Metodología ICONIX

Los desarrollos de aplicaciones van cambiando por innovaciones tecnológicas, estrategias de mercado y otros avatares de la industria de la informática, esto lleva a los desarrolladores de aplicaciones a evolucionar para obtener aplicaciones en menor tiempo, más vistosas y de menor costo.

Los usuarios exigen calidad frente a los requisitos y los desarrollos de aplicaciones deben contar con técnicas y herramientas logrando satisfacer las necesidades de los usuarios y obteniendo sistemas fáciles de mantener, extender y modificar.

Claro esta, que es indispensable, el uso de una metodología para el desarrollo de sistemas, logrando un sistema sano, que cumpla con los requerimientos de los usuarios.

Una metodología consiste en un lenguaje de modelamiento y un proceso. El lenguaje de modelamiento es la notación gráfica (incluye diferentes tipos de diagramas) en este caso UML. El proceso define quien debe hacer qué, cuando y como alcanzar un objetivo.

La realidad de la industria del software de gestión impone la adopción de procesos ágiles de desarrollo para lograr competitividad, ya que el **proceso** de desarrollo de software trae aparejado: altos costos, alta complejidad, dificultades de mantenimiento y una disparidad entre las necesidades de los usuarios y los productos desarrollados.

Reflejo de ello, en el ámbito internacional, es la creciente consolidación de la filosofía AGILE. El objetivo principal de un método ágil es minimizar la documentación de desarrollo empleándola fundamentalmente como vehículo de comprensión de problemas dentro del grupo de trabajo y de comunicación con los usuarios.

Esta herramienta importa una contribución para la comunidad informática dedicada al desarrollo de sistemas de gestión, dado que implica la adopción de una metodología simple y precisa que favorece la participación de los usuarios finales y mantiene a todo desarrollo permanentemente documentado.

La participación y el compromiso de los usuarios finales en desarrollos basados en esta herramienta se presumen garantizados debido a que los modelos empleados para las especificaciones son de un alto nivel de abstracción y comprensibles para personas no especializadas; además el modelo dinámico tal como el de casos de uso en el Proceso Unificado de Desarrollo permite verificar la completitud y rastrear el cumplimiento de sistemas a partir de la especificación del diseño de interfaces, optimiza las relaciones contractuales facilitando la aprobación de fases y ciclos de evolución.

En éste contexto el proceso **ICONIX** (Rosenberg & Scott, 1999) se define como un “proceso” de desarrollo de software práctico. ICONIX está entre la complejidad del RUP (Rational Unified Processes) y la simplicidad y pragmatismo del XP (Extreme Programming), sin eliminar las tareas de análisis y de diseño que XP no contempla.

ICONIX es un proceso simplificado en comparación con otros procesos más tradicionales, que unifica un conjunto de métodos de orientación a objetos con el objetivo de abarcar todo el ciclo de vida de un proyecto. Fue elaborado por Doug Rosenberg y Kendall Scott a partir de una síntesis del proceso unificado de los "tres amigos" Booch, Rumbaugh y Jacobson y que ha dado soporte y conocimiento a la metodología ICONIX desde 1993. Presenta claramente las actividades de cada fase y exhibe una secuencia de pasos que deben ser seguidos. Además ICONIX está adaptado a los patrones y ofrece el soporte de UML, dirigido por casos de uso y es un proceso iterativo e incremental.

Las tres características fundamentales de ICONIX son:

- **Iterativo e incremental:** varias iteraciones ocurren entre el desarrollo del modelo del dominio y la identificación de los casos de uso. El modelo estático es incrementalmente refinado por los modelos dinámicos.
- **Trazabilidad:** cada paso está referenciado por algún requisito. Se define trazabilidad como la capacidad de seguir una relación entre los diferentes artefactos producidos.
- **Dinámica del UML:** La metodología ofrece un uso "dinámico del UML" como los diagramas del caso de uso, diagramas de secuencia y de colaboración.

Las Tareas de ICONIX

Rosenberg y Scoot destacan un análisis de requisitos, un análisis y diseño preliminar, un diseño y una implementación como las principales tareas.

1- Análisis de Requisitos

a) Identificar en el "mundo real" los objetos y todas las relaciones de agregación y generalización entre ellos. Utilizar un diagrama de clases de alto nivel definido como **modelo de dominio.**

El trabajo es iniciado con un relevamiento informal de todos los requisitos que en principio deberían ser parte del sistema. Luego con los requisitos se construye el diagrama de clases, que representa las agrupaciones funcionales con que se estructura el sistema que se desarrolla.

De generarse el sistema a este nivel de especificación, se obtendría el menú principal del sistema con la interfaces iniciales de los casos o actividades de cada división funcional. Los diagramas del segundo nivel o superior, accesibles a partir de cada escenario o estado del nivel anterior, representan los casos, actividades y secuencias de interacción de cada división funcional. En estos se pueden reutilizar interfaces ya definidas en otros diagramas, representándose con bordes tenues.

b) Presentar, si es posible, una **prototipación** rápida de las interfaces del sistema, los diagramas de navegación, etc., de forma que los clientes puedan comprender mejor el sistema propuesto.

Con el prototipo se espera que las especificaciones iniciales estén incompletas. En general se necesita entre 2 y 3 reuniones para establecer las especificaciones iniciales. La rapidez con la que se

genera el sistema es esencial para que no se pierda el estado de ánimo sobre el proyecto y que los usuarios puedan comenzar a evaluar la aplicación en la mayor brevedad posible.

Durante la evaluación se debe capturar información sobre lo que les gusta y lo que les desagrada a los usuarios, al mismo tiempo poner atención al porque reaccionan los usuarios en la forma en que lo hacen.

Los cambios al prototipo son planificados con los usuarios antes de llevarlos a cabo.

El proceso se repite varias veces y finaliza cuando los usuarios y analistas están de acuerdo en que el sistema ha evolucionado lo suficiente como para incluir todas las características necesarias o cuando es evidente que no se obtendrá mayor beneficio con una iteración adicional.

El diseño de prototipos es una técnica popular de ingeniería para desarrollar modelos a escala (o simulados) de un producto o sus componentes. Cuando se aplica al desarrollo de sistemas de información el diseño de prototipos implica la creación de un modelo o modelos operativos de trabajo de una sistema o subsistema.

Existen cuatro tipos de prototipos:

- Prototipo de viabilidad: para probar la viabilidad de una tecnología especifica aplicable a un sistema de información.
- Prototipo de Necesidades: utilizado para "descubrir" las necesidades de contenido de los usuarios con respecto a la empresa.
- Prototipo de Diseño: es el que usa Iconix. Se usa para simular el diseño del sistema de información final. Se centra en la forma y funcionamiento del sistema deseado. Cuando un analista crea un prototipo de diseño, espera que los usuarios evalúen este prototipo, como si formara parte del sistema final. Los usuarios deberían evaluar la facilidad de aprendizaje y manejo del sistema, así como el aspecto de las pantallas y los informes y los procedimientos requeridos para utilizar el sistema. Estos prototipos pueden servir como especificaciones parciales de diseño o evolucionar hacia prototipos de información.
- Prototipo de Implantación: es una extensión de los prototipos de diseño donde el prototipo evoluciona directamente hacia el sistema de producción.

Los prototipos de pantallas también proporcionan una manera de obtener las reacciones de los usuarios hacia la cantidad de información presentada sobre la pantalla de visualización. Tal vez el usuario decida que un diseño en particular es muy denso ya que existen demasiados detalles sobre la pantalla. En otros casos la información sobre la pantalla aunque no es excesiva en el sentido de causar que la pantalla se vuelva densa, tal vez sea mucho mayor que la que un individuo necesita durante todo el tiempo.

Ventajas:

- Los usuarios se hacen participantes más activos en los desarrollos del sistema. Suelen mostrarse mas interesados en los prototipos de trabajo que en las especificaciones de diseño.

- La definición de necesidades se simplifica por el hecho de que muchos usuarios finales no comprenden o no son capaces de enumerar detalladamente sus necesidades hasta que ven un prototipo.

- La probabilidad de que los usuarios aprueben un diseño y luego rechacen su implantación se reducirá notablemente.

- Según se dice el diseño mediante prototipos reduce el tiempo de desarrollo, aunque algunos cuestionan este ahorro.

- Los prototipos suelen pasar a las fases de análisis y diseño con demasiada rapidez. Ello empuja al analista a pasar demasiado rápido a la codificación, sin haber comprendido las necesidades y los problemas. Condición deseable en un proceso ágil.

c) Identificar los casos de uso del sistema mostrando los actores involucrados. Utilizar para representarlo el **modelo de casos de uso**.

Los casos de uso describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista de un usuario; permiten definir los limites del sistema y las relaciones entre el sistema y el entorno.

Un caso de uso es una manera especifica de utilizar un sistema. Es la imagen de una funcionalidad del sistema, desencadenada en respuesta a la estimulación de un actor externo.

El modelo de los casos de uso comprende los actores, el sistema y los propios casos de uso. El conjunto de funcionalidades de un sistema se determina examinando las necesidades funcionales de cada actor.

Los casos de usos reubican la expresión de las necesidades sobre los usuarios partiendo del punto de vista muy simple que dice que un sistema se construye ante todo para sus usuarios. La estructuración del método se efectúa respecto a las interacciones de una sola categoría de usuarios a la vez; esta partición del conjunto de necesidades reduce considerablemente la complejidad de la determinación de las necesidades.

Los casos de uso permiten a los usuarios estructurar y articular sus deseos; les obligan a definir la manera como querrían interactuar con el sistema, a precisar que informaciones quieren intercambiar y a describir lo que debe hacerse para obtener el resultado esperado. Los casos de uso concretan el futuro sistema en una formalización próxima al usuario, incluso en ausencia de un sistema a criticar.

d) Organizar los casos de uso en grupos, o sea, utilizar los **diagramas de paquetes.**
e) Asociar los requisitos funcionales con los casos de uso y con los objetos del dominio (trazabilidad).

Un importante aspecto de ICONIX es que un requisito se distingue explícitamente de un caso de uso. En este sentido, un caso de uso describe un comportamiento; un requisito describe una regla

para el comportamiento. Además, un caso de uso satisface uno o más requisitos funcionales; un requisito funcional puede ser satisfecho por uno o más casos de uso.

2- Análisis y Diseño Preliminar

a) Describir los casos de uso, como un flujo principal de acciones, pudiendo contener los flujos alternativos y los flujos de excepción. La principal sugerencia de ICONIX, en esta actividad, es que no se debe perder mucho tiempo con la descripción textual. Debería usarse un estilo consistente que sea adecuado al contexto del proyecto.
b) Realizar un **diagrama de robustez**. Se debe ilustrar gráficamente las interacciones entre los objetos participantes de un caso de uso. Este diagrama permite analizar el texto narrativo de cada caso de uso e identificar un conjunto inicial de objetos participantes de cada caso de uso.

El análisis de robustez ayuda a identificar los objetos que participaran en cada caso de uso. Estos objetos que forman parte de los diagramas de robustez se clasifican dentro de los tres tipos siguientes:

- Objetos de interfaz: usados por los actores para comunicarse con el sistema. Son con los que los actores interactúan con el sistema, generalmente como ventanas, pantalla, diálogos y menús.
- Objetos entidad: son objetos del modelo del dominio. Son a menudo tablas y archivos que contiene archivos para la ejecución de dicho caso de uso.
- Objetos de control: es la unión entre la interfaz y los objetos entidad. Sirven como conexión entre los usuarios y los datos. Los controles son "objetos reales" en un diseño, pero usualmente sirven como una especie de oficinista para asegurar que no se olvide ninguna funcionalidad del sistema la cual puede ser requerida por algún caso de uso.

Esta técnica tan simple pero poderosa sirve como interfaz entre el "que" y el "como" de un análisis. Además el análisis de robustez provee de una gran ayuda a saber si las especificaciones del sistema son razonables.

El análisis de robustez facilita el reconocimiento de objetos. Esto es un paso crucial ya que es casi seguro que se olvida algunos objetos durante el modelado del dominio; y de esta manera se podrán identificar antes de que esto cause problemas serios, además sirve para identificar mas y mejores clases, antes del desarrollo del diagrama de secuencias.

Las reglas básicas que se deben aplicar al realizar los diagramas de análisis de robustez:

- Actores solo pueden comunicarse con objetos interfaz.
- Las interfaces solo pueden comunicarse con controles y actores.
- Los objetos entidad solo pueden comunicarse con controles.
- Los controles se comunican con interfaces, objetos identidad y con otros controles pero nunca con actores.

Tomando en cuenta que los objetos entidad y las interfaces son sustantivos y los controles son verbos. Se pueden enunciar de manera sencilla que los sustantivos nunca se comunican con otros sustantivos, pero los verbos, si pueden comunicarse con otros verbos y a su vez con sustantivos.

c) Actualizar el diagrama de clases ya definido en el modelo de dominio con las nuevas clases y atributos descubiertas en los diagramas de robustez.

3- Diseño

a) Especificar el comportamiento a través del **diagrama de secuencia.** Para cada caso de uso identificar los mensajes entre los diferentes objetos. Es necesario utilizar los diagramas de colaboración para representar la interacción entre los objetos.

El diagrama de secuencia muestra interacciones entre objetos según un punto de vista temporal. El contexto de los objetos no se representa de manera explicita como en los diagramas de colaboración. La representación se concentra sobre la expresión de las interacciones.

A pesar de que a partir de los diagramas de casos de uso y de los diagramas de robustez ya tenemos entre un 75 y 80 por ciento de atributos de nuestras clases identificados, es hasta el diagrama de secuencia donde se empiezan a ver que métodos llevaran las clases de nuestro sistema. Esto se debe a que hasta que vemos interactuando a los objetos de nuestras clases con los actores y con otros objetos de manera dinámica, hasta ese momento tenemos suficiente información como para poder empezar a especificar los métodos de nuestras respectivas clases.

El diagrama de secuencia es el núcleo de nuestro modelo dinámico y muestra todos los cursos alternos que pueden tomar todos nuestros casos de uso. Los diagramas de secuencia se componen de 4 elementos que son: el curso de acción, los objetos, los mensajes y los métodos (operaciones)

b) Terminar el modelo estático, adicionando los detalles del diseño en el **diagrama de clases**.
c) Verificar si el diseño satisface todos los requisitos identificados

4- Implementación

a) Utilizar el diagrama de componentes, si fuera necesario para apoyar el desarrollo. Es decir, mostrar la distribución física de los elementos que componen la estructura interna del sistema.

El diagrama de componentes describe los elementos físicos y sus relaciones en el entorno de realización. El diagrama muestra las opciones de realización.

b) Escribir/ Generar el código

La importancia de la interactividad, interactividad, accesibilidad y navegación en el software harán que el usuario se sienta seguro y cómodo al poder hacer uso de la aplicación sin inconvenientes tales como son los problemas de comunicación. Este y otros problemas como la realización de cambios, son factores que deben ser tenidos en cuenta.

Pero además debemos tener en cuenta factores como:

- **La Reusabilidad**: que es la posibilidad de hacer uso de los componente en diferentes aplicaciones.
- **La Extensibilidad**: que consiste en modificar con facilidad el software.
- **La Confiabilidad**: realización de sistemas descartando las posibilidades de error.

c) Realizar pruebas. Test de unidades, de casos, datos y resultados. Test de integración con los usuarios para verificar la aceptación de los resultados.