Ciclos de vida

Análisis de Sistemas, F.R.B.A. U.T.N.

Abril de 2009

Ing. Rodrigo N. Uroz

Introducción

Desde que a fines de la década de los 60 comenzó a hablarse de “Ingeniería de Software” como una disciplina y profesión en sí misma se ha suscitado un debate acerca de si realmente merece ser llamada Ingeniería.

El fundamento de los detractores no se basa en que el resto de las Ingenierías tienen muchos más años de historia, lo cual no deja de ser un punto fuerte, sino en que la Ingeniería de Software está lejos de tener métodos tan exactos y repetitivos como en las demás.

Desde que Von Neumann ideó la arquitectura que lleva su nombre y que todavía hoy utilizamos, se creó la diferencia entre hardware y software. Eso llevó a que la programación de computadoras dejara de ser una tarea exclusiva de los Ingenieros en Electrónica, y comenzara a tener lugar la abstracción y la programación lógica, que podía ser atacada por muchas personas que no tuvieran conocimientos de electrónica.

Desde entonces la popularidad de la informática fue creciendo y cada vez más organizaciones comenzaron a utilizarla, con la consecuencia de que cada vez más profesionales eran necesarios. De todas maneras, nunca se pudieron alcanzar niveles de eficiencia en la construcción de Software similares a los conseguidos en otras disciplinas, y de hecho hubieron estrepitosos fracasos en la industria, uno detrás de otro. Por eso es que hay muchas dudas con respecto a su carácter de Ingeniería. Varias causas se pueden encontrar que expliquen esto.

La primera de ella es que la informática ha penetrado tanto en nuestras vidas, que muchas personas tienen acceso a la misma diariamente. Además, se ha avanzado mucho en las herramientas de construcción y esto hace que hoy en día sea mucho más fácil y lleve menos tiempo programar una computadora que construir un motor de combustión o un circuito integrado en una plaqueta. Esto significa que la mayoría de la gente que participa en la construcción de Software no tiene una edudación formal en la materia e ignora muchos de los avances hechos en la disciplina. Todavía hoy se cometen errores que ya se conocían en la década del 70.

La segunda causa y también muy importante es que en la construcción del Software intervienen muchos factores distintos y uno de ellos es de extrema complejidad, como es el ser humano.

Dentro de todo este caos, uno de los desarrollos de la Ingeniería de Software viene a traer un poco de control, y es el estudio de los distintos ciclos de vida. En

cualquier proceso productivo es indispensable y obligado un esquema de la producción, que indica las etapas por las que va pasando un producto hasta estar finalizado. Esa misma analogía se introduce en la construcción de un producto de Software.

Ciclos de vida

Las actividades de construcción de Software se estructuran dentro de lo que se denomina “ciclo de vida”, que busca atacar las distintas etapas por las que va pasando un Software.

Un ciclo de vida no sólo especifica las etapas por las que pasa el proceso de construcción, sino también los criterios de transición entre etapas.

Modelo en cascada puro

Este modelo es el abuelo de todos los ciclos de vida. Fue el primer modelo conocido, y se originó en los diagramas de secuencia de los procesos industriales y en los diagramas de Gantt. Con este modelo comienza la separación de 3 etapas principales, definición, desarrollo y mantenimiento.

La fase de definición se centra en qué es lo que hay que hacer. En el desarrollo se busca determinar el cómo. Por último, el mantenimiento se centra en el cambio, que es inherente a cualquier actividad humana.

Este ciclo de vida es muy estricto, en el sentido en que no comienza una etapa hasta que está completamente terminada y aprobada la etapa anterior. Surgió con la premisa de que el usuario conoce de antemano con mucho detalle el Sistema que desea construir y estos requerimientos son estables. Tiene la característica de que no se tiene un producto de Software hasta las últimas etapas del ciclo. Como se puede ver, no es un ciclo de vida flexible.

Con el tiempo otras etapas se han ido agregando, y no existe un consenso en la cantidad de etapas que tiene. Aquí se detallan las más habituales y aceptadas.

1. Estudio preliminar: también conocido como reconocimiento, tiene como objetivo conocer a la organización, el ambiente en dónde se debe construir el Sistema. Es un primer acercamiento a la problemática. Finaliza con un diagnóstico
2. Relevamiento: en esta etapa se buscan capturar las necesidades y documentarlas.
3. Estudio de factibilidad: a esta altura ya se está en condiciones de estudiar distintas alternativas de solución y buscar la más adecuada para el cliente.
4. Análisis: la alternativa seleccionada es la que se empieza a analizar. Esta etapa busca encontrar una solución lógica al problema. Determinar las características que tiene que tener la solución, es decir, “qué” se debe construir.
5. Diseño: en esta etapa se define cómo se va a llevar a cabo la solución lógica esbozada en la etapa de Análisis, acá se determina el “cómo”. Es un modelo

físico de solución
6. Implementación: se construye (programa e implementa) la solución de diseño.
7. Mantenimiento: se atacan los cambios y problemas que ocurran en el sistema. Hay 3 tipos de mantenimiento: adaptativo, correctivo y perfectivo.
8. Retiro: cuando el sistema llega al final de su vida útil, se retira y reemplaza por otro.

Code and Fix

Este ciclo de vida es el utilizado cuando no se elige ningún otro tipo de ciclo de vida. Es, lamentablemente, el ciclo de vida más utilizado, incluso a veces de manera encubierta.

En esta metodología se comienza con una idea general de lo que se quiere construir, luego sin ninguna restricción se utilizan distintos enfoques para llegar a tener un producto de Software.

El aspecto positivo de este ciclo de vida es que no gasta tiempo en planificación, documentación, calidad, etc. Como se comienza en seguida con la codificación, tiene una alta visibilidad y se puede mostrar un producto en progreso muy temprano en el desarrollo. Además, no requiere ninguna experiencia, cualquiera puede utilizar este ciclo de vida.

Modelo en espiral

Este es uno de los modelos más complejos y está basado en la determinación de los riesgos. Lo que hace este modelo es dividir el proyecto en sub-proyectos. De ahí la analogía con la espiral, en cada iteración de la espiral el sistema va incrementando su tamaño.

Cada iteración entonces tiene sus etapas definidas, y se comienzan por definir los riesgos inherentes a la iteración, teniendo al final un entregable que sirve para continuar con la siguiente etapa y los riesgos fueron manejados.

La ventaja es que se pueden minimizar los riesgos lo más que se quiera, y se puede obtener una especificación y un producto de mucha calidad, pero a costa de ser muy costoso de implementar y no menos complicado.

Modelo en cascada modificado

Conocido como Sashimi, lo que agrega este modelo por sobre el de cascada puro es que las etapas se pueden solapar. Es decir, la etapa siguiente puede comenzar incluso mucho antes de que la precedente esté terminada. Esto permite someter a prueba y juicio lo antes posible lo que uno está haciendo en etapas posteriores, y si debe modificar algo puede volver y hacerlo.

Aquí es importante recordar la frase de Brooks: “Los errores en la especificación son inmensamente más costosos a los errores en la construcción”. Un corolario de esto sería: “Al descubrir un error en el producto, cuanto más temprano en el ciclo de vida se haya originado ese error, más costoso será repararlo”.

La desventaja de este modelo es que no tiene hitos tan definidos, ya que es más difícil determinar cuándo terminó una etapa.

Prototipado evolutivo

En este ciclo de vida, se captura un conjunto inicial de necesidades y se implantan rápidamente con la intención de expandirlas a medida que vaya aumentando la comprensión del sistema. Con el tiempo el prototipo se acepta y se termina de construir el sistema.

La ventaja es que el usuario no tiene que conocer al principio todos los requerimientos, puede ir descubriéndolos a medida que el proyecto avanza.

Como punto negativo se puede señalar que no se puede saber de antemano cuánto va a llevar el proyecto.

Prototipado desechable

Una variante del Prototipado Evolutivo se conoce como Prototipado Desechable. En este ciclo de vida, se construye un prototipo del sistema lo antes posible para entender y validar los requerimientos, pero una vez que se refinan los requerimientos y se entiende el sistema que se quiere construir, el prototipo se elimina, y se comienza a construir el sistema desde cero.

Agile development

En 2001, luego de la publicación de un libro llamado “Extreme Programming”, se creó un manifiesto denominado “Agile Development”, que busca mostrar nuevas técnicas de construcción de software. Se basa en los siguientes principios:

- A los individuos y su interacción, por encima de los procesos y las herramientas.
- El software que funciona, por encima de la documentación exhaustiva.
- La colaboración con el cliente, por encima de la negociación contractual.
- La respuesta al cambio, por encima del seguimiento de un plan.

Conclusiones

En el mencionado ensayo “No Silver Bullet”, Brooks afirma que la construcción de software es un proceso complejo por naturaleza. Esta construcción tiene dos tipos de tareas: esenciales y accidentales.

Las tareas esenciales son aquellas inherentes a la naturaleza del software y de las necesidades del usuario. Se refieren a construir un modelo abstracto de solución. Las tareas accidentales, en cambio, son aquellas que existen en relación a la concreción del producto, a cómo es llevado a la práctica.

El producto de Software a su vez tiene 4 características que lo distinguen del resto de los procesos de construcción de otras Ingenierías donde existe más previsibilidad.

- Complejidad: No hay 2 partes iguales, no hay elementos repetitivos, a diferencia de otras construcciones del hombre. El Software tiene una enorme cantidad de estados posibles, lo cual dificulta todo el proceso de diseño, construcción y pruebas.
- Conformidad: A menudo el producto de Software tiene que adherirse a estándares, normas, leyes, interfaces, y todo esto no tiene que ver con los requerimientos del usuario, sino que son consecuencias del ambiente o entorno en donde el sistema debe interactuar.
- Cambiabilidad: Las cosas construidas muy poco frecuentemente sufren cambios. Los cambios al Software son más habituales, porque existe la sensación de que pueden ser cambiados más fácilmente, a bajo costo.
- Invisibilidad: El Software es una construcción abstracta, y por lo tanto su validación ocurre en el plano de las ideas. No puede examinarse un producto concreto en el plano físico para encontrar fallas.

Por todo esto es importante reconocer que construir un Sistema de Información es extremadamente complejo, no sólo determinar qué construir, sino construirlo adecuadamente también, a bajo costo.

Como dice Brooks: “Lo difícil de construir Software es decidir qué decir, no decirlo”.

Pero algo que ayuda a este proceso es elegir un ciclo de vida adecuado. Hay que estudiar las características y el contexto del proyecto y determinar qué ciclo de vida es el más apropiado.

Se puede concluir diciendo que hay una receta para el desastre en la construcción de un Software, y es no elegir ningún ciclo de vida.

Referencias

1. Artículo sobre Ingeniería de Software, http://en.wikipedia.org/wiki/Software_engineering. Abril de 2009
2. Artículo sobre Manifiesto Ágil, http://es.wikipedia.org/wiki/Manifiesto_%C3%A1gil. Abril de 2009
3. The Mythical Man-Month. Anniversary edition. Frederick Brooks. 1995 (sobre la versión de 1975).
4. Análisis Estructurado Moderno. Edward Yourdon. 1989.
5. Rapid Development. Steve McConnell.