracias a los paradigmas de análisis, diseño y programación orientada a objetos. Con sus características esenciales, los lenguajes orientados a objetos permiten generar componentes denominados clases. De esta manera se obtienen elementos que pueden ser estandarizados, evaluados y alterados para aplicarlos a un proyecto en particular.

**PARADIGMA DE DESARROLLO**

Comentamos, al iniciar el capitulo, las diferencias entre metodología y paradigma. El desarrollo estructurado y el orientado a objetos son paradigmas o modelos de desarrollo. A pesar de coexistir en el ambiente real, la programación estructurada ha perdido fuerza debido, principalmente, a que el paradigma de objetos, junto con las metodologías de desarrollo que lo utilizan o recomiendan, han podido solucionar las principales falencias del desarrollo estructurado en especial en productos industriales o con alta escalabilidad. Es conveniente conoces los principios del desarrollo orientado a objetos independistamente del lenguaje que se piense utilizar para implementarlo.

**DESARROLLO ORIENTADO A OBJETIVOS**

El software presenta características particulares que hacen que su complejidad alcance niveles importantes. A través de los años, los investigadores, ingenieros y desarrolladores han intentado minimizar esa dificultad incorporando distintas ideas de otros ámbitos al del software. Entre los paradigmas de desarrollo actuales mas destacados encontramos el orientado a objetos y el orientado a estructuras.

**El modelo de objetos**

El modelo de objetos no es solo una forma de programar sino que nos brinda los cimientos para poder desarrollar todo tipo de soluciones bajo sus principios. Muchas de las premisas de este paradigma han sido obtenidas de otras ingenierías y actividades. La Ingeniería Civil y la Aeronáutica han servido de guía para la incorporación de elementos al software. La principal razón que hace atractivo a este modelo es la capacidad que tiene para combatir la complejidad inherente y disminuir los riesgos en el desarrollo. Otros paradigmas también tienen sus éxitos en estos puntos pero el modelo de objetos no se resiente Canto en proyectos de gran alcance. La escalabilidad no lo afecta de forma tan profunda como a otros paradigmas.

**Jerarquía**

La jerarquía no es más que la posibilidad dc realiza un ordenamiento en niveles de lo que deseamos representar. De manera practica, esa jerarquía se ve representada por la herencia, que puede ser de distintos Tipos. Entre las más frecuentes encontramos la simple, la múltiple y la restrictiva. Todos los lenguajes orientados a objetos (OO) modernos nos brindan la posibilidad de heredar comportamiento.

**Abstracción**

Este termino es quizás el mas simple de entender, pero el que mas problemas representa a la hora de ser puesto en practica. La abstracción es un proceso intelectual humano per el cual somos capaces de concentrarnos particularmente en las características que nos interesan Para la solución de una situación. Al atacar un problema, intentamos resolverlo mediante la aplicación de pasos que hemos utilizado previamente y que sabemos (o creemos) que funcionaran.

**Modularidad**

Este concepto no es propio de la OO, sino que es uno de los que más se ha desarrollado en la ingeniería de software. El objetivo final es la división de un problema complejo en unidades más pequeñas y casi siempre más sencillas. A su vez, la separación en módulos crea fronteras artificiales que permiten que los grupos desarrollen soluciones por separado integrándolas con mayor facilidad. Los módulos interactúan entre ellos y pueden ser transportados a otros proyectos en caso de necesidad. La reutilización es uno de los pilares del modelo orientado a objetos.

**Encapsulamiento**

El encapsulamiento es el ocultamiento de la información de forma tal que solo este disponible para interactuar con un objeto sin la necesidad de conocer como se comporta internamente. Este factor hace que los objetos sean fácilmente reutilizables. Además, junto con los conceptos vistos anteriormente, crea la estructura precisa para una reducción de la complejidad. Muchas veces al encapsulamiento as lo relaciona correctamente con el termino caja negra. Un ejemplo sencillo ocurre cuando somos capaces de manejar algún artefacto sin saber exactamente que es lo que ocurre internamente. En este caso), el ocultamiento de información nos facilita la interacción.

**Tipos (tipificación)**

Los tipos permiten representar las abstracciones de forma adecuada. Es mas sencillo ejemplificar el concepto diciendo que si realizamos una suma de dos números enteros esperamos obtener otro número entero y no uno decimal.

**Concurrencia**

La concurrencia supone que en la solución de un problema se manejan toda clase de eventos y muchas veces algunas interactúan de manera asincrónica, pero también pueden hacerlo simultáneamente. Es por esta posibilidad que debemos incorporar mecanismos para la correcta operación en casos de interacción concurrente. Entre estos podemos citar el manejo de las sobrecargas y la prioridad de accesos.

**Persistencia**

La persistencia es lo que posibilita que trabajemos guardando información durante el tiempo que necesitamos operar con ella. Decimos que un objeto ocupa un espacio en memoria y existe o permanece en ella durante cierto lapso de tiempo. La persistencia nos permite mantener el estado del objeto.

**El objeto como base**

Lógicamente, la orientación a objetos presenta muchas características comunes a otro tipo de prácticas. Lo que la distingue es que pone el énfasis en definir y caracterizar de forma clara los componentes del sistema, dotándolos de sus capacidades. En el objeto se unen los datos y los algoritmos. Es así cómo se transforma en la piedra fundamental de esta estructura.

**Programación orientada a objetos**

Si queremos definir qué es exactamente la programación orientada a objetos y que no es, nos encontraremos en serias dificultades. La explicación podría tornarse trivial o, muy por el contrario, alcanzar gran cantidad de confusas expresiones.