Introducción al Diseño de Sistemas de Información

Unidad Nº I: Ciclo de Vida de Proyectos

Facultad Regional Santa Fe Universidad Tecnológica Nacional

El ciclo de vida del proyecto

Para que un proceso de desarrollo de software pueda controlarse y asegurarse que construya software de alta calidad de una forma productiva, es necesario proveedor de *métodos*, *herramientas* y *procedimientos*.

Los *métodos* indican "cómo" construir técnicamente el software. Los métodos abarcan un amplio espectro de tareas que incluyen: planificación y estimación de proyectos, análisis de los requisitos del sistema y del software, diseño de estructura de datos, arquitectura de programas y procedimientos algorítmicos, codificación, prueba y mantenimiento. Los métodos introducen frecuentemente una notación especial orientada a un lenguaje o gráfica y un conjunto de criterios para la calidad del software.

Las *herramientas* suministran un soporte automático o semiautomático para los métodos. Existen herramientas para soportar cada uno de los métodos mencionados anteriormente. Cuando se integran las herramientas de forma que la información creada por una herramienta pueda ser usada por otra, se establece un sistema para el soporte de desarrollo del software, llamado ingeniería del software asistida por computadora (CASE).

Los *procedimientos* son el pegamento que junta los métodos y las herramientas y facilita un desarrollo racional y oportuno del software. Los procedimientos definen la secuencia en la que se aplican los métodos, las entregas (documentos, informes, formas, etc.) que se requieren, los controles que ayudan a asegurar la calidad y coordinar los cambios, y las directrices que ayudan a los gestores del software a evaluar el progreso.

A su vez se definen una serie de pasos que abarcan los métodos, las herramientas y los procedimientos. Estos pasos se denominan frecuentemente *paradigmas de la ingeniería del software o ciclos de vida de proyecto*.

Ciclo de vida clásico

Este paradigma es llamado algunas veces 'modelo en cascada', el paradigma del ciclo de vida exige un enfoque sistemático y secuencial del desarrollo del software que comienza en el nivel del sistema y progresa a través del análisis, diseño, codificación, prueba y mantenimiento.

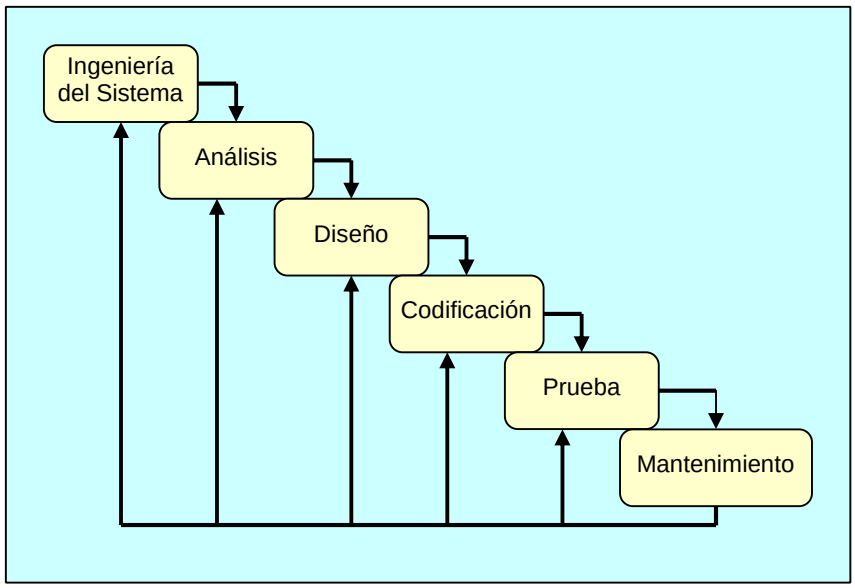

Figura I. 1 - Ciclo de Vida Clásico

Modelizado a partir del ciclo convencional de una ingeniería, el paradigma del ciclo de vida abarca las siguientes actividades:

Ingeniería y análisis del sistema. Debido a que el software es siempre parte de un sistema mayor, el trabajo comienza estableciendo los requisitos de todos los elementos del sistema y luego asignando algún subconjunto de estos al software. Este planteamiento del sistema es esencial cuando el software debe interrelacionarse con otros elementos, tales como hardware, personas y bases de datos. La ingeniería y el análisis del sistema abarca

los requisitos globales a nivel del sistema con una pequeña cantidad de análisis y de diseño a un nivel superior.

Análisis de los requisitos del software. El proceso de recopilación de los requisitos se centra e intensifica especialmente para el software. Para comprender la naturaleza de los programas que hay que construir, el analista debe comprender el ámbito de la información del software, así como la función, el rendimiento y las interfaces requeridos. Los requisitos, tanto del sistema como del software, se documentan y se revisan con el cliente.

Diseño. El diseño es realmente un proceso multipaso que se enfoca sobre cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software que pueda ser establecida de forma que obtenga la calidad requerida antes de que comience la codificación.

Codificación. El diseño debe traducirse en una forma legible para la máquina. El paso de codificación realiza esta tarea. Si el diseño se realiza de una manera detallada, la codificación puede realizarse mecánicamente.

Prueba. Una vez que se ha generado el código, comienza la prueba del programa. La prueba se centra en la lógica interna del software, asegurando que todas las sentencias se han probado, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren.

Mantenimiento. El software, indudablemente, sufrirá cambios después de que se entregue al cliente. Los cambios ocurrirán debido a que se hayan encontrado errores, a que el software deba adaptarse a cambios del entorno externo (por ejemplo, un cambio solicitado debido a que se tiene un nuevo sistema operativo o dispositivo periférico), o debido a que el cliente requiera ampliaciones funcionales o de requerimientos. El mantenimiento del software aplica cada uno de los pasos precedentes del ciclo de vida a un programa existente en vez a uno nuevo.

El ciclo de vida clásico es el paradigma más antiguo y más ampliamente usado en la ingeniería del software. Sin embargo, con el paso de unos cuantos años, se han producido críticas al paradigma que cuestionan su aplicabilidad a todas las situaciones. Entre los problemas que se presentan algunas veces, cuando se aplica este paradigma se encuentra:

1. Los proyectos raramente siguen el flujo secuencial que propone el modelo. Siempre hay iteraciones y se crean problemas en la aplicación del paradigma.
2. Normalmente, es difícil para el cliente establecer explícitamente al principio todos los requisitos. El ciclo de vida clásico lo requiere y tiene dificultades en acomodar posibles incertidumbres que pueden existir al comienzo de muchos proyectos.
3. El cliente debe tener paciencia. Hasta llegar a las etapas finales del desarrollo del proyecto, no estará disponible una versión operativa del programa. Un error importante no detectado hasta que el programa esté funcionando puede ser desastroso.

Cada uno de estos problemas es real. Sin embargo, el paradigma clásico del ciclo de vida tiene un lugar definido e importante dentro del trabajo realizado en ingeniería del software. Suministra una plantilla en la que pueden colocarse los métodos para el análisis, diseño, codificación, prueba y mantenimiento. Y, a pesar de sus inconvenientes, es significativamente mejor que desarrollar el software sin guías.

Construcción de prototipos

Normalmente un cliente define un conjunto de objetivos generales para el software, pero no identifica los requisitos detallados de entrada, proceso o salida. En otros casos, el programador puede no estar seguro de la eficiencia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma en que debe realizarse la interacción hombre-máquina. En estas y muchas otras situaciones, puede ser mejor método la construcción de un prototipo.

La construcción de prototipos es un proceso que facilita al programador la creación de un *modelo* del software a construir. El modelo tomará una de las tres formas siguientes:

1. un prototipo en papel o un modelo basado en PC que describa la interacción hombre-máquina, de forma que facilite al usuario la comprensión de cómo se producirá tal interacción;
2. un prototipo que implemente algunos subconjuntos de a función requerida del programa deseado, o
3. un programa existente que ejecute parte o toda la función deseada, pero que tenga otras características que deban ser mejoradas en el nuevo trabajo de desarrollo.

En la Figura I.2 se muestra la secuencia de sucesos del paradigma de construcción de prototipos.

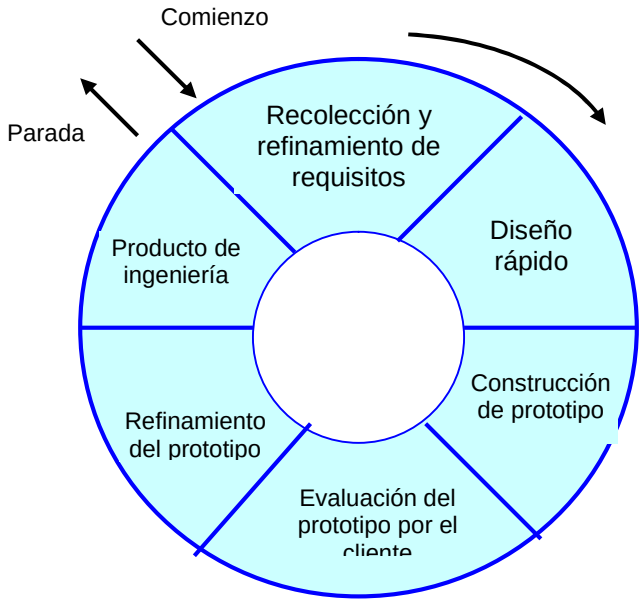

Figura I. 2 – Ciclo de Construcción de Prototipos

Como en todos lo métodos de desarrollo de software, la construcción de prototipos comienza con la recolección de los requisitos. El técnico y el cliente se reúnen y definen los objetivos globales para el software, identifican todos los requisitos conocidos y perfilan las áreas en donde será necesario una mayor definición. Luego se produce un "diseño rápido". El diseño rápido se enfoca sobre la representación de los aspectos del software visibles al usuario (por ejemplo, métodos de entrada y formatos de salida). El diseño rápido conduce a la construcción de un prototipo. El prototipo es evaluado por el cliente-usuario y se utiliza para refinar los requisitos del software a desarrollar. Se produce un proceso interactivo en el que el prototipo es "afinado" para que satisfaga las necesidades del cliente, al mismo tiempo que facilita al que lo desarrolla una mejor comprensión de lo que hay que hacer.
Idealmente, el prototipo sirve como mecanismo para identificar los requisitos del software. Si se va a construir un prototipo que funcione, el realizador intenta hacer uso de fragmentos de programas existentes o aplica herramientas (generadores de informes, generadores de interfaces de usuarios, etc.) que faciliten la generación de programas que funcionen.
Pero, ¿qué debemos hacer con el prototipo cuando ya ha servido para el propósito establecido?. Brooks nos da una respuesta:

> *En la mayoría de los proyectos, el primer sistema construido apenas es utilizable. Puede ser demasiado lento, demasiado grande, difícil de usar o las tres cosas. No hay más alternativa que comenzar de nuevo y construir una versión rediseñada que resuelva los problemas que se presenten... Cuando se utiliza un nuevo concepto de sistema o de tecnología, hay que construir un sistema para desecharlo, porque incluso la mejor planificación no puede asegurar que vaya a ser bueno la primera vez. Por tanto, la cuestión no es si hay que construir un sistema piloto y tirarlo. Se tirará. La única cuestión es si planificar de antemano la construcción de algo que se va a desechar, o prometer entregar el desecho a los clientes...*

El prototipo puede servir como "primer sistema". Pero esto puede ser una visión idealizada. Al igual que en el ciclo de vida clásico, la construcción de prototipos como paradigma para la ingeniería del software, puede ser problemática por las siguientes razones:

1. El cliente ve funcionando lo que parece ser una primera versión del software, ignorando que el prototipo se ha hecho con "plastilina y alambres", ignorando que, por las prisas en hacer que funcione, no hemos considerado los aspectos de calidad o de mantenimiento del software a largo plazo. Cuando se le informa de que el producto debe ser reconstruido, el cliente se vuelve loco y solicita que se apliquen "cuantas mejoras" sean necesarias para

hacer del prototipo un producto final que funcione. El gestor del desarrollo del software cede demasiado a menudo.

2. El técnico de desarrollo, frecuentemente, impone ciertos compromisos de implementación con el fin de obtener un prototipo que funcione rápidamente. Puede que utilice un sistema operativo o un lenguaje de programación inapropiados, simplemente porque ya está disponible y es conocido; puede que implemente ineficientemente un algoritmo, sencillamente para demostrar su capacidad. Después de algún tiempo, el técnico puede haberse familiarizado con esas elecciones y haber olvidado las razones por las que eran inapropiadas. La elección menos ideal forma ahora parte integral del sistema.

Aunque pueden aparecer problemas, la construcción de prototipos es un paradigma efectivo. La clave está en definir al comienzo las reglas del juego; esto es, el cliente y el técnico debe estar de acuerdo en que el prototipo se construya para servir sólo como un mecanismo de definición de los requisitos. Posteriormente, ha de ser descartado (al menos en parte) y debe construir el software real, con los ojos puestos en la calidad y en el mantenimiento.

El modelo en espiral

El modelo en espiral ha sido desarrollado para cubrir las mejores características tanto del ciclo de vida clásico, como de la creación de prototipos, añadiendo al mismo tiempo un nuevo elemento: *el análisis de riesgo*. El modelo, representado mediante el espiral de la Figura I.3, define cuatro actividades principales, representadas por los cuatro cuadrantes de la figura:

1. *Planificación*: determinación de objetivos, alternativas y restricciones.
2. *Análisis de riesgo*: análisis de alternativas e identificación/resolución de riesgos.
3. *Ingeniería*: desarrollo del producto de "siguiente nivel".
4. *Evaluación del cliente*: valoración de los resultados de la ingeniería.

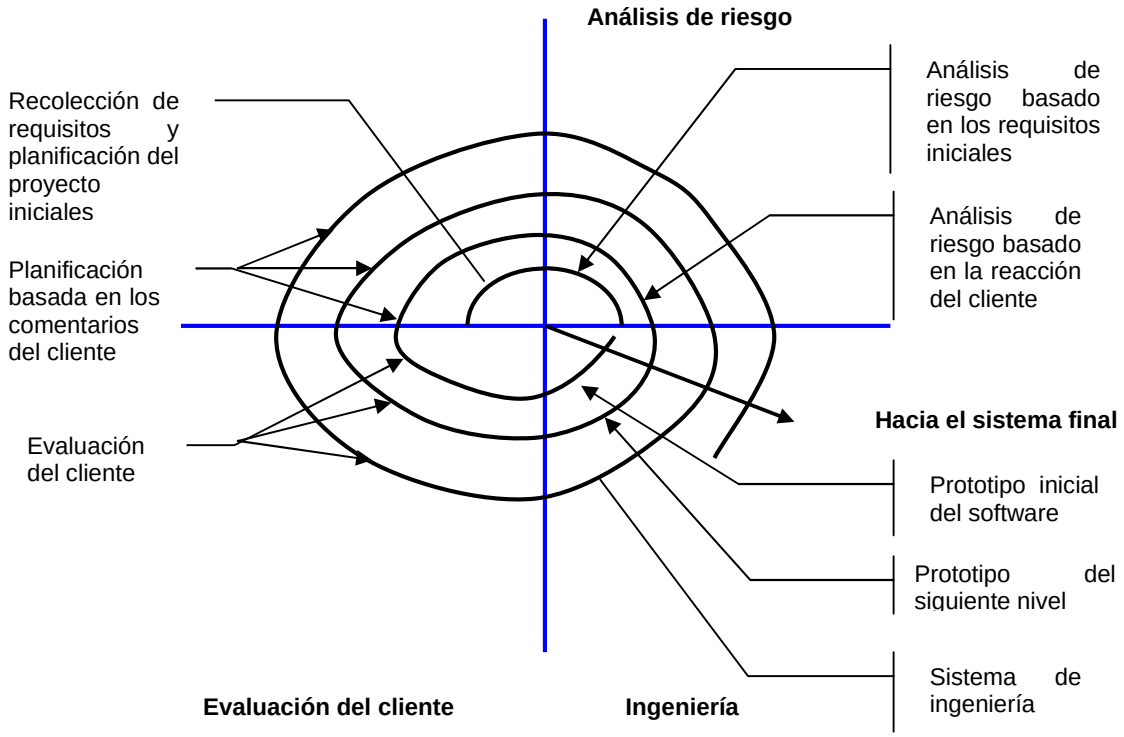

Figura I. 3 – El modelo de espiral

Un aspecto intrigante del modelo en espiral se hace evidente cuando consideramos la dimensión radial representada en la figura anterior. Con cada iteración alrededor de la espiral, se construyen sucesivas versiones del software, cada vez más completas. Durante la primera vuelta alrededor de la espiral se definen los objetivos, las alternativas y las restricciones, y se analizan e identifican los riesgos. Si el análisis de riesgo indica que hay una incertidumbre en los requisitos, se puede usar la creación de prototipos en el cuadrante de ingeniería para dar

asistencia tanto al encargado de desarrollo como al cliente. Se pueden usar simulaciones y otros modelos para definir más el problema y refinar los requisitos.
El cliente evalúa el trabajo de ingeniería y sugiere modificaciones. En base a los comentarios del cliente se produce la siguiente fase de planificación y de análisis de riesgo. En cada bucle alrededor de la espiral, la culminación del análisis de riesgo resulta en una decisión de "seguir o no seguir". Si los riesgos son demasiados grandes, se puede dar por terminado el proyecto.

Sin embargo, en la mayoría de los casos, se sigue avanzando alrededor del camino de la espiral, y ese camino lleva a los desarrolladores hacia fuera, hacia un modelo más completo del sistema, y, al final, al propio sistema operacional. Cada vuelta alrededor de la espiral requiere ingeniería (cuadrante inferior derecho), que se puede llevar a cabo mediante el enfoque del ciclo de vida clásico o de la creación de prototipos. Debe tenerse en cuenta que el número de actividades de desarrollo que ocurren en el cuadrante inferior derecho aumenta al alejarse del centro de la espiral.
Este paradigma es actualmente el enfoque más realista para el desarrollo de software y de sistemas a gran escala. Utiliza un enfoque "evolutivo" para la ingeniería del software permitiendo al desarrollador y al cliente entender y reaccionar a los riesgos en cada nivel evolutivo. Utiliza la creación de prototipos como un mecanismo de reducción del riesgo, pero, lo que es más importante, permite a quien lo desarrolla aplicar el enfoque de creación de prototipos en cualquier etapa de la evolución del producto. Mantiene el enfoque sistemático correspondiente a los pasos sugeridos por el ciclo de vida clásico, pero incorporándola dentro de un marco de trabajo interactivo que refleja de forma más realista el mundo real. El modelo en espiral demanda una consideración directa de riesgos técnicos en todas las etapas del proyecto y, si se aplica adecuadamente, debe reducir los riesgos antes de que se conviertan en problemáticos.

El ciclo de vida estructurado

Veamos ahora la propuesta que realiza Edward Yourdon en su libro "Análisis Estructurado Moderno", en proceso claramente orientado al enfoque estructurado.
En la Figura I.4, se ve este ciclo de vida.
Los terminadores son los usuarios, los administradores y el personal de operaciones. Se trata de individuos o grupos que proporcionan las entradas al equipo del proyecto, y son los beneficiados finales del sistema. Ellos interactúan con las nueve actividades que se muestran en la figura I.4.

Actividad 1: La encuesta

Esta actividad también se conoce como el *estudio de factibilidad* o como el *estudio inicial de negocios*. Normalmente, empieza cuando el usuario solicita que una o más partes de su sistema se automaticen. Los principales objetivos de la encuesta son los siguientes:

- *Identificar a los usuarios responsables y crear un "campo de actividad" inicial del sistema*. Esto puede comprender la conducción de una serie de entrevistas para determinar qué usuarios estarán comprendidos en (o serán afectados por) el proyecto propuesto. Pudiera también implicar el desarrollo de un diagrama inicial de contexto, que es un diagrama de flujo de datos sencillo en el cual se representa el sistema completo con un solo proceso.
- *Identificar las deficiencias actuales en el ambiente del usuario*. Esto en general comprenderá la lista de funciones que hacen falta o que se están llevando a cabo insatisfactoriamente en el sistema actual. Por ejemplo, esto pudiera incluir declaraciones como las siguientes:
    - El hardware del sistema actual no es confiable y el vendedor se acaba de declarar en quiebra.
    - El software del sistema actual no se puede mantener, y no podemos ya contratar programadores de mantenimiento dispuestos a darle mantenimiento en el lenguaje que originalmente se utilizó para desarrollarlo.
    - El sistema actual no es capaz de producir los informes requeridos por la modificación a los impuestos decretada el año anterior.
    - El sistema actual no es capaz de recibir los informes sobre límites de crédito del departamento de contabilidad, y no puede producir los informes de promedio de volumen de pedidos que requiere el departamento de mercadotecnia.

- *Establecer metas y objetivos para un sistema nuevo.* Esto puede ser también una simple lista narrativa que contenga las funciones existentes que deben reimplantarse, las nuevas que necesitan añadirse y los criterios de desempeño del nuevo sistema.
- *Determinar si es factible automatizar el sistema y de ser así, sugerir escenarios aceptables.* Esto implicará algunas estimaciones bastante rudimentarias y aproximadas del costo y el tiempo necesarios para construir un sistema nuevo y los beneficios que se derivarán de ello; también involucrará dos o más escenarios. Aunque a estas alturas la administración y los usuarios usualmente querrán una estimación precisa y detallada, el analista tendrá mucha suerte si logra determinar el tiempo, los recursos y los costos con un error menor del 50% en esta etapa tan temprana del proyecto.

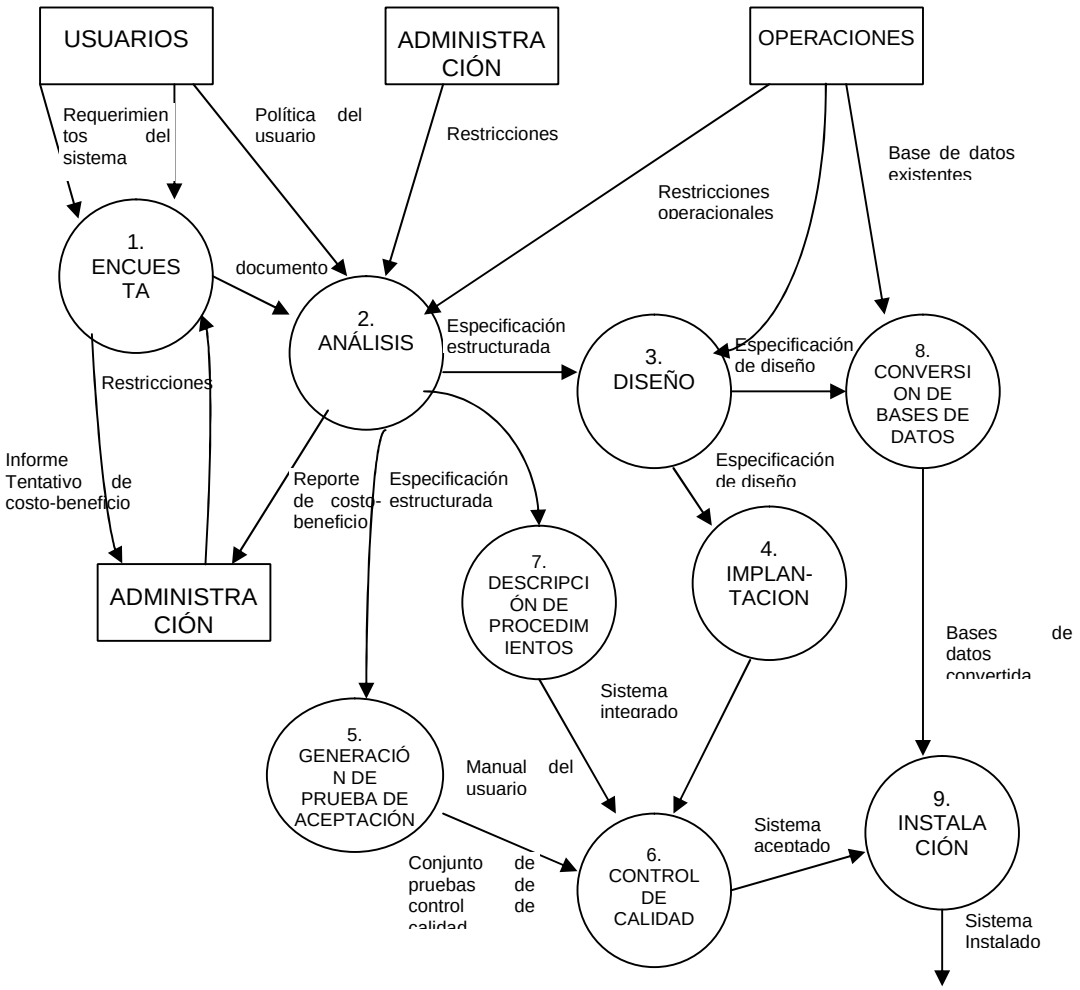

Figura I. 4 – El ciclo de Vida del Proyecto Estructurado

- *Preparar el esquema que se usará para guiar el resto del proyecto.* Este esquema incluirá toda la información que se lista anteriormente, además de identificar al administrador responsable del proyecto. También pudiera describir los detalles del ciclo de vida que seguirá el resto del proyecto.

En general, la encuesta ocupa sólo del 5 al 10 % del tiempo y los recursos de todo el proyecto, y para los proyectos pequeños y sencillos pudiera ni siquiera ser una actividad formal. Sin embargo, aún cuando no consuma mucho del tiempo y de los recursos del proyecto , es una actividad verdaderamente importante: al final de la encuesta, la administración pudiera decidir cancelar el proyecto si no parece atractivo desde el punto de vista de costo-beneficio.

Actividad 2: El análisis de sistemas

El propósito principal de la actividad de análisis es transformar sus dos entradas principales, las políticas del usuario y el esquema del proyecto, en una especificación estructurada. Esto implica modelar el ambiente del usuario con diagramas de flujo de datos, diagramas de entidad – relación, diagramas de transición de estados y demás herramientas que pueden utilizarse con este fin.
Esta actividad implica, simplificadamente, el desarrollo de un *modelo ambiental* y el desarrollo de un *modelo de comportamiento*. Estos dos modelos se combinan para formar el *modelo esencial*, que representa una descripción formal de lo que el nuevo sistema debe hacer, independientemente de la naturaleza de la tecnología que se use para cubrir los requerimientos.
Además del modelo del sistema que describe los requerimientos del usuario, generalmente se prepara un conjunto de presupuestos y cálculos de costos y beneficios más precisos y detallados al final de la actividad de análisis.
Obviamente, como analista de sistemas, en esto pasará la mayor parte de su tiempo.

Actividad 3: El diseño

La actividad de diseño se dedica a asignar porciones de la especificación (modelo esencial) a procesadores adecuados (sean máquinas o humanos) y a tareas apropiadas dentro de cada procesador. Dentro de cada tarea, la actividad de diseño se dedica a la creación de una jerarquía apropiada de módulos de programas y de interfases entre ellos para implantar la especificación creada en la actividad 2. Además, la actividad de diseño se ocupa de la transformación de modelos de datos de entidad-relación en un diseño de bases de datos.

Actividad 4: Implantación

Esta actividad incluye la codificación y la integración de módulos en un esqueleto progresivamente más completo del sistema final. Por eso, esta actividad incluye tanto programación como implantación.

Actividad 5: Generación de pruebas de aceptación

La especificación estructurada debe contener toda la información necesaria para definir un sistema que sea aceptable desde el punto de vista del usuario. Por eso, una vez generada la especificación, puede comenzar la actividad de producir un conjunto de casos de prueba de aceptación desde la especificación estructurada.

Actividad 6: Garantía de calidad

La garantía de calidad también se conoce como la prueba final o la prueba de aceptación. Esta actividad requiere como entradas los datos de la prueba de aceptación generada en la actividad 5 y el sistema integrado producido en la actividad 4.
Esta actividad también es conocida como “control de calidad”. Sin importar como se llame, se necesita una actividad que *verifique* que el sistema tenga un nivel apropiado de calidad. Nótese también que es importante llevar a cabo actividades de garantía de calidad *en cada una* de las actividades anteriores para asegurar que se hayan realizado con un nivel apropiado de calidad. Por esto, se esperaría que esto se haga durante toda la actividad de análisis, diseño y programación para asegurar que el analista esté desarrollando especificaciones de alta calidad, que el diseñador esté produciendo diseños de alta calidad y que el programador esté escribiendo código de alta calidad.

Actividad 7: Descripción del procedimiento

El desarrollo de un sistema implica tanto la parte automatizada como la parte que llevarán a cabo las personas. Por ello, una de las actividades importantes a realizar es la generación de una descripción formal de las partes del sistema que se harán en forma manual, lo mismo que la descripción de cómo interactuarán los usuarios con la parte automatizada del nuevo sistema.
El resultado de la actividad 7 es un manual para el usuario.

Actividad 8: Conversión de bases de datos

En algunos proyectos, la conversión de bases de datos involucrara más trabajo (y más planeación estratégica) que el desarrollo de programas de computadora para el nuevo sistema. En otros casos, pudiera no haber existido una base de datos que convertir. En general, esta actividad requiere como entrada la base de datos actual del usuario, al igual que la especificación del diseño producida por medio de la actividad 3.

Actividad 9: Instalación

La actividad final, desde luego, es la instalación; sus entradas son el manual de usuario producido en la actividad 7, la base de datos convertida que se creó con la actividad 8 y el sistema aceptado producido por la actividad 6. En algunos casos, sin embargo, la instalación pudiera significar simplemente un cambio de la noche a la mañana al nuevo sistema, sin mayor alboroto; en otros casos, la instalación pudiera ser un proceso gradual, en el que un grupo tras otro de usuarios van recibiendo manuales y entrenamiento y comenzando a usar el nuevo sistema.

Resumen del ciclo de vida del proyecto estructurado

Es importante ver la figura I.4 como lo que es: un *diagrama de flujo de datos*. Nada implica que toda la actividad N debe concluir antes de comenzar la actividad N + 1. Por el contrario, la red de flujos de datos que conectan las actividades hacen ver con claridad que pudieran estarse llevando a cabo diversas actividades paralelamente. Debido a este aspecto no secuencial, usamos la palabra *actividad* en el ciclo de vida del proyecto estructurado en lugar de “fase”, que es más convencional. El término fase tradicionalmente se refiere a un período particular en un proyecto en el cual se estaba desarrollando una, y sólo una, actividad.

Hay otra cosa que debe recalcarse acerca del uso de un diagrama de flujo de datos para describir el ciclo de vida del proyecto: un diagrama de flujo de datos clásico, como el que se muestra en la figura I.4, no muestra en forma explícita la retroalimentación, ni el control. Prácticamente todas las actividades de la figura I.4 pueden y suelen producir información que puede llevar a modificaciones adecuadas de una o más de las actividades precedentes. De aquí que la actividad de diseño puede producir información que acaso cambie algunas de las decisiones de costo-beneficio en la actividad de análisis; de hecho, el conocimiento que se obtiene a partir de la actividad de diseño pudiera incluso llevar a revisar decisiones anteriores acerca de la factibilidad básica del proyecto.

Más aún, en casos extremos, ciertos eventos que pudiera darse en cualquier actividad pueden causar que todo el proyecto termine repentinamente. Las entradas de la administración se muestran sólo para la actividad de análisis pues ésta es la única que requiere datos de la administración, sin embargo, se supone, que la administración ejerce *control* sobre las actividades.

En resumen, la representación del proceso sólo señala la o las entradas requeridas por cada actividad, y la o las salidas o productos que se generan. La secuencia de las actividades sólo puede suponerse en la medida en que la presencia o ausencia de datos haga posible comenzar una determinada actividad.

Observando estos ‘Ciclos de Vida de Proyectos’, podemos ver que

en general, la actividad de Diseño se lleva a cabo ***luego*** de haber realizado el Análisis o Relevamiento de Requisitos, y ***antes*** de comenzar la construcción del software.

Referencias - Bibliografía

- “Ingeniería del Software – Un enfoque práctico”, Roger. S. Pressman, Mc Graw-Hill, 1993
- “Análisis Estructurado Moderno”, Edward Yourdon, Prentice Hall, 1989