UNIVERSIDAD DE PANAMÁ
FACULTAD DE CIENCIAS NATURALES Y EXACTAS
ESCUELA DE MATEMÁTICA
CENTRO REGIONAL UNIVERSITARIO DE VERAGUAS

MONOGRAFÍA:

EL USO DE LAS COMPUTADORAS EN LA SOLUCIÓN DE PROBLEMAS

PRESENTA:

RAÚL E. DUTARI D.

1995

TABLA DE CONTENIDOS

INTRODUCCIÓN

La computadora y sus tecnologías colaterales, en el poco tiempo de existencia que tienen en la práctica (unos cincuenta años), han sido uno de los factores de cambio más importantes de nuestra sociedad. Nuestra sociedad admira la facilidad que tienen estas máquinas para facilitar los procesos que realiza el hombre en su vida diaria.

Sin embargo, es notorio el desconocimiento generalizado acerca de lo que se debe hacer para que la computadora pueda facilitarnos las cosas. Poca gente sabe que una computadora debe “PROGRAMARSE ADECUADAMENTE” para que realice las cosas que tanto nos maravillan. Además, de los que conocen esta verdad, solo una fracción sabe con alguna certeza los pasos que se deben seguir para programar adecuadamente a la máquina.

Esta monografía pretende precisamente ilustrar, en términos generales, el proceso que se debe seguir para lograr que un problema se resuelva, dentro de lo posible, haciendo uso de la computadora.

1. Observaciones Preliminares.

Cuando programamos una computadora, no hacemos más que señalar, a la máquina, una serie de órdenes o instrucciones, que ella realizará exactamente, para procesar la información que les suministramos.

El diseño de un programa de computadora (también llamado sistema computacional, aplicación, o simplemente, sistema), es un proceso delicado. En él, en principio, le señalamos a la máquina, exactamente, cuáles son los procesos que debe seguir para el procesamiento de la información.

Es decir, “NOSOTROS” le señalamos a “LA COMPUTADORA” lo que debe hacer. Quién usa la computadora, tiene la responsabilidad de señalarle “EXACTAMENTE”, lo que debe hacer. La máquina siempre realizará lo que programemos, y solamente eso. Todo lo que hagamos usando computadoras debe enmarcarse, necesariamente, dentro de este lineamiento fundamental.

La programación de computadoras es, para la mayoría de las personas, un tema misterioso. No se atreven a incursionar en él, tal vez porque temen a lo desconocido. Esta monografía pretende, precisamente, darnos algunas luces acerca de esta materia.

2. La Creación De Programas Para Computadoras: Un Proceso Con Siete Pasos.

La creación de un programa de computadora significa más que el simple trabajo mecánico del hombre y la máquina. Constituye la culminación de todo un proceso complejo, que involucra a los dos entes, aportando cada uno su respectiva parte en la realización de la tarea.

Esta tarea ha sido sistematizada por los especialistas del área, buscando optimizar la eficiencia de los programas producidos. La sistematización comprende un proceso que consta de siete pasos. Estos son:

⇒ Definición del problema.

⇒ Selección del método de solución.

⇒ Creación del algoritmo.

⇒ Programación del algoritmo.

⇒ Depuración y documentación del programa.

⇒ Validación de la solución.

⇒ Instalación, producción y mantenimiento del programa.

La denominación exacta de cada uno de ellos, así como el número de pasos involucrados, generalmente varía de un autor a otro. Sin embargo, en esencia, son la “APLICACIÓN DEL MÉTODO CIENTÍFICO” a la solución de un problema práctico.

A continuación, nos referiremos a cada uno de estos pasos con más detenimiento.

2.1. Definición Del Problema.

La definición o enfoque del problema es el primer paso en el diseño de un programa de computadora. Consiste, fundamentalmente, en definir cual es la “SITUACIÓN QUE INTENTAMOS RESOLVER”.

La definición del problema debe ser tan clara y precisa como sea posible, de modo que no pueda ser malinterpretada por otras personas. Dicho requisito es muy difícil de lograr. Sin embargo, este paso fundamenta todo lo que se hará en los otros pasos. Luego, no se puede avanzar en el proceso sin contar con esa definición.

La importancia que reviste a esta etapa es enorme. Si el programador no entiende bien el problema que enfrenta, es prácticamente imposible que la computadora lo comprenda mejor. Ella no es capaz de señalar la solución que buscamos, sin embargo, está diseñada para facilitar el estudio de los casos que analizamos, al procesar la información suministrada con gran rapidez y exactitud.

Generalmente, no obtendremos la comprensión absoluta del problema en estudio, con el primer intento. Es necesario realizar un proceso de tanteos y aproximaciones paulatinas al dominio de la situación que analizamos. Es más, generalmente, debemos retroceder a las etapas iniciales del proceso, debido a un detalle que se planteó ambiguamente al definir el problema. El retroceso, al inicio del diseño, es obligatorio, aunque nos encontremos en sus etapas finales.

Debemos enfocar la definición del problema, a través de las siguientes caracterizaciones:

⇒ El modelo de situación típica que se estudiará.

⇒ Las limitaciones impuestas.

⇒ Los datos con que se cuenta.

⇒ La información que se espera obtener.

⇒ Las metas u objetivos a lograr al resolver el problema (este aspecto es sumamente importante).

⇒ Las alternativas disponibles para lograrlas.

⇒ El balance de los criterios involucrados en el problema (velocidad, precisión, confiabilidad de las respuestas, facilidad de uso, ..., entre otros).

Simultáneamente se establece el problema a resolver, deben diseñarse los objetivos específicos que se espera lograr al resolverlo, así como el alcance que ellos tendrán. Estos aspectos de la definición del problema se relacionan, íntimamente, entre sí.

El costo de ignorar dicha relación, generalmente, se refleja en los pasos siguientes. Ellos resultarán más difíciles de alcanzar. Fácilmente tendremos que volver a replantear todo el problema, debido a cualquier condición importante que pasaba inadvertida.

Esta situación es una molestia en cuanto al tiempo, dinero y esfuerzo humano que se pierde, al retroceder en el diseño del programa. Sin embargo, es evidente que el precio de este descuido supera, enormemente, el costo de hacer un estudio más detenido y minucioso del problema.

2.2. Selección Del Método De Solución.

Cuando el problema está bien enfocado, se le tiene que encontrar una solución. El método empleado para obtenerla puede ser planteado de distintas maneras. Pero, casi todos, podemos agruparlos dentro de tres categorías básicas. Ellas son, en orden creciente de importancia:

⇒ Métodos geométricos.

⇒ Métodos analíticos.

⇒ Métodos iterativos.

A continuación, explicaremos con más detalle esta clasificación.

2.2.1. Métodos Geométricos.

Los métodos geométricos, generalmente, emplean gráficas para resolver los problemas. Ellas pueden ser de tipo sintético o analítico.

Las construcciones de tipo sintético se realizan, comúnmente, dentro del marco de la geometría euclidiana clásica. Es decir, se toman como base: los axiomas fundamentales de la teoría, la observación directa de la figura, y las proposiciones previamente demostradas. Con base a estas premisas, se derivan nuevas propiedades de las figuras. Este enfoque geométrico es el más primitivo de los dos y para efectos de programación de computadoras, no es funcional, en la actualidad.

Por otro lado, las construcciones de tipo analítico se realizan estableciendo un sistema de ejes coordenados en la figura en estudio. A cada punto importante se le asocia un par ordenado de números reales (que puede ser constante o variable). El par ordenado en mención, esta dado en función a las coordenadas del punto con base al sistema referencial. Las figuras, en tanto, serán representadas como ecuaciones; obtenidas a través de métodos de sección y proyección de segmentos. Esta representación define una biyección entre puntos geométricos y coordenadas algebraicas. En consecuencia, se transforma el análisis geométrico en algebraico, empleando las reglas, herramientas y ecuaciones típicas del álgebra.

De los dos enfoques, geométricos éste tiene más vigencia en la programación de computadoras.

Un ejemplo del último enfoque, lo vemos en la solución de sistemas de ecuaciones simultáneas por el método gráfico.

2.2.2. Métodos Analíticos.

Los métodos analíticos se basan en planteamientos derivados de las ramas clásicas de las matemáticas abstractas (álgebra, geometría analítica y análisis matemático, entre otras).

Generalmente, consisten en la aplicación de una fórmula o método preestablecido, con base a los supuestos con que fue construido. Ellos originan, el último enfoque para la solución de problemas (los planteamientos iterativos).

La programación de estos métodos puede presentar grados muy diversos de dificultad (dependiendo de cada fórmula particular).

Como ejemplo de ellos, tenemos la fórmula para la solución de una ecuación cuadrática, en el caso general.

2.2.3. Métodos Iterativos.

Los métodos iterativos son el enfoque más importante para resolver problemas, usando computadoras. También se fundamentan en las ramas clásicas de la matemática abstracta.

Se caracterizan porque emplean procesos iterativos y recurrentes en la solución de los problemas (fundamentados en la teoría de sucesiones y series del análisis matemático).

Los métodos basados en técnicas iterativas y analíticas, forman la esencia de la disciplina matemática conocida como “ANÁLISIS NUMÉRICO”.

El análisis numérico trata de expresar y calcular, en forma tan exacta como sea posible, algunos de los conceptos, métodos y expresiones matemáticas que son estudiadas dentro del “ANÁLISIS MATEMÁTICO”, tales como:

- ⇒ Integrales definidas.
- ⇒ Derivadas evaluadas en un punto.
- ⇒ Funciones trigonométricas de un ángulo dado.
- ⇒ Solución de ecuaciones diferenciales con valores a la frontera, entre otros tópicos.

El planteamiento de estas expresiones se realiza en función a métodos “CONSTRUCTIVOS” que involucran un número finito de operaciones matemáticas. En el análisis numérico, el auxilio de los programas de computadora, generalmente, es imprescindible; por el volumen de operaciones aritméticas y lógicas a realizar.

Independientemente de los tipos de métodos establecidos, en la práctica, la solución de un problema puede involucrar la combinación de uno o más métodos conocidos, y hasta el diseño de uno completamente nuevo.

Cuando no es posible encontrar una vía para resolver al problema, se hace necesario su replanteamiento, para simplificarlo. En consecuencia, se facilita su solución. Por otro lado, se ven casos en los que hay un procedimiento de solución que resulta obvio.

Su programación también puede presentar grados muy diversos de dificultad.

Como ejemplo de estos métodos, tenemos el cálculo de raíces de una función, empleando el procedimiento de bisección de Bolzano.

2.3. Creación Del Algoritmo.

Cuando se ha escogido un método de solución, debemos establecerlo, por escrito, con tanto detalle como sea posible. Se debe establecer como una secuencia de pasos “BIEN DEFINIDA” y tan “COMPLETA” como sea posible, para conservar la claridad de la definición del problema y del método que se aplicará en su solución.

No debe dejarse nada al azar. En ese sentido, si se presenta un problema particular, que el método no puede resolver, él debe ser capaz, al menos, de informar al usuario que no puede obtener una respuesta satisfactoria con ese enfoque.

Estas necesidades particulares se satisfacen a través del empleo de un algoritmo, que define “INEQUÍVOCAMENTE” la lógica usada en la solución del problema. En esencia, ellos no difieren significativamente de los usados dentro del álgebra cotidiana. En consecuencia, debemos construirlos, hasta donde sea posible, empleando el lenguaje natural.

Sin embargo, a nivel de la programación de computadoras, debemos ajustar el lenguaje natural a las tres clases fundamentales de órdenes estructuradas descendentes (la asignación, la decisión y la repetición). Adicionalmente, se deben implementar, en el algoritmo, estructuras adicionales de:

⇒ Entrada y salida de la información.

⇒ Cálculo numérico y manipulación de datos.

⇒ Comparación lógica y relacional.

⇒ Almacenamiento y recuperación de información.

Es importante resaltar en este punto, que cada algoritmo diseñado para resolver un problema, está asociado a una estructura o tipo de datos que le permite administrar la información del problema de manera más o menos eficiente. Dependiendo de esto, un mismo problema se puede resolver de varias maneras, cada una de ellas con cierto grado de eficiencia frente a las demás.

Esto nos lleva a que en un momento dado podemos tener un algoritmo eficiente, con una estructura de datos deficiente, o viceversa. Queda entonces al buen juicio del analista, el decidir si es más importante tener un algoritmo o una estructura de datos eficiente, para lograr la solución al problema originalmente planeado.

En todo caso, la creación de un algoritmo es un proceso repetitivo, que se refina paulatinamente. Se parte de un esquema burdo de la lógica que soluciona el problema, y se llega (luego de mucho esfuerzo), al grado de detalle que caracteriza a los programas de computadora. Cuando logramos ese nivel en el planteamiento del algoritmo, su programación se realiza prácticamente sin esfuerzo (se reduce a la “TRADUCCIÓN” de las órdenes al formato que comprende la máquina).

2.3.1. Características Que Debe Reunir Todo Algoritmo.

Cuando creamos un algoritmo, debemos tratar de reunir, en él, una serie de características importantes. Ellas nos garantizan que el algoritmo producido tiene un alto grado de calidad. Estas son:

⇒ Integridad.

⇒ Claridad.

⇒ Simplicidad.

⇒ Eficacia.

⇒ Modularidad.

⇒ Generalidad.

A continuación, explicaremos con más detalle estas características.

2.3.1.1. Integridad.

Esta característica se refiere a la exactitud y precisión de los cálculos involucrados. No tiene sentido crear un algoritmo que tendrá un comportamiento impredecible, y en consecuencia, que manipule nuestra información de manera aleatoria.

2.3.1.2. Claridad.

Es un aspecto más sutil. Se refiere a la legibilidad de la lógica involucrada. Si un algoritmo es claro, es posible entregarlo a otra persona para que lo modifique, sin grandes dificultades. Adicionalmente, el programador original podrá comprender y modificar su lógica, aunque tenga mucho tiempo de haberla elaborado.

2.3.1.3. Simplicidad.

Esta característica suele recoger a las dos anteriores. Si un algoritmo es fácil de comprender, desde el punto de vista lógico, y manipula correctamente la información, su simplicidad se establece como una consecuencia inmediata.

2.3.1.4. Eficacia.

La eficacia del algoritmo se refiere a su velocidad de ejecución y la administración eficiente de los recursos de la máquina. Cuando nos enfrentamos a problemas complejos, que exigen llevar los recursos disponibles al límite de su

agotamiento, se hace imperativo que los algoritmos sean eficaces, aun a expensas de su claridad. En estas situaciones, el sentido común del programador juega un papel muy importante, pues debe decidir cuanto se sacrificará una característica frente a la otra.

2.3.1.5. Modularidad.

“DIVIDE Y VENCERÁS” dice un viejo adagio popular. Él, tiene vigencia en la creación de algoritmos. Cuando nos enfrentamos a un problema complejo, es importante subdividirlo en tareas más sencillas. En la mayoría de los casos, la modularidad de un algoritmo incrementa su claridad, y eficiencia, simultáneamente. Sin embargo, el sentido común del programador es el mejor catalizador de esta característica.

2.3.1.6. Generalidad.

El esfuerzo de programar se hace inútil si no damos generalidad a los algoritmos, ya que se tendrá que repetir todo el análisis del problema, cada vez que se plantea una situación no contemplada en el estudio. Sin embargo, la generalidad de un algoritmo debe ser limitada, generalmente a las tareas que se pueden prever de manera razonable. En ese sentido, nuevamente, el sentido común juega un papel muy importante en el diseño de algoritmos.

Como ejemplo de un algoritmo representativo (bastante simple), podemos mencionar el aplicado en el caso de las raíces de la ecuación cuadrática de coeficientes y raíces reales.

2.4. Programación Del Algoritmo.

Dependiendo del grado de precisión logrado en la etapa anterior, el paso que mencionaremos en este apartado será, directamente, más sencillo de lograr.

En esencia, cada una de las pautas lógicas establecidas dentro del algoritmo, se debe traducir a una instrucción equivalente en el lenguaje de programación. Así, las características del algoritmo definen, directamente, las características de su programación. Técnicamente, hablamos de “CODIFICAR EL ALGORITMO”.

Posteriormente, el listado del programa es grabado en un medio de almacenamiento de la computadora. Luego, es traducido al lenguaje de la máquina empleando un conjunto de programas, denominados compiladores-enlazadores o intérpretes. Ellos, siguen procedimientos específicos de empleo, dependiendo de cada sistema particular que se use.

Muchos programadores novatos sienten la “TENTACIÓN” de escribir la solución del problema directamente en el lenguaje de programación, sin completar las etapas previas a este paso. Esta acción es una pérdida de tiempo, ya que generalmente, estos intentos no abarcan la totalidad del problema, o dejan algunos detalles al azar. Estas fallas, en su momento, se harán evidentes y habrá que retroceder a esta etapa.

La referencia a lenguajes de programación, compiladores-enlazadores, e intérpretes amerita algunas aclaraciones adicionales.

Como mencionamos anteriormente, la programación del algoritmo consiste, fundamentalmente, en su traducción a un formato equivalente, que comprende la computadora (llamado lenguaje de máquina). Este proceso se

logra a través del empleo de una representación intermedia, denominada lenguaje de programación. Un lenguaje de programación se puede definir, en consecuencia, como un sistema completo de simbolismos, caracteres y reglas, que nos permiten comunicarnos con la computadora.

En teoría, todos los lenguajes de programación deben brindar la facilidad de implementar las estructuras de datos y de control definidas dentro de los algoritmos. Pero, en realidad, existe una marcada diferenciación en la implementación de cada uno de ellos. Estas diferencias los ubican, individualmente, con debilidades, libertades y potencialidades específicas, frente a un problema dado. Generalmente, aquellos que son más fáciles de usar e implementan mayores posibilidades de programación, se denominan “LENGUAJES DE ALTO NIVEL”.

Los lenguajes de programación se implementan a través de los intérpretes y compiladores-enlazadores. Ambos son programas, generalmente diseñados en lenguaje de máquina. Se encargan de “TRADUCIR” las secuencias de órdenes de los programas escritos en un lenguaje de programación de alto nivel, al lenguaje de máquina.

Los compiladores-enlazadores realizan la traducción de todas las órdenes a la vez; como un todo, creando en el proceso un programa que puede ser ejecutado directamente en la máquina sin depender del compilador-enlazador.

Los intérpretes, en cambio, realizan la traducción de las órdenes al lenguaje de máquina, procesándolas y ejecutándolas una a una, dinámicamente. Los intérpretes no crean programas ejecutables (como los creados por los compiladores-enlazadores).

2.5. Depuración Y Documentación Del Programa.

Es de humanos cometer errores, pero la verdad de esta afirmación es más evidente para quienes han tenido experiencias en la programación de computadoras. Prácticamente ningún analista logra que sus programas funcionen correctamente en el primer intento. Debe someterlos a un proceso de corrección y depuración de errores de todo tipo.

Así, los compiladores-enlazadores e intérpretes requieren que el programa cumpla con toda una serie de reglas semánticas y sintácticas preestablecidas dentro del lenguaje, para realizar la traducción al lenguaje de máquina. Una coma mal ubicada, un paréntesis que falte o sobre, una orden mal escrita; todas son situaciones que provocan que un programa no se pueda compilar, o funcione incorrectamente.

Adicionalmente, en esta etapa podemos enfrentarnos a una serie de situaciones no previstas dentro del proceso. El caso más importante lo vemos cuando surgen resultados incorrectos; debido a que los algoritmos, que son exactos en teoría, sufren los problemas conocidos como error por redondeo y error por truncamiento, de las cifras involucradas en los cálculos. Sus efectos pueden ser tan catastróficos al aparecer, que llegan a invalidar completamente la efectividad del algoritmo (es decir, harán que no resuelva el problema originalmente planteado).

Estos errores deben ser eliminados del programa a través de un cuidadoso y exhaustivo análisis de los procesos previos a este paso. Es decir, revisando el método de solución de una manera más detenida y considerando los detalles de implementación en la computadora.

Por otro lado, se deben crear documentos escritos, complementarios al programa diseñado. Esta, información debe reunirse de manera histórica y sistemática, paralelamente a los cambios que sufra el programa en su vida útil. Estos escritos se conocen como “DOCUMENTACIÓN DEL SISTEMA” y deben reflejar, en lo posible, lo que pensaba el programador al diseñar la aplicación. Debe reunir, entre otras cosas, detalles acerca de:

- ⇒ La definición del problema que se intenta resolver.
- ⇒ La descripción del método de solución empleado.
- ⇒ “TODOS” los algoritmos y listados de los módulos que integran al programa principal y sus partes.
- ⇒ La enumeración detallada de “TODAS” las instrucciones que pueden realizar los operadores del programa, así como “TODOS” los pasos para realizarlas correctamente (manual del usuario).
- ⇒ La administración del teclado, por parte del programa.
- ⇒ Las convenciones que deben asumirse en el manejo de la información.
- ⇒ Los controles para verificar la integridad de la información.
- ⇒ Las limitaciones, libertades y potencialidades que tiene el programa.

Estos documentos deben reposar en el código fuente y en copias impresas separadas. Dicha información se orienta a los posibles usuarios del programa y a los programadores que, en el futuro, modifiquen su contenido.

La importancia de estos escritos se debe a que, el programador “GUARDA” la lógica del sistema, en su mente, cuando está recién confeccionado. Sin embargo, con el transcurrir del tiempo, esa información se olvida y, consecuentemente, el código fuente que una vez fue claro, lógico y sencillo; nos parecerá incomprensible, e imposible de modificar.

La documentación que reposa en el código del programa se debe crear tan pronto se completa cada parte del sistema, para lograr que los conceptos originalmente planteados se coloquen íntegramente allí. Está orientada fundamentalmente hacia los programadores.

En tanto, la documentación escrita es de tipo más general y se orienta a los usuarios del programa. Fundamentalmente, trata de auxiliar al usuario en la implementación del sistema, explicando la naturaleza de la información que se suministra al programa, las salidas que produce, y los procedimientos empleados para realizar éstas y otras operaciones.

Resumiendo, esta información es importante para facilitar la depuración, mantenimiento y posterior uso del sistema, por quienes no lo programaron.

2.6. Validación De La Solución.

Si nuestro algoritmo no a sido objeto de un estudio cuidadoso y profundo en las etapas iniciales, al llegar a este punto, nos enfrentaremos a preguntas tales como:

⇒ ¿Las soluciones obtenidas tienen sentido al enfrentarlas a la teoría del problema planteado?

⇒ ¿Los casos triviales son contemplados adecuadamente dentro del enfoque de la solución planteada?

⇒ ¿Se obtienen las soluciones correctas al aplicar el algoritmo a una amplia gama de problemas específicos, con respuesta conocida?

⇒ ¿Las simplificaciones realizadas inicialmente (al definir el problema) afectan al algoritmo, al punto que sus resultados no tienen ninguna utilidad práctica?

Las preguntas planteadas, evidentemente, no las puede responder la computadora. Es la persona que realizó el análisis del problema quien debe enfrentarlas. Dependiendo del grado de eficiencia que se logró en las etapas iniciales del análisis del problema, la respuesta a estas preguntas será una trivialidad, o nos obligará a replantear todo el proyecto desde la definición del problema. Esta etapa se cumple, a veces, simultáneamente a la depuración del código fuente.

2.7. Instalación, Producción Y Mantenimiento Del Programa.

Finalmente, y luego de muchos esfuerzos, el programador logra que su diseño funcione correctamente. Nuestro programa es instalado permanentemente como sistema y entra a la fase de producción. En ella obtenemos los beneficios de la rapidez en los cálculos, al resolver los problemas planteados, con gran velocidad y confiabilidad. En este momento, podemos

pensar que la faena del programador ha concluido. Desgraciadamente, eso es falso.

La implantación del programa no se puede realizar a la ligera. Es necesario que se establezca un período de “PRUEBA” para el sistema. En ese tiempo, se verificará que, en realidad, el programa ejecuta todos los procesos originalmente contemplados en el diseño. Esto se logra a través de la comparación de resultados logrados con la máquina, frente a los obtenidos por métodos manuales. Si es necesario, el programa es nuevamente revisado y corregido. Dichas correcciones deben ser reflejadas, lógicamente, en la documentación escrita que lo acompaña.

Por otro lado, al igual que cualquier creación del hombre, los programas de computadora no son eternos. Deben recibir un mantenimiento adecuado, con el objeto de que se mantengan a tono con las necesidades que van surgiendo, dinámicamente, con el transcurrir del tiempo.

Por ejemplo, pueden surgir nuevas situaciones que originalmente no fueron previstas, y requieren la atención de quienes emplean el programa. En tal caso, el programa debe ser modificado para adaptarlo a esas circunstancias imprevistas inicialmente.

Usualmente, el proceso de mantenimiento se lleva a cabo en tanto el programa sea provechoso. Una vez deja de ser útil, sencillamente se abandona y la información que manejaba (si es necesario) es transferida a otros sistemas para que la aprovechen.

3. Comentarios Finales.

El proceso al que nos hemos referido, se ve como algo bastante difícil y tedioso. Sin embargo, la práctica de la programación de computadoras siguiendo este procedimiento, acarrea grandes beneficios, a largo plazo. Entre ellos, tenemos la creación de programas altamente confiables y eficientes. Para facilitar la comprensión de este proceso, hemos confeccionado la figura que, a continuación, mostramos:

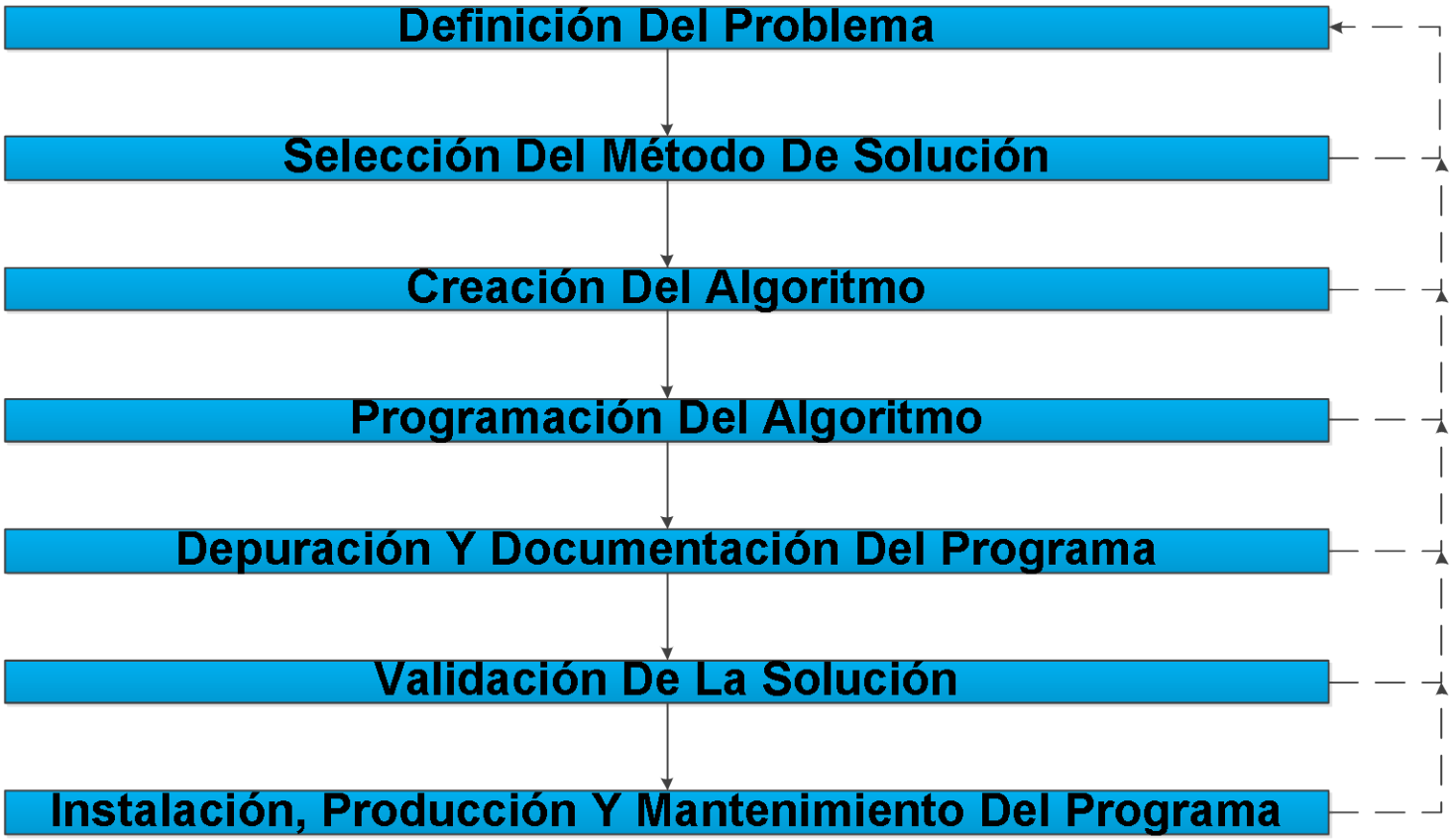

Figura 1: *Los siete pasos necesarios para producir programas de calidad. Las flechas con trazo continuo indican la secuencia a seguir, obligatoriamente. Las fechas punteadas señalan que se puede retroceder en el proceso cuando sea necesario, para mejorar el producto final.*

Este proceso, permite desprender toda una serie de señalamientos bien claros, a manera de conclusiones.

⇒ La computadora no es capaz de resolver problemas por sí sola. Simplemente puede realizar secuencias de pasos preestablecidas, a través de los programas que se le han instalado.

⇒ La computadora no releva al ser humano de la responsabilidad de estructurar su trabajo. Al contrario, lo obliga a realizar una planificación más detallada y cuidadosa del mismo.

⇒ La computadora no releva al hombre de interpretar los resultados obtenidos. Ella le exige un conocimiento más amplio de los problemas que enfrenta, al tener que conjugar una serie de situaciones no previstas cuando trabaja manualmente, y que pasan a formar parte adicional del problema que enfrenta.

⇒ La programación de computadoras puede resultar más simple para quienes tienen una formación de índole matemática, que a los miembros de otras disciplinas (salvo, lógicamente, los analistas y programadores de sistemas).

La última afirmación requiere algunas aclaraciones adicionales. Generalmente las personas con formación matemática tienen la lógica altamente desarrollada y educada. Se les puede facilitar, enormemente, la creación de métodos, algoritmos y programas, con respecto a los integrantes de otras disciplinas del saber. Todo es cuestión de que, previamente, organicen sus conocimientos lógicos, de modo que desarrollen las soluciones de los problemas en función a las disposiciones de la programación estructurada descendente.

Sin embargo, esto no significa que alegremente podrán programar a las computadoras. Ellos deben adquirir algunos conocimientos básicos de programación de computadoras y análisis de sistemas de información.

Adicionalmente, esto no impide a las personas, que en general, son capaces de organizar sus pensamientos de manera estructurada, que realicen la programación de computadoras para resolver sus problemas personales. Todo depende del grado de organización de sus ideas y planteamientos, además del dominio de algunos conceptos básicos de programación de computadoras.

BIBLIOGRAFÍA

1. ALLEN SMITH, W. Análisis Numérico. Traducido por Francisco Javier Sánchez Bernabe. Primera edición. México, D.F., México: Prentice-Hall, 1988. 608 páginas.

2. CHAPRA, Steven, y CANALE, Raymond P. Métodos numéricos para ingenieros con aplicaciones en computadoras personales. Traducido por Carlos Zapata S. Primera edición. México D.F., México: McGraw-Hill, 1990. 641 páginas.

3. GOTTFRIED, Byron S. Programación en Pascal. Traducido por Alfredo Bautista Paloma. Primera edición. México D.F., México: McGraw-Hill, 1988. 398 páginas.

4. JAMES, Merlin L, SMITH, Gerald M., y WOLFORD, James C. Métodos numéricos aplicados a la computación digital con FORTRAN. Traducido por José A. Nieto Ramírez. Primera edición. México, D.F., México: Representaciones y Servicios de Ingeniería, 1973. 575 páginas.

5. JOYANES AGUILAR, Luis. Turbo Basic. Manual de Programación. Primera edición. Madrid, España: McGraw-Hill, 1989. 525 páginas.

6. HENRICE, Peter. Elementos de análisis numérico. Traducido por Federico Velasco Coba. Primera edición. México, D.F., México: Trillas, 1972. 363 páginas.

7. KORFHAGE, Robert R. Lógica y algoritmos: Con aplicaciones a las ciencias de la computación e información. Traducido por Federico Velasco C. Primera edición. México, D.F., México: Limusa, 1970. 222 páginas.

8. LIPSCHUTZ, Seymour. Estructura de datos. Traducido por Manuel Ortega Ortíz de Apodaca y Luis Hernández Yañez. Primera edición. México, D.F., México: McGraw-Hill, 1987. 390 páginas.

9. LUTHE, Rodolfo, OLIVERA, Antonio, y SCHUTZ, Fernando. Métodos numéricos. Primera edición. México, D.F., México: Limusa, 1986. 443 páginas.

10. MCCRAKEN, Daniel D., y DORN, William S. Métodos numéricos y programación FORTRAN. Traducido por José A. Nieto Ramírez. Primera edición. México, D.F., México: Limusa, 1986. 476 páginas.

11. NELL, Dale, y LILLY, Susan C. Pascal y estructura de datos. Traducido por José María Troya Linero. Primera edición. México, D.F., México: McGraw-Hill, 1988. 491 páginas.

12. PALMER, Scott D. Domine Turbo Pascal 6. Traducido por Eduardo de la Calle Suárez. Primera edición. México, D.F., México: Ventura, 1992. 597 páginas.

13. PHILIPPAKIS, A. S. y KAZMIER, Leonard J. Diseño de programas con aplicaciones en COBOL. Traducido por José María Troya Linero. Primera Edición. Madrid, España: McGraw-Hill, 1984. 239 páginas.

14. RALSTON, Anthony. Introducción al análisis numérico. Traducido por Carlos E. Cervántes de Gortari. Primera edición. México, D.F., México: Limusa, 1970. 629 páginas.

15. REINHARDT, Fritz y SOEDER, Heinrich. Atlas de matemáticas 1: Fundamentos, álgebra y geometría. Traducido por Juan Luis Vázquez Suárez y Mario Rodríguez Artalejo. Primera edición. Madrid, España: Alianza Editorial, 1984. 265 páginas.

16. SANDERS, Donald H. Informática: Presente y futuro. Traducido por Roberto Luis Escalona. Tercera Edición. México D.F., México: McGraw-Hill, 1991. 887 páginas.

17. SANTALÓ SORS, Marcelo, y CARBONELL CHAURE, Vicente. Geometría Analítica. Primera Edición. México D.F., México: Librería de Manuel Porrúa, 1959. 255 páginas.

18. SCHEID, Francis. Análisis numérico. Traducido por Hernando Alonso Castillo. Primera Edición. México D.F., México: McGraw-Hill, 1972. 422 páginas.

19. SCHEID, Francis. Introducción a la ciencia de las computadoras. Traducido por Alberto Jaime Sisa. Segunda Edición. México D.F., México: McGraw-Hill, 1985. 402 páginas.

20. SCHEID, Francis, y Di Constanzo, Rosa Elena. Métodos numéricos. Traducido por Gabriel Nagore Cázares. Segunda Edición. México D.F., México: McGraw-Hill, 1991. 709 páginas.

21. SEEN, James A. Análisis y diseño de sistemas de información. Traducido por Edmundo Gerardo Urbina Medal y Óscar Alfredo Palmas Velasco. Segunda Edición. México D.F., México: McGraw-Hill, 1992. 942 páginas.

22. SQUIRE, Enid. Introducción al diseño de sistemas. Traducido por Jaime Luis Valls Cabrear. Primera Edición. México D.F., México: Fondo Educativo Interamericano, 1984. 345 páginas.