FASES DE PROCESO DE PROGRAMACION

El proceso de la creación de software requiere el uso de una metodología sistemática de desarrollo que permita un acercamiento gradual a la solución del problema que se intenta resolver. Esta metodología, llamada *Ciclo de Desarrollo del Software*, consta de una serie de pasos lógicos secuenciales denominados *Fases*, las cuales son el tema de este artículo. Aunque es posible crear programas sin la aplicación de esta metodología, el producto resultante carece de los beneficios que provee la utilización de este enfoque.

Las Fases de la Programación son:

1. **Definición del problema**
2. **Análisis del problema**
3. **Diseño de la solución**
4. **Codificación**
5. **Prueba y Depuración (Puesta a Punto o Testing)**
6. **Documentación**
7. **Implementación (Producción)**
8. **Mantenimiento**

Aunque el proceso de crear software es esencialmente un proceso creativo, el seguir esta serie de pasos lógicos conduce a la obtención de programas de mayor calidad. Es muy común que los principiantes se salten algunos pasos de esta metodología por desconocimiento o pereza, y procedan directo a la codificación de los programas. Esta práctica no sólo es incorrecta, sino que hace perder tiempo, dinero y esfuerzo. Aún los programadores experimentados y los profesionales utilizan esta metodología en el desarrollo de su programas. Los resultados que se obtienen con su aplicación son más confiables, rápidos y seguros que los obtenidos mediante prácticas incorrectas y desordenadas.

1. Definición del Problema

Consiste en la obtención sin ambigüedades de una visión general y clara del problema. Ayuda a identificar los elementos claves del problema y los de la futura solución, así como fijar los límites de los mismos basados en su planteamiento textual sobre el papel. Un problema mal planteado, incompleto o mal comprendido es un mal inicio para la programación. Las respuestas a las siguientes preguntas son claves para la correcta definición de un problema:

- ¿Qué entradas se requieren, de qué tipo, en qué orden y qué cantidad?
- ¿Qué salidas se desean, de qué tipo, en qué orden y qué cantidad?
- ¿Qué método(s) o fórmula(s) produce(n), o puede(n) producir las salidas deseadas?

Dependiendo de qué tan precisas sean las respuestas a esas preguntas, así será la definición del problema, sobre todo en cuanto al orden explícito de las entradas y las salidas esperadas. Mientras no se comprenda con claridad el problema por resolver no puede pasarse a la fase siguiente.

2. Análisis del Problema

Es la comprensión a fondo del problema y sus detalles y es un requisito para lograr una solución eficaz. Es precesamente en esta fase donde se definen formal y correctamente **la Entrada que recibirá el programa** (datos o materia prima), **la Salida que producirá** (información o resultados) y **el Proceso necesario para su solución** (el método para convertir los datos de entrada en información de salida). Cada uno de estos aspectos coincide respectivamente con las preguntas planteadas en la fase de Definición del Problema. A este enfoque se le conoce comúnmente como **E-P-S (Entrada-Proceso-Salida)**.

3. Diseño de la solución

En esta fase se diseña la lógica de la solución a usar, o sea, cómo hará el programa la tarea que se desea automatizar usando los datos de entrada para generar los datos de salida,

enfatizándose los diseños limpios, sencillos y claros. Pueden plantearse diferentes alternativas de solución al problema y elegir la más adecuada, la que produzca los resultados esperados en el menor tiempo y al menor costo. El proceso de diseño se realiza en dos pasos:

3.1 Elaboración del Algoritmo

Un ***algoritmo*** es una secuencia lógica y cronológica de pasos encaminados a resolver un problema. Las acciones básicas que puede llevar a cabo un algoritmo son: pedir datos, desplegar datos, evaluar condiciones y ejecutar operaciones.

Los programas se estructuran a partir de los algoritmos, los cuales se pueden escribir utilizando la técnica convencional del pseudocódigo (mezcla de lenguaje común, términos técnicos de computación, símbolos y palabras reservadas de algún lenguaje de programación) y los diagramas de flujo (flujogramas) que son la representación gráfica de un algoritmo, plasmados en papel para su estudio. En el caso de emplear diagramas de flujo pueden emplearse herramientas de software tales como el DFD v1.0. Si se opta por el pseudocódigo, se recomienda escribirlos a doble interlínea para efecto de facilitar modificaciones o adición de acciones no consideradas y dotarlos de las siguientes características.

3.1.1 Características de los Algoritmos

- **Un algoritmo debe tener un punto de inicio** o partida.
- **Debe ser preciso** e indicar el orden de realización de cada paso.
- **Debe estar bien definido**. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- **Debe ser finito** (tener un número finito de pasos). Si se sigue un algoritmo, se debe terminar en algún momento.

La definición de un algoritmo debe describir con claridad las tres partes fundamentales del problema: Entrada, Proceso y Salida encontradas en las fases de Definición y Análisis del problema.

Se deja sentado que todos los programas empiezan primero en papel, no directamente frente a la computadora. Aún los programadores más experimentados plasman en papel sus ideas y soluciones antes de programarlas. Pero es común que los novatos vayan directamente a la computadora sin haber siquiera leído bien el problema o pensado siquiera el algoritmo. Eventualmente podrán concluir el programa y alcanzar una solución, pero sólo después de probar diferentes ideas, hacer miles de cambios y perder gran cantidad de tiempo y esfuerzo. Los que se toman tiempo para analizar el problema, pensar y plasmar su solución en papel mediante un algoritmo tendrán un tiempo de respuesta (el tiempo para obtener el programa terminado) mucho menor, y se convierten en mejores programadores. Es un hecho.

Los problemas complejos pueden solucionarse más eficazmente utilizando el método "Divide y Vencerás", el que consiste en fraccionar un problema complejo en otros más simples y más fáciles de solucionar. Esto conduce a la Modulación del programa auxiliado por el método de diseño Top-Down o Descendente en el que se da un refinamiento de los pasos del algoritmo. De hecho, el enfoque E-P-S mencionado anteriormente es un buen ejemplo de esto, pues al concentrarse en resolver cada uno de los tres aspectos del enfoque de manera independiente se logrará la solución del problema completo.

3.2 Realización de Pruebas de Escritorio

O sea, comprobaciones a mano del algoritmo planteado (en pseudocódigo o en diagrama de flujo) con datos y resultados de prueba conocidos, papel, lápiz y calculadora (si es necesaria) para simular su ejecución y evaluar su correcta operación. Si la lógica es correcta, los resultados serán satisfactorios. Si no, el algoritmo deberá modificarse y volverlo a probar hasta que esté correcto. Algunos programas no son fáciles de probar a mano por su complejidad y/o tamaño, pero en tu ayuda está la verificación durante la programación inicial (la creación del algoritmo), el trace and debugging (rastreo y detección de errores) automático que incorporan los lenguajes de compiladores de los lenguajes de programación y otras técnicas. Se hace notar que éste acápite depende de los anteriores. Si la definición y el análisis son errados, el diseño del programa también lo será, por lo que se tendrá que rehacer, retrocediendo quizá hasta la fase de Definición del problema.

4. Codificación

En este paso se traduce el algoritmo ya estructurado, verificado y comprobado a mano, al lenguaje de programación que vaya a utilizarse. Sólo se convierten las acciones del algoritmo en instrucciones de computadora usando la sintaxis de un lenguaje particular, pero requiere de conocimientos del lenguaje y de sumo cuidado en la colocación de las instrucciones, las que deben apegarse y seguir fielmente a la lógica del algoritmo y la semántica y sintaxis del lenguaje.

La digitación, el acto de teclear el algoritmo codificado, se lleva a cabo para almacenar el programa en la memoria de la computadora (virtual o física) y pueda ser aceptado por esta. Con frecuencia los programadores realizan la codificación y la digitación al mismo tiempo a fin de ahorrar tiempo, pero esto puede conducir a errores debido a la pérdida de concentración que implica el uso de un editor.

La compilación, o corrección de los errores sintácticos y semánticos del código, es la eliminación de los errores "gramaticales" según las reglas de construcción de instrucciones particulares del propio lenguaje (la sintaxis). Puede hacerse a medida que se traduce, pero es mejor al final para no perder la secuencia de la codificación. Al terminar debe tenerse el código libre de los errores antes mencionados.

Para realizar la compilación puede hacerse uso de un **compilador**, el cual es un programa especial que analiza todo el código fuente y detecta los errores antes mencionados ocasionados durante la codificación o la digitación. Las fallas de lógica que puedan existir en nuestro programa no son detectadas por este software. Los errores que sí son evidenciados por el compilador deben corregirse modificando el programa fuente.

5. Prueba y Depuración (Puesta a Punto o Testing)

Una vez compilado el programa, este es sometido a pruebas a fin de determinar si resuelve o no el problema planteado en forma satisfactoria. Para ello le suministramos datos de prueba, como lo hicimos en la prueba de escritorio. El programa codificado y compilado no garantiza que funcione correctamente. Debe depurarse (librarse de errores de lógica o de ejecución) realizando corridas de prueba continuas con datos y respuestas conocidas como lo hicimos en la prueba de escritorio, verificando todas las posibles alternativas del programa y sus respuestas y haciendo el mayor número de variantes con sus combinaciones, a fin de determinar si resuelve o no el problema planteado en forma satisfactoria.

Las pruebas que se aplican al programa son de diversa índole y generalmente dependen del tipo de problema que se está resolviendo. Comúnmente se inicia la prueba de un programa introduciendo datos válidos, inválidos e incongruentes y observando como reacciona en cada ocasión.

Los resultados obtenidos en las pruebas pueden ser cualquiera de los siguientes:

a. La lógica del programa esta bien, pero hay errores sencillos, los cuales los corregimos eliminando o modificando algunas instrucciones o incluyendo nuevas.
b. Hay errores ocasionados por fallas en la lógica, lo que nos obliga a regresar a las fases de Diseño y Codificación para revisión y modificación del diagrama.
c. Hay errores muy graves y lo más aconsejable es que regresemos a la fase 2 para analizar nuevamente el problema, y repetir todo el proceso.
d. No hay errores y los resultados son los esperados. En este caso guardamos el programa permanentemente en un medio de almacenamiento.

Puede ser necesario en la mayoría de los casos retroceder a fases previas de desarrollo, revisar el algoritmo otra vez en caso de errores de análisis y/o lógica (que son los más difíciles de detectar, a diferencia de los de sintaxis y semántica), realizar ajustes al código y una serie de nuevas ejecuciones de prueba para que el programa funcione correctamente. Si no existen errores en el programa, puede entenderse la depuración como una etapa de refinamiento en la que se ajustan detalles para optimizar el desempeño del programa.

Si se está automatizando alguna tarea manual, es común poner a funcionar por un tiempo y de forma paralela ambas alternativas, a fin de comparar las salidas de ambas y adquirir confianza en la solución automatizada.

6. Documentación

Es la fase más ignorada por la mayoría de los programadores noveles, por razones de tiempo, costos o simple pereza. Pero no documentar los programas es un mal hábito en programación y un gran error. Será muy difícil a los usuarios entender un programa si no cuentan con un manual de operaciones (el Manual de Usuario). También para los programadores que necesiten darle mantenimiento o hacerle modificaciones si no existe ninguna documentación acerca de sus fases de desarrollo. Incluso será difícil de entender para el mismo autor, algún tiempo después.

La documentación es la guía o comunicación escrita en sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas y sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones (mantenimiento). Recoge todos los elementos encontrados y material creado en las diferentes fases del desarrollo, además de las normas de instalación o las recomendaciones para la ejecución del programa.

La documentación se divide en tres partes:

- Documentación Interna
- Documentación Externa
- Manual del Usuario

Documentación Interna: Son los comentarios que se añaden al código fuente para clarificarlo.

Documentación Externa: Es todo el material creado y empleado en las diferentes fases del desarrollo del programa. Incluye:

- Descripción del Problema
- Narrativo con la descripción de la solución
- Autor(s)
- Algoritmo (diagrama de flujo y/o pseudocódigo)
- Código Fuente (programa)
- Relación de los elementos utilizados en el programa, cada uno con su respectiva función
- Limitaciones del programa

Manual del Usuario: Describe paso a paso la manera como funciona el programa, con el fin de que los usuarios pueda operarlo correctamente y obtener los resultados deseados.

7. Implementación (Producción)

El programa ya probado, revisado y mejorado se considera terminado y puede utilizarse con un alto grado de confianza para resolver los problemas que dieron origen a su creación. Si se está automatizando alguna tarea manual, ésta última se desecha para emplear solamente la programa.

8. Mantenimiento

Es posible que el programa deba revisarse cada cierto tiempo para ajustes. Estos cambios pueden ser por la dinámica del problema, por la naturaleza del código, las exigencias del tiempo o las modernas necesidades que surgen frecuentemente, por lo que se considera que ningún programa es estático.