|  |
| --- |
| Laboratorio de Computación  Salas A y B |
|  |

|  |  |
| --- | --- |
| Profesor: | García Morales Karina |
| Asignatura: | Fundamentos de programación |
| Grupo: | 1121 |
| No de Práctica(s): | 03 |
| Integrante(s): | Valle Olivas Guillermo |
|  |  |
|  |  |
|  |  |
| No. de Equipo de cómputo empleado: |  |
| Semestre: | 2019-1 |
| Fecha de entrega: | 02/09/2018 |
| Observaciones: |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Solución de problemas y Algoritmos**

**Objetivo:**

Elaborar algoritmos correctos y eficientes en la solución de problemas siguiendo las etapas de Análisis y Diseño pertenecientes al Ciclo de vida del software.

**Actividades:**

x A partir del enunciado de un problema, identificar el conjunto de entrada y el conjunto de salida.

x Elaborar un algoritmo que resuelva un problema determinado (dado por el profesor), identificando los módulos de entrada, de procesamiento y de salida.

**Introducción**

Un problema informático se puede definir como el conjunto de instancias al cual corresponde un conjunto de soluciones, junto con una relación que asocia para cada instancia del problema un subconjunto de soluciones (posiblemente vacío).

Para poder solucionar un problema nos apoyamos en la Ingeniería de Software que de acuerdo a la IEEE se define como “La aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software". Por lo que el uso y establecimiento de principios de ingeniería sólidos, son básicos para obtener un software que sea económicamente fiable y funcione eficientemente.

La Ingeniería de Software provee métodos que indican cómo generar software. Estos métodos abarcan una amplia gama de tareas:

x Planeación y estimación del proyecto.

x Análisis de requerimientos del sistema y software.

x Diseño de la estructura de datos, la arquitectura del programa y el procedimiento algorítmico.

x Codificación

x Pruebas y mantenimiento (validación y verificación).

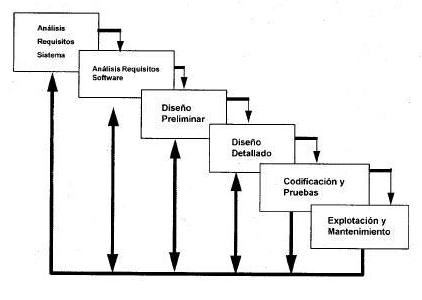
**Ciclo de vida del software**

La ISO (International Organization for Standarization) en su norma 12207 define al ciclo de vida de un software como:

Un marco de referencia que contiene las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando desde la definición hasta la finalización de su uso.



En los años 70 se impuso un enfoque de desarrollo del software, introducido por Royce, a través de un ciclo de vida en "cascada". Este método modela el ciclo convencional de la ingeniería del software, aplicando un enfoque sistemático y secuencial de desarrollo que comienza con la ingeniería del sistema y progresa a través del análisis, diseño, codificación, pruebas y mantenimiento.



Normalmente, el “ciclo de vida del software” se suele dividir en tres fases: una de planificación, otra de desarrollo y una tercera de mantenimiento, que engloban a las seis etapas (Ingeniería del sistema, análisis de los requisitos, diseño, codificación, pruebas y mantenimiento) tradicionales del ciclo de vida.

**Teoría de la computabilidad**

El objetivo central de esta disciplina es la caracterización de una propiedad específica de ciertos procesos: la de ser o no computables.

Decimos que una determinada propiedad es computable si es posible deducir todos sus valores siguiendo la aplicación de un procedimiento determinado en un número también determinado de pasos. La importancia de caracterizar una propiedad o un procedimiento como computable reside en que, si efectivamente lo es, la propiedad en cuestión podrá ser puesta en funcionamiento en una máquina diseñada para computar.

**Solución de problemas**

Dentro del ciclo de vida del software, en el análisis se busca comprender la necesidad, es decir, entender el problema.

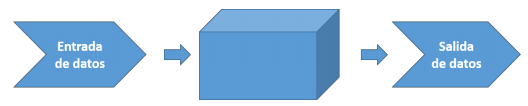
El análisis es el proceso para averiguar qué es lo que requiere el usuario del sistema de software (análisis de requisitos). Esta etapa permite definir las necesidades de forma clara y concisa (especificación de requisitos).

Por lo tanto, la etapa del análisis consiste en conocer qué es lo que está solicitando el usuario. Para ello es importante identificar dos grandes conjuntos dentro del sistema: el conjunto de entrada y el conjunto de salida.

El **conjunto de entrada** está compuesto por todos aquellos datos que pueden alimentar al sistema.

El **conjunto de salida** está compuesto por todos los datos que el sistema regresará como resultado del proceso. Estos datos se obtienen a partir de los datos de entrada.

La unión del conjunto de entrada y el conjunto de salida forman lo que se conoce como el dominio del problema, es decir, los valores que el problema puede manejar.



La etapa de análisis es crucial para la creación de un software de calidad, ya que si no se entiende qué es lo que se desea realizar, no se puede generar una solución. Sin embargo, es común caer en ambigüedades debido al mal entendimiento de los requerimientos iniciales.

Ejemplo 1

PROBLEMA: Determinar si un número dado es positivo o negativo.

RESTRICCIONES: El número no puede ser cero.

DATOS DE ENTRADA: El conjunto de datos de entrada E está compuesto por el conjunto de los números reales, excepto el cero.

E ⊂ R1, donde num ∈ E de (−∞, ∞) − {0}

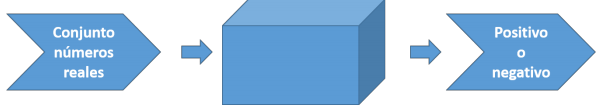
NOTA: R1 representa al conjunto de números reales de una dimensión.

DATOS DE SALIDA: El conjunto de salida S está compuesto por dos valores mutuamente excluyentes.

Un posible conjunto de salida son los valores enteros 0 o 1, donde 0 indica que el valor es positivo y 1 indica el valor es negativo.

res = 0, si num (0, ∞), res = 1, si num (−∞, 0)

Otro posible conjunto de datos de salida son los valores booleanos o lógicos Verdadero o Falso, donde Verdadero indica que el valor es positivo y Falso indica que el valor es negativo; o viceversa, Verdadero indica que el valor es negativo y Falso indica que el valor es positivo.

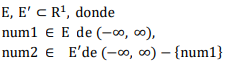


Ejemplo 2

PROBLEMA: Obtener el mayor de dos números diferentes dados.

RESTRICCIONES: Los números de entrada deben ser diferentes.

DATOS DE ENTRADA: El conjunto de entrada E está dividido en dos subconjuntos E y E'. El primer número (num1) puede adquirir cualquier valor del conjunto de los números reales (E = (-∞, ∞)), sin embargo, el conjunto de entrada del segundo número (num2) es un subconjunto de E, es decir, E' está compuesto por el conjunto de los números reales excepto num1 (E' = (-∞, ∞) ≠ num1).



DATOS DE SALIDA: El conjunto de datos de salida S que puede tomar el resultado r está compuesto por el conjunto de los números reales.

S ⊂ R1, donde r ∈ S de (−∞, ∞)



Ejemplo 3

PROBLEMA: Obtener el factorial de un número dado. El factorial de un número está dado por el producto de ese número por cada uno de los números anteriores hasta llegar a 1. El factorial de 0 (0!) es 1:

n! = n \* (n-1)!

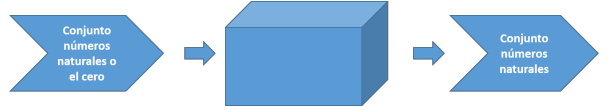
RESTRICCIONES: El número de entrada debe ser entero positivo o cero. No puede ser negativo.

DATOS DE ENTRADA: El conjunto de entrada E está dado por el conjunto de los números naturales o por el cero.

E ⊂ N1, donde num ∈ E de [1, ∞) ⋃ {0}

DATOS DE SALIDA: El conjunto de salida S está conformado por el conjunto de los números naturales.

S ⊂ N1; donde res ∈ S de [1, ∞)



**Algoritmos**

Una vez realizado el análisis, es decir, ya que se entendió qué es lo que está solicitando el usuario y ya identificado el conjunto de entrada y el conjunto de salida, se puede proceder al diseño de la solución, esto es, a la generación del algoritmo.

Dentro del ciclo de vida del software, la creación de un algoritmo se encuentra en la etapa de diseño.

Durante el diseño se busca proponer una o varias alternativas viables para dar solución al problema y con base en esto tomar la mejor decisión para iniciar la construcción.

Un problema matemático es computable si éste puede ser resuelto, en principio, por un dispositivo computacional.

La teoría de la computabilidad es la parte de la computación que estudia los problemas de decisión que pueden ser resueltos con un algoritmo.

Un algoritmo se define como un conjunto de reglas, expresadas en un lenguaje específico, para realizar alguna tarea en general, es decir, un conjunto de pasos, procedimientos o acciones que permiten alcanzar un resultado o resolver un problema. Estas reglas o pasos pueden ser aplicados un número ilimitado de veces sobre una situación particular.

Un algoritmo es la parte más importante y durable de las ciencias de la computación debido a que éste puede ser creado de manera independiente tanto del lenguaje como de las características físicas del equipo que lo va a ejecutar.

Las principales características con las que debe cumplir un algoritmo son:

x Preciso: Debe indicar el orden de realización de paso y no puede tener ambigüedad

x Definido: Si se sigue dos veces o más se obtiene el mismo resultado.

x Finito: Tiene fin, es decir tiene un número determinado de pasos.

x Correcto: Cumplir con el objetivo.

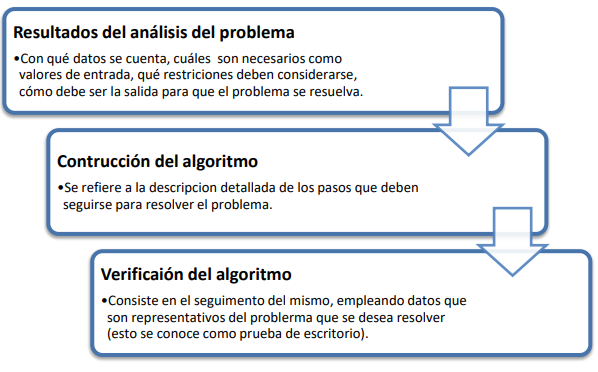
x Debe tener al menos una salida y esta debe de ser perceptible

x Debe ser sencillo y legible x Eficiente: Realizarlo en el menor tiempo posible

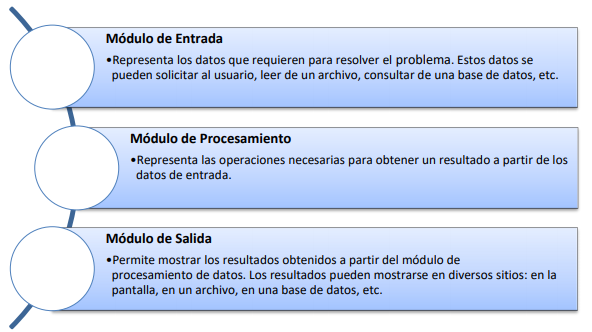
x Eficaz: Que produzca el efecto esperado

Por tanto, un buen algoritmo debe ser correcto (cumplir con el objetivo) y eficiente (realizarlo en el menor tiempo posible), además de ser entendible para cualquier persona.

Las actividades a realizar en la elaboración de un algoritmo para obtener una solución a un problema de forma correcta y eficiente se muestran en la figura



Un algoritmo consta de 3 módulos básicos:



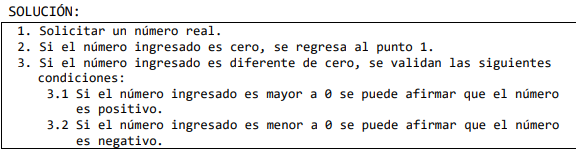
**Ejemplo 1**

PROBLEMA: Determinar si un número dado es positivo o negativo.

RESTRICCIONES: El número no puede ser cero.

DATOS DE ENTRADA: Número real. DATOS DE SALIDA: La validación de si el número es positivo

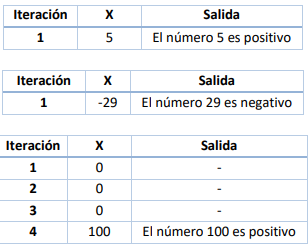
DOMINIO: Todos los número reales.



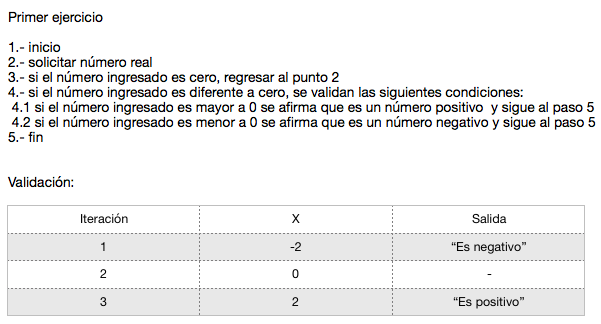
**Prueba de escritorio**

El diseño de la solución de un problema implica la creación del algoritmo y la validación del mismo. La validación se suele realizar mediante una prueba de escritorio

Una prueba de escritorio es una matriz formada por los valores que van adquiriendo cada una de las variables del programa en cada iteración. Una iteración es el número de veces que se ejecuta un código y permite ver los valores que van adquiriendo las variables en cada repetición. Para el ejemplo en cuestión la prueba de escritorio quedaría de la siguiente manera (considerando a X como el número solicitado):



**Algoritmo corregido durante la práctica con iteraciones**



**Ejemplo 2**

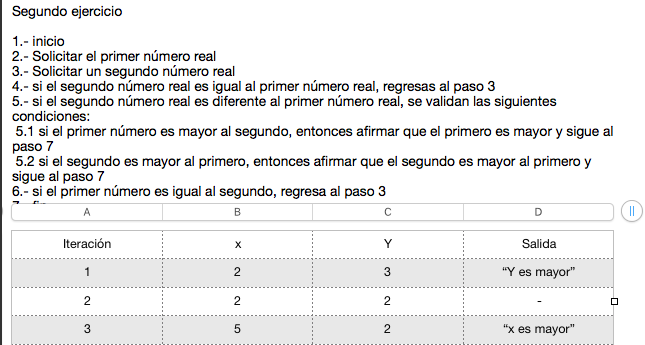
PROBLEMA: Obtener el mayor de dos números dados.

RESTRICCIONES: Los números de entrada deben ser diferentes.

DATOS DE ENTRADA: Número real.

DATOS DE SALIDA: La impresión del número más grande.

DOMINIO: Todos los número reales.



**Ejercicio 1**

PROBLEMA: Seguir el algoritmo para obtener una figura

ENTRADA: Hoja tamaño carta en limpio, regla y lápiz.

SALIDA: Figura correcta.

Algoritmo

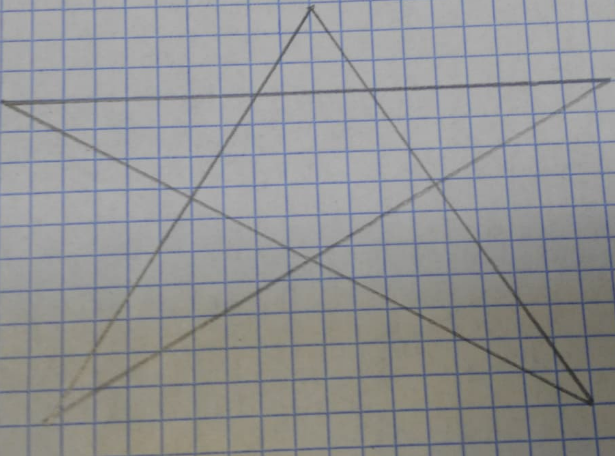
1. Dibuja una V invertida. Empieza desde el lado izquierdo, sube, y baja hacia el lado derecho, no levantes el lápiz.

2. Ahora dibuja una línea en ángulo ascendente hacia la izquierda. Debe cruzar la primera línea más o menos a 1/3 de la altura. Todavía no levantes el lápiz del papel.

3. Ahora, dibuja una línea horizontal hacia la derecha. Debe cruzar la V invertida más o menos a 2/3 de la altura total. Sigue sin levantar el lápiz.

4. Dibuja una línea en un ángulo descendente hasta el punto de inicio. Las líneas deben unirse.

5. Ahora ya puedes levantar el lápiz del papel. Has terminado la estrella de 5 puntas.



**Ejercicio 2**

PROBLEMA: Seguir el algoritmo para obtener una figura

ENTRADA: Hoja tamaño carta en limpio, regla y lápiz.

SALIDA: Figura correcta.

Algoritmo

1. Empieza dibujando un círculo con un compás. Coloca un lápiz en el compás. Coloca la punta del compás en el centro de una hoja de papel.

2. Ahora gira el compás, mientras mantienes la punta apoyada en el papel. El lápiz dibujará un círculo perfecto alrededor de la punta del compás.

3. Marca un punto en la parte superior del círculo con el lápiz. Ahora, coloca la punta del compás en la marca. No cambies el radio del compás con que hiciste el círculo.

4. Gira el compás para hacer una marca en el propio círculo hacia la izquierda. Haz una marca también en el lado derecho.

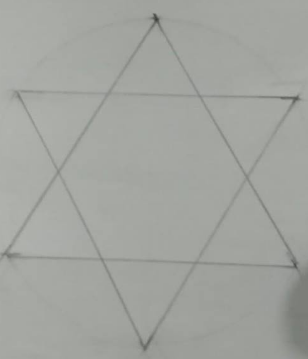
5. Ahora, coloca la punta del compás en uno de los puntos. Recuerda no cambiar el radio del compás. Haz otra marca en el círculo.

6. Continúa moviendo la punta del compás a las otras marcas, y continúa hasta que tengas 6 marcas a la misma distancia unas de otras. Ahora, ya puedes dejar tu compás a un lado.

7. Usa una regla para crear un triángulo que empiece en la marca superior del círculo. Coloca el lápiz en la marca superior. Ahora dibuja una línea hasta la segunda marca por la izquierda. Dibuja otra línea, ahora hacia la derecha, saltándote la marca de la parte más baja. Complementa el triángulo con una línea hacia la marca superior. Así completarás el triángulo.

8. Crea un segundo triángulo empezando en la marca en la base del círculo. Coloca el lápiz en la marca inferior. Ahora conéctala con la segunda marca hacia la izquierda. Dibuja una línea recta hacia la derecha, saltándote el punto superior. Completa el segundo triángulo dibujando una línea hasta la marca en la parte inferior.

9. Borra el círculo. Has terminado de dibujar tu estrella de 6 puntos.



**Ejercicios de tarea**

**1.- Calcular el volumen de un cilindro a partir del radio de la base y la altura. (Hacer uso de la fórmula V = π R2 h).**

**Análisis:**

Entrada: Radio, altura, π= 3.1416

Salida: Volumen del cilindro

Restricciones: r<=0, h<=0

Metodología: V = π R2 h

**Construcción, diseño:**

1.- Inicio

2.- Escribir radio y altura

3.-Leer radio y altura

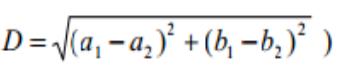
4.- si R y h>0, imprimir el volumen resultante

5.- si R y h<=0, regresar al punto 2 o seguir al punto 6

6.- Fin

|  |  |  |  |
| --- | --- | --- | --- |
| **Iteración** | **Radio** | **Altura** | **Salida** |
| 1 | 2 | 5 | “V= x” |
| 2 | 0 | 0 | - |

**2.- Calcular la distancia entre dos puntos. (Sea P1 (a1, b1) y P2 (a2, b2), hacer uso de**



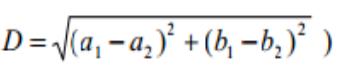
**Análisis:**

Entrada: coordenadas de p1 y p2

Salida: Distancia entre puntos

Restricciones: no es posible que p1 = p2

Metodología:



**Construcción, diseño:**

1.-Inicio

2.- escribir coordenadas de p1 y p2

3.- Leer coordenadas de p1 y p2

4.- si p1 ≠ p2, entonces imprimir distancia resultante y seguir al punto 6

5.- Si p1 = p2, entonces regresar al punto 2 o seguir al punto 6

6.-Fin

|  |  |  |  |
| --- | --- | --- | --- |
| Iteración | Punto1 | Punto2 | Salida |
| 1 | (-2, 2) | (3,5) | “D= x” |
| 2 | (-2, 2) | (-2, 2) | - |

**3.- Leer 2 números y verificar si son divisibles, o el resultado no existe, o es infinito. (Considere que los números deben ser enteros)**

**Analisis:**

Entrada: Dos números enteros, siendo A y B

Salida: C

Restricciones: Solo números enteros

Metodología:



**Construcción, diseño:**

1.- Inicio

2.- Escribir números A y B

3.- Leer números A y B

4.- Si A o B no son enteros, entonces regresar al punto 2 o seguir al paso 8

5.- Si A y B son enteros divisibles, imprimir el resultado y seguir al paso 8

6.- Si A y B son enteros no divisibles, imprimir “no son divisibles” y seguir al paso 8

7.- Si B = 0, imprimir “indeterminado” y seguir al paso 8

8.- Fin

|  |  |  |  |
| --- | --- | --- | --- |
| Iteracion | Número A | Número B | Salida |
| 1 | 10 | 2 | C |
| 2 | 15 | 6 | “No son divisibles” |
| 3 | 7 | 0 | “Indeterminado” |

**4.- Leer un número y verificar si un número es par o impar**

**Analisis:**

Entrada: Un número entero

Salida: Par o Impar

Restricciones: Solo números enteros

Metodología: n/2

**Construcción, diseño:**

1.- Inicio

2.- Escribir un número

3.- Leer el número

4.- Si el número no es entero, regresar al paso 2 o seguir al paso 7

5.- Si el residuo de n/2 = 0, imprimir par y seguir al paso 7

6.- Si el residuo de n/2 ≠0, imprimir “impar**”** y seguir al paso 7

7.- Fin

|  |  |  |
| --- | --- | --- |
| Iteración | Número | Salida |
| 1 | 10 | “par” |
| 2 | 7 | “impar” |

**5.- Leer del número 1 al 50 e indicar cuales números son múltiplos de 3**

**Analisis:**

Entrada: Dos números

Salida: Múltiplos de tres

Restricciones: Solo números enteros

Metodología: n/3

**Construcción, diseño:**

1.- Inicio

2.- Escribir un número de inicio y uno de final (A y B)

3.- Leer los números

4.- Si un número no es entero, regresar al paso 2 o seguir al paso 7

5.- Inicio ciclo PARA desde A hasta B

6.- Si el residuo de n/3≠0, entonces no es múltiplo de 3, ignorar y seguir al paso 6

7.- Si el residuo de n/3=0, entonces es múltiplo de 3, imprimir y seguir al paso 8

8.- Fin PARA, y sigue al paso 9

9.- Fin

|  |  |  |  |
| --- | --- | --- | --- |
| Iteración | Número A | Número B | Salida |
| 1 | 1 | 50 | Múltiplos de 3 |

**Conclusiones**

Hasta el momento la creación de los algoritmos no me está pareciendo tan complicada, con lo que si presento algo de problemas es con las iteraciones, al momento de formar la tabla llego a confundirme bastante, no sé si lo estoy haciendo bien o mal, y en eso pierdo algo de tiempo que es vital en esto de los algoritmos.

**Referencias**

x Raghu Singh (1995). International Standard ISO/IEC 12207 Software Life Cycle Processes. Agosto 23 de 1996, de ISO/IEC. Consulta: Junio de 2015. Disponible en: <http://www.abelia.com/docs/12207cpt.pdf>

x Carlos Guadalupe (2013). Aseguramiento de la calidad del software (SQA). [Figura 1].Consulta: Junio de 2015. Disponible en: https://www.mindmeister.com/es/273953719/aseguramiento-de-la-calidad delsoftware-sqa

x Andrea S. (2014). Ingeniería de Software. [Figura 2]. Consulta: Junio de 2015. Disponible en: <http://ing-software-verano2014.blogspot.mx>

x Michael Littman. (2012). Intro to Algorithms: Social Network Analysis. Consulta: Junio de 2015, de Udacity. Disponible en: <https://www.udacity.com/course/viewer#!/ccs215/l-48747095/m-48691609>

# x  Ignacio Bosque Muñoz. (2009). Fundamentos de sintaxis formal <https://books.google.com.mx/books?id=Uy4n9MzmDJQC&pg=PA74&dq=que+es+la+teor%C3%ADa+de+la+computabilidad&hl=es&sa=X&ved=0ahUKEwjL6dSw8ZzdAhUGXK0KHZnDCHUQ6AEIPTAE#v=onepage&q=que%20es%20la%20teor%C3%ADa%20de%20la%20computabilidad&f=false>

# x  Fernando Alonso Amo . (2005). Introducción A la ingeniería del software <https://books.google.com.mx/books?id=rXU-WS4UatYC&pg=PA77&dq=ciclo+de+vida+del+software&hl=es&sa=X&ved=0ahUKEwid-uWy5ZzdAhUKbK0KHZ_YCO8Q6AEILDAB#v=onepage&q=ciclo%20de%20vida%20del%20software&f=false>