Revista Educación en Ingeniería
ISSN 1900-8260
Diciembre 2006 • Nº. 2 • Pp 62-70 • Publicado en línea por la Asociación Colombiana de Facultades de Ingeniería -ACOFI- www.acofi.edu.co
Recibido: 29/09/2006 • Aceptado: 26/10/2006

LOS CICLOS DE VIDA DE DESARROLLO, UNA NECESIDAD LATENTE

Juan Pablo Giraldo Rendón y Carlos Alberto Cortés Carrillo
Facultad de Ingeniería, Universidad de Manizales, Manizales (Colombia)

Resumen

Los ciclos de vida de desarrollo se plantean en la actualidad como herramientas de trabajo asociadas a los procesos de desarrollo de software, herramientas que permiten orientar las fases de un proyecto, sus controles, la documentación y los procesos de ingeniería de software requeridos para obtener productos de calidad.

En el documento se presentan generalidades asociadas a los procesos de desarrollo, a los modelos orientados a objetos, a las características generales de los ciclos de vida y las propuestas aplicadas a tres ciclos de vida. De los **procesos de desarrollo**, se tocan temáticas asociadas a los elementos de calidad, y las estrategias que apoyan estos procesos. Acerca de los **modelos orientados a objetos** se realiza un repaso por los beneficios que aportan y la necesidad de su aplicación en la actualidad. En los temas de **ciclos de vida** se presentan características y ventajas y un recorrido histórico. Finalmente se presentan casos de ejemplo de adaptación y **aplicación de ciclos de vida** a casos exitosos.

Palabras clave: ciclo de vida, metodología, calidad de software

Abstract

Currently the developmental life cycles are presented as tools of associated to the software development processes, tools that that allow to guide the phases of a project, its controls, the documentation and the required software engineering processes to obtain products of quality.

In the present written report associated generalities to the processes of development are presented, to the models oriented to objects, to the general characteristics of the cycles of life and the proposals applied to three cycles of life. From the processes of development, topics associates to the elements of quality they are studied, and the strategies that support these processes. A review about the models oriented to objects is carried out from the benefits that contribute and the need of their application nowadays. In the life cycles themes characteristics are studied and advantages and a historical path. Finally, adaptation example cases are presented and life cycles application to successful cases.

Key words: OMT (Object Modelling Technique), UML (Unified Modelling Languaje), life cycles, methodology.

Introducción

Sin importar cualquiera que sea el tipo de software a ser desarrollado sea de sistemas (son programas que sirven a otros programas en el trabajo de desarrollo como compiladores, editores, ..), tiempo real (software encargado de analizar datos del mundo en forma real tales como análisis de datos, control automatizado, monitoreo de datos), gestión (a esta categoría se incluye el software comercial a nivel

empresarial nominas, inventarios), ingeniería y científico (es software que posee un amplio manejo numérico usado en biología, astronomía, CAD), empotrado (software que se encuentra residente en memoria, tales como : controles automáticos en los vehículos, sistemas de background, partes del sistema operativo), computación personal (software comercial de uso local como procesadores de texto, hojas electrónicas, navegadores web, calendarios, agendas, recetarios), inteligencia artificial (software de procesamiento especial sistemas expertos, sistemas basados en el conocimiento, generalmente no usan algoritmos numéricos). Todos los tipos de software mencionados requieren que los analistas, diseñadores y desarrolladores apliquen características y elementos de calidad para que se logren productos a las necesidades del usuario, estas necesidades se comienzan a encontrar un camino de solución a través de la aplicación de estrategias en el proceso, una de ellas es la aplicación de ciclos de vida adecuados a los proyectos y a los grupos de trabajo. Describiremos la forma como se acoplan elementos como calidad, modelos de desarrollo y ciclos de vida, presentando ejemplos.

1. Procesos de desarrollo

1.1 Calidad en la ingeniería del software

En una versión sucinta **la calidad** en la ingeniería del software es un grupo de características que representa la efectividad y la eficiencia de un sistema de información. Es importante enfatizar en dos puntos:

- *Un software de calidad debe ser **eficaz**, es decir, que debe realizar las funciones establecidas, debe ser amigable*. Un usuario debe utilizar el software porque produce resultados confiables, realiza todas las operaciones que se requieren, ejecuta las operaciones en un tiempo aceptado y es fácilmente usado por el grupo de usuarios a quien este dirigido.
- *Un software de calidad debe ser **eficiente**, es decir el costo de su desarrollo tomando todos los recursos y el costo de su operación debe ser tal que las organizaciones involucradas en su desarrollo y uso obtengan el máximo beneficio o por lo menos un beneficio aceptable en un período de tiempo establecido.*

Se define la calidad de software como la ausencia de errores de funcionamiento, la adecuación a las necesidades del usuario, y el alcance de un desempeño apropiado (tiempo, volumen, espacio), además del cumplimiento de los estándares. Los objetivos que la calidad persigue son: La aceptación (utilización real por parte del usuario) y la mantenibilidad (posibilidad y facilidad de corrección, ajuste y modificación durante largo tiempo). Para alcanzar estos objetivos, es necesaria una actitud y compromiso de todo el personal que se encuentre en el desarrollo del proyecto, y en todas y cada una de las etapas (en general, planeación, análisis, diseño, programación, pruebas, mantenimiento) correspondientes al ciclo de vida que se hubiese seleccionado para el proyecto. En forma adicional durante el proceso de aplicación de las metodologías se requiere tener en cuenta:

- Realización de revisiones técnicas formales durante cada etapa.
- Realización de pruebas y revisiones por personas "externas" al proyecto.
- Elaboración de la adecuada documentación del software, y de los cambios.
- Verificación del cumplimiento de los estándares de desarrollo
- Medición permanente de la productividad del proceso y de la calidad de los resultados.
- Desarrollo y ajustes de modelos estadísticos de calidad y productividad.
- Control de la desviación de los promedios de calidad y productividad.

Uno de los elementos que permite dar garantía acerca de la calidad del software es la aplicación de métricas, éstas son medidas estadísticas aplicadas a un software determinado, garantizando calidad así como lo afirma Pressman (1998) "la garantía de calidad del software, es una *actividad de protección* que se aplica a lo largo de todo el proceso de ingeniería del software".

Todos los elementos anteriormente enumerados indican herramientas que se deben tener en cuenta al momento de desarrollar un software, agrupando

en una definición estos elementos se afirma que: un software debe estar desarrollado "en concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software", si cumple los aspectos señalados se puede afirmar que se posee un software de calidad. Teniendo en cuenta esto, se puede afirmar :

- Los requisitos del software son la base de las medidas de la calidad.
- Los estándares especificados definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería del software, si no se distinguen esos criterios no habrá calidad del software.
- Existe un conjunto de requisitos implícitos que a menudo no se mencionan, si no se alcanzan estos requerimientos podría la calidad quedar en entredicho. Los requisitos son llamados por los usuarios finales elementos obvios, los cuales el diseñador no debe dejar pasar sin explicación.

Para estar seguros de las anteriores afirmaciones se tienen en cuenta factores que se pueden medir estos son llamados factores de calidad. Éstos se agrupan en dos bloques así:

- Factores que pueden ser medidos directamente (errores, líneas, tiempo),
- Factores que sólo pueden ser medidos indirectamente (facilidad de uso, mantenimiento)

Para ilustrar el concepto de calidad de manera más profunda, es necesario considerar algunos aspectos fundamentales que caracterizan al software de calidad como son: solidez, exactitud, completitud, mantenibilidad, reutilizabilidad, claridad en la documentación, entre otros que serán descritos a continuación.

1.2 Aspectos básicos de calidad de software

La descripción que se hace de los factores que influyen en un software de calidad se basan principalmente en las ideas presentadas por Dunn (1990), Crosby (1979) y Pressman (1998). Sin embargo, también se han tomado algunos aportes de Bertrand Meyer y Mauricio Fernando Alba (1992).

Dunn (1990) presenta la calidad en el software tomando dos puntos de vista: la calidad en el proceso de desarrollo y la calidad en el producto final, estos dos grupos principales los agrupa en los siguiente aspectos de calidad: *confiabilidad, utilizabilidad, mantenibilidad, y adaptabilidad.* Por su parte, Presuman (1998) describe similares factores de calidad agrupados en tres grupos: calidad en operación, calidad en revisión y calidad en transición.

A continuación se presentan los factores de calidad de acuerdo con el orden dado por Dunn (1990).

1.2.1 Confiabilidad

Este término es necesario que sea separado en varios elementos que permiten darle al software el matiz de fiable. Sus componentes son:

- Completitud
- Consistencia y precisión
- Solidez
- Simplicidad
- Calidad en los procesos de desarrollo
- Seguridad y verificabilidad, estas dos últimas que se determinan con el sistema en uso.

1.2.2 Usabilidad

Si bien es cierto que la confiabilidad es un factor muy importante en la calidad del software también lo es el hecho de que es necesario considerar otros factores como los que se mencionan en esta sección puesto que de nada sirve un software que funcione correcta y confiablemente si el usuario prefiere no utilizarlo.

- Exactitud de los procesos
- Claridad y exactitud de la documentación
- Completitud
- Eficiencia y verificabilidad del software
- Claridad y amigabilidad de la interfaz

1.2.3 Mantenibilidad

Este aspecto de calidad involucra los elementos que simplifican la labor de prevención, corrección o ampliación del código del programa. Retomar un código escrito meses antes es un trabajo dispendioso y agobiante, en especial cuando las aplicaciones no cuentan con la característica a la cual aquí se hace

referencia. Se pueden considerar como atributos de este aspecto:
- Exactitud y claridad en la documentación
- Modularidad (acoplamiento)
- Facilidad de lectura
- Simplicidad

1.2.4 Portabilidad
Es la capacidad que posee un sistema de información que le permite funcionar en diferentes plataformas ya sean hardware o de software.

Otro autor que contribuye con el aspecto de las medidas en el software es McCall, él y sus colegas proponen tres factores de calidad y sus partes así:
Factor 1. Características operativas, relacionadas con las operaciones del producto.
- Corrección
- Fiabilidad
- Eficiencia
- Integridad
- Facilidad de uso

Factor 2. Capacidad de soportar cambios, relacionado con la revisión del producto.
- Facilidad de mantenimiento
- Flexibilidad
- Facilidad de prueba

Factor 3. Adaptabilidad, relacionado con la transición del producto.
- Portabilidad
- Reusabilidad – Reutilizabilidad
- Interoperabilidad

Como se aprecia en la anteriores palabras son muchas las características a tenerse en cuenta al momento de desarrollar software, estos deben aparecer como políticas o lineamientos en los procesos de desarrollo al interior de posproyectos o empresas, ya que se ha llegado al caso colombiano; donde las aseguradoras no realizan amparos para contratos de desarrollo de software; dadas las pocas garantías de entrega exitosa y el gran número de demandas realizadas por la empresas a las compañías desarrolladoras de software, esto llegó a situaciones no deseadas, y una de las razones principales es no poseer **ciclos de desarrollo adecuados** a las necesidades del proyecto y a las formas de trabajo del personal.

Siendo esta preocupación, no solo actual sino histórica se plantean nuevas formas de desarrollo que permitan llevar el ejercicio del desarrollo de software a otro nivel apareciendo los modelos orientados a objetos.

2. Modelos orientados a objetos

2.1 El enfoque orientado a objetos.

El paradigma orientado a objetos se caracteriza por la solución de problemas a través del estudio y representación de los objetos del mundo real y las interacciones entre sí. Coad and Yourdon (1990) representa la orientación a objetos en la ecuación:

Orientado_Objeto = clases y objetos + herencia + comunicación con mensajes cohesión

Coad y Yourdon (1990) consideran las siguientes razones para hacer uso del enfoque orientado a objetos:

- Toda la información está basada en los datos y estos conforman la parte más estable de un sistema y por ende le dan estabilidad a la aplicación. Las estructuras se especifican de forma que sean flexibles al cambio, esto implica que las modificaciones a los requerimientos no afectan la estabilidad del sistema ya que no provocan alteraciones traumáticas a menos que se trate de cambios significativos.
- Los resultados del análisis, diseño e implementación son reutilizables. La representación del mundo real a través del *Análisis Orientado a Objetos (AOO)* lleva a una modularidad formada por paquetes de clases y objetos. Estos a su vez conforman subsistemas reutilizables en futuros proyectos. Es de anotar que la Modularidad aquí llega más allá de dividir un programa en funciones como se hace en las metodologías estructuradas.
- Mejora el análisis y la interacción expertos-dominio "Casos de Uso" del problema. A través del enfoque orientado a objetos se construye un modelo desde la visión del experto que representa el problema en forma más natural, haciendo prevalecer el pensamiento de los usuarios directos del futuro sistema.

- La manera como se agrupan los elementos del mundo con sus operaciones como un todo, incrementa la consistencia entre el modelo y el mundo real.
- Representa explícitamente los elementos comunes. AOO utiliza la herencia para identificar y capitalizar los atributos y servicios comunes.
- Promociona la utilizabilidad (usabilidad).

Otras razones del uso del enfoque orientado a objetos son las siguientes:

- De acuerdo con la experiencia de algunos, se ha notado que se produce menos código y por ende es más fácil de mantener favoreciendo la relación costo beneficio.
- Muchas de las personas que no trabajan con computador encuentran el enfoque orientado a objetos muy natural.

Para Rumbaugh, et al (1995), el enfoque orientado a objetos, "constituye un nueva forma de pensar acerca de problemas, empleando modelos que se han organizado tomando como base conceptos del mundo real", plantean su utilidad en los siguientes aspectos:

- Comprensión del problema
- Comunicación con expertos en esa aplicación
- Modelamiento de empresas
- Preparación de documentos
- Diseño de programas
- Diseño de bases de datos

No se ahondará mucho en lo asociado con las características y metodologías orientadas a objetos, puesto que se trata de presentarlo como la alternativa primaria para los procesos de desarrollo de software actuales, además de la predominancia de herramientas con este enfoque.

3. Aplicación de ciclos de vida

Los modelos actuales de desarrollo de software en las empresas, presentan las siguientes opciones: 1) Se desarrollan los modelos de análisis y diseño, y se contrata una compañía que los implemente e implante; 2) Se contrata una compañía de desarrollo que realice todos los pasos del ciclo de vida; 3) En la empresa se poseen los recursos necesarios para realizar los desarrollos a la medida de las necesidades; 4) La empresa posee en sus objetivos el desarrollo de aplicaciones para la comercialización.

Cualquiera que sea la opción de la empresa, o donde se encuentre, es necesario que la empresa de desarrollo o el departamento de sistemas, informática, IT, o como sea la denominación, debe apoyar los procesos de desarrollo en ciclos de vida eficientes y eficaces que permitan avanzar en forma rápida en las diferentes fases y actividades que los compongan; de forma tal que tanto los usuarios finales como el equipo de desarrollo puedan valorar y verificar la madurez del producto. Para esta valoración se usan herramientas de Ingeniería del software como son las revisiones técnicas formales, pero ¿dónde se deben realizar estas valoraciones? ¿en qué momento del desarrollo? Estas preguntas y otras más que surgen del proceso de desarrollo, se responden en la marcha, y con la creación de políticas adecuadas al momento de poseer grupos de desarrollo en los cuales interactúan muchos profesionales. Para programar esto, se desarrollan formas y metodologías asociadas a la aplicación para cada uno de los ciclos de vida.

Históricamente los procesos de análisis y diseño estaban asociados a procesos de ciclos de vida de desarrollo en los cuales las entregas eran demoradas y estaban limitadas por procesos de análisis y diseño rígidos, como el ciclo de vida clásico, éste y otros promulgaban la necesidad de realizar procesos de análisis y diseño extensos y detallados para finalizar en un producto software implementado por un grupo que poco conocía de los resultados iniciales. Con una preocupación latente de esto se generó una avalancha de modelos, metodología y otras, contando 40 de estas en los años ochenta y noventa.

En esta misma época, hay mayor interés por los modelos orientados a objetos, el desarrollo WEB y otras tecnologías que permitieron avanzar en las características detalladas de los procesos de desarrollo entre ellas y con la bandera en alto aparece OMT (Object Modelling Technique) como la alternativa para unificar los análisis y diseños, la cual fue ampliamente acogida. Finalizando los 90 nos encontramos con UML (Unified Modelling Language) que con las revisiones iniciales dadas a

la metodología y su notación se convierte en la herramienta estándar para presentación de procesos de análisis y diseño.

Las metodologías y notaciones por si solas proveen estrategias que permiten avanzar en el desarrollo de aplicaciones software, estas preparan los espacios para que cualquier proyecto pueda ser puesto en su estándar y se alcancen resultados finales, pero no siempre su aplicación religiosa es la alternativa para el grupo de trabajo, los cliente, los coordinadores de proyecto, entre algunas de las personas involucradas en estos procesos.

Los casos siguientes presentan características propias de aplicación a ciclos de vida conocidos y a otros adaptados con base en las experiencias obtenidas en el ejercicio profesional en 25 proyectos, con los cuales se aplican modelos de análisis, diseño e implementación, todas ellas orientadas a través de modelos orientados a objetos, usando OMT y UML en sus procesos.

3.1 Prototipado rápido

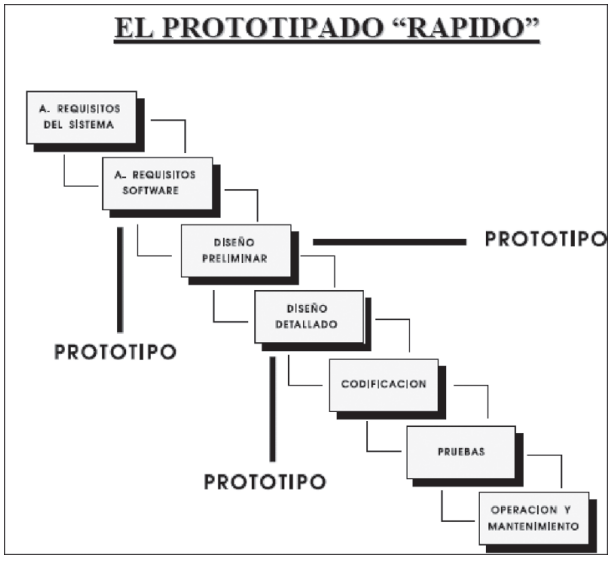

Figura 1. Representación Gráfica de Prototipado Rápido

El ciclo de vida acá planteado se aplicará a la propuesta metodológica, la cual en cada una de sus fases toma elementos del análisis y diseño de OMT y presenta como resultados los diagramas de UML correspondientes, la aplicación se presenta de la siguiente manera:

Requisitos del sistema
- Especificación del problema

Requisitos del software.
- Define del análisis
  - Requisitos del software
  - Modelo estático (Diagramas de clases + diccionarios)
- Define del diseño
  - Plantea modelo de datos inicial (Modelo E/R)
  - Plantea modelo básico de interfaces

Diseño preliminar
- Define del análisis
- Modelo dinámico (estados + secuencia + colaboración)
- Define del diseño
  - Requerimientos de Hw / Sw
  - Diagrama de paquetes + operaciones básicas implementadas funcionando sobre las interfaces ya definidas.

Diseño detallado
- Define del análisis
  - Modelo funcional – casos de uso finales.
- Define del diseño
  - Diseño objetos. (Despliegue + Componentes)
  - Algoritmos + mediciones (De las operaciones principales del sistema)

Codificación
- Modelo de implementación

Pruebas

Operación y mantenimiento

3.2 Fusión

A continuación se presenta una alternativa de ciclo de vida denominado "Fusión", por su mezcla entre los ciclos de vida de prototipos y el de prototipado rápido; ésta propuesta permite que las revisiones técnicas formales sean realizadas en cada uno de las fases de construcción de prototipos del sistema final, y permite guiar el proceso de desarrollo a través de modelos de UML (Unified Modelling Language).

El ciclo de vida planteado, presenta dos elementos fundamentales denotados en dos fases y el desarrollo de cada fase se promueve con el apoyo de prototipos del sistema de información, y documentos de entrega de cada uno de los prototipos.

Este modelo permite al grupo de trabajo, que en cada una de las fases realice refinamientos continuos usando ciclos, representados por las circunferencias, estas revisiones se hacen hasta que se considere adecuado el resultado y se pueda continuar con la fase siguiente, en caso de generar nuevas especificaciones durante el proceso de desarrollo, el modelo permite retomar en el punto donde sea necesario y refinar. A continuación se describen los elementos y resultados de cada fase.

Fase 1

Análisis de requisitos del sistema
- Especificación del problema
  - Descripción del caso de estudio
  - Diagramas representativos del problema(Casos de uso)
- Acta de revisión de los casos de uso

Análisis de requisitos del software.
- Define del análisis
  - Requisitos del software
  - Modelo estático (Diagramas de clases + diccionarios)
- Define del diseño
  - Plantea modelo de datos inicial (Modelo E/R)
  - Plantea modelo básico de interfaces
- Acta de entrega de interfaces

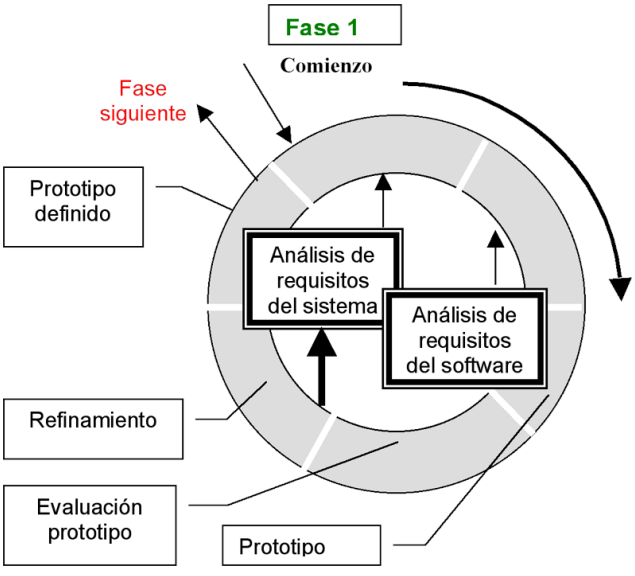

Figura 2. Representación Gráfica Fase 1 - Fusión

Fase 2

Diseño preliminar
- Define del análisis
  - Modelo dinámico (Diagramas de Secuencia)
- Define del diseño
  - Requerimientos de Hw / Sw
  - Diagrama de paquetes + operaciones básicas implementadas funcionando sobre las interfaces ya definidas.
- Acta de validación de las operaciones

Diseño detallado
- Define del análisis
  - Modelo funcional – Casos de uso finales.
- Define del diseño
  - Diseño objetos. (Despliegue + Componentes)
  - Algoritmos (De las operaciones principales del sistema)
- Acta de entrega de prototipo funcional

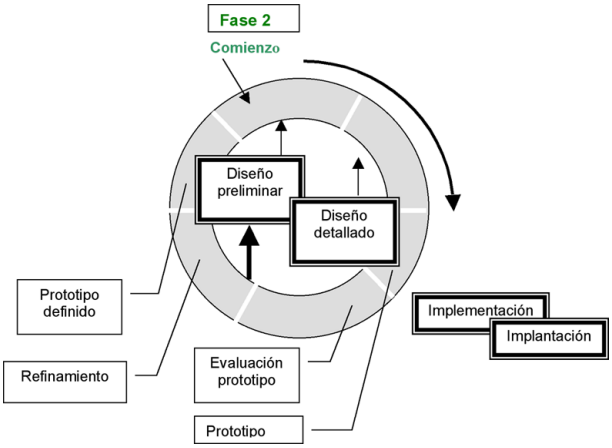

Figura 3. Representación Gráfica Fase 2 - Fusión

Implementación - codificación.
- Modelo de implementación – Son las decisiones finales de codificación, presentadas en términos de un lenguaje de programación.

Implantación - Se presenta en dos elementos:

Pruebas. Es un proceso continuado para las evaluaciones de los prototipos, estas deben ser planeadas con base en las excepciones posibles registradas al momento de trabajar con los casos de uso, además de los casos especiales registrados en el desarrollo del proyecto.

Operación y mantenimiento. Corresponde a los elementos de puesta en marcha y control de sistemas.

Las actas que se presentan en cada una de las fases corresponden a documentos que permitan dar claridad, guía, orden y rigor administrativo al desarrollo del sistema.

3.3 Prototipo incremental

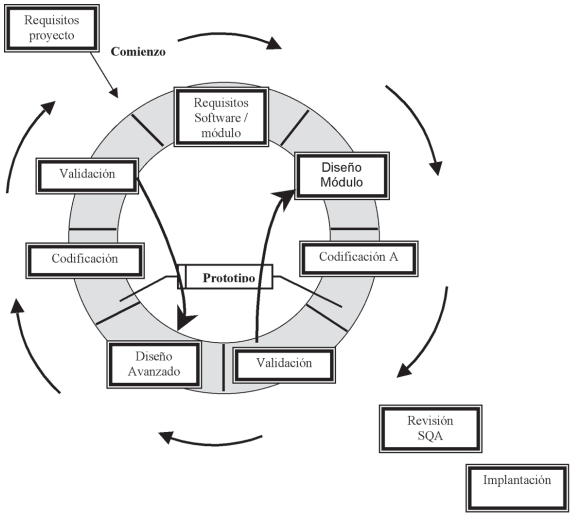

Figura 4. Representación Gráfica Ciclo de Prototipo Incremental

Esta propuesta es una referencia al ciclo de vida por prototipos y al modelo incremental, al iniciar el proceso de requisitos del proyecto, éste permite ser detallado en exceso en las necesidades y requerimientos del sistema con este insumo se procede a trabajar desarrollando los diferentes módulos del sistema; importante denotar que cada vez que se realiza un ciclo completo, se esta completando un módulo del sistema. Al momento de realizar validaciones se puede saltar al diseño del módulo o al diseño avanzado para que se proceda con las modificaciones necesarias, o si es necesario retornar a los requerimientos.

A continuación se detallan las acciones, de esta propuesta de ciclo de vida:

Requisitos del proyecto
- Enunciado detallado
- Diagrama de clases
- Diccionarios
- Modelo de datos

Requisitos del software (módulo)
- Corresponde a determinar las necesidades del módulo o subsistema, además de requisitos software y hardware necesarios, justificando su aparición.

Diseño del módulo
- Corresponde al diagrama de paquetes, del módulo que se esta resolviendo. (siendo claro en cuales clases son propias del módulo, cuales son compartidas, y cuales son de otros módulos para obtener datos e información)

Codificación A
- Corresponde al desarrollo del prototipo funcional, con las pantallas iniciales definidas y con las operaciones básicas implementadas.

Validación
- Al prototipo es necesario ponerlo a prueba y realizarle operaciones y acciones para validar su aplicación, esta actividad es responsabilidad del SQA, y evaluadores si se poseen.

Diseño avanzado
- Corresponde a las actividades de construir:
  - Diagrama de estados – Se realizan sobre las clases propias que posean comportamiento dinámico complejo, es decir poseen relaciones con tres o mas clases.
  - Diagramas de secuencia – Este se realiza solo a las operaciones principales del módulo.

Codificación B
- Este prototipo corresponde a la implementación de las operaciones principales del módulo, ya diseñadas en la etapa anterior, es común que al momento de implementar se hagan modificaciones es importante que estas queden reflejadas en los diagramas.

Validación
- Al prototipo es necesario ponerlo a prueba y realizarle operaciones y acciones para validar su aplicación, esta actividad es responsabilidad del SQA ,y evaluadores si se poseen.

Ya entregado el módulo a producción, es necesario completar elementos asociados con la revisión del SQA, esta revisión esta asociada con la integración del módulo, la garantía de modularidad, la usabilidad y otros elementos de ingeniería de software que se

tengan previstos en el proyecto. Se desarrollan: diagramas de actividad del módulo y revisión de las características de codificación.

Al ser aprobado el módulo se realiza el diagrama de componentes, para reconocer su implementación detallada.

Estos son algunos ejemplos aplicados en diferentes empresas que han dado resultados exitosos, luego de romper la barrera impuesta por los informáticos al encontrarse que muchas veces al igual que el usuario final solo quieren conocer la aplicación terminada.

Finalmente, se invita a las empresas y entidades educativas a fortalecer procesos en el área de ingeniería de software y desarrollo para fortalecer esta naciente empresa en Latinoamérica.

Conclusiones

Las anteriores propuestas son resultados de aplicaciones exitosas en algunos proyectos de desarrollo, luego de estas aplicaciones se determina:

- Al momento de iniciar procesos de desarrollo se debe elegir el modelo (ciclo de vida) que mejor se adapte a las condiciones del proyecto (complejidad, tiempo, entrega final, comunicación, requerimientos), entre otros elementos.
- Los ciclos de vida son herramientas de apoyo a la garantía de la calidad de software provista por las metodologías de análisis y diseño.
- Cualquiera que sea la decisión acerca de los ciclos de vida el responsable de SQA debe revisar y aplicar las políticas de calidad definidas para cada uno de los resultados de cada etapa o fase aplicada.

Referencias

ALBA C., Mauricio Fernando (1992). Calidad en la Producción de Software En : Calidad de Software. Primer Congreso Nacional de Estudiantes de Ingeniería de Sistemas. Universidad Autónoma de Manizales.

COAD, Peter y YOURDON, Edward (1990). Object Oriented Análisis. Prentice – Hall.

CROSBY, Philip (1979). Quality is Free. Mc Graw Hill.

MEYER, Bertrand. Object Oriented Software Construction

PRESSMAN, Roger S. (1998). Ingeniería del Software. Un enfoque practico. Tercera edición, McGraw Hill.

RUMBAUGH, James. et al. (1995). Modelado y diseño orientado a objetos.

Sobre los autores

Juan Pablo Giraldo Rendón
Estudiante Doctorado Ingeniería Informática Universidad Pontificia de Salamanca – Madrid.
Ingeniero de Sistemas y docente del programa de Ingeniería de Sistemas y Telecomunicaciones de la Universidad de Manizales (Colombia). Investigador Grupo de Investigación y Desarrollo en Informática y Telecomunicaciones, Escalafonado en Colciencias, Categoría A (Colciencias, Colombia). Investigador proyecto "Construcción participativa de un currículo basado en problemas para la formación por ciclos de tecnólogos e ingenieros de sistemas y telecomunicaciones en la Facultad de Ingeniería de la Universidad de Manizales", (Colciencias, Colombia).
jpgiraldo@umanizales.edu.co

Carlos Alberto Cortés Carrillo
Magíster en Gerencia del Talento Humano, Especialista en Telecomunicaciones, Ingeniero de Sistemas. Profesional en Ciencias de la Información y la Documentación. Decano Facultad de Ingeniería – Universidad de Manizales (Colombia). Investigador Grupo de Investigación y Desarrollo en Informática y Telecomunicaciones, Categoría A (Colciencias, Colombia). Investigador Grupo de Investigación y Desarrollo en Geomática y Medio Ambiente, Categoría B, (Colciencias, Colombia).Investigador proyecto "Construcción participativa de un currículo basado en problemas para la formación por ciclos de tecnólogos e ingenieros de sistemas y telecomunicaciones en la Facultad de Ingeniería de la Universidad de Manizales", (Colciencias, Colombia). Coordinador especialización Telecomunicaciones. Convenio Universidad de Manizales – UNAB. Coordinador Doctorado en Ingeniería Informática. Convenio Universidad de Manizales – Universidad Pontificia de Salamanca España. ccortes@umanizales.edu.co
Universidad de Manizales: Cra 9 Nº 19 – 03. Tel. +57 +6 884 14 50 Ext. 286 – 249. Fax +57 +6 883 27 18. Manizales, Colombia.