Ej. 1. Explique la Programación Estructurada.

La programación estructurada se inició en los 70, fundamentalmente contra el uso indiscriminado de constructores de control como los “gotos”.

En la programación estructurada un programa tiene una estructura estática la cual es el orden de las sentencias en el código (el cual es un orden lineal) y una estructura dinámica que es el orden el cual las sentencias se ejecutan.

Tiene como objetivo simplificar la estructura de los programas de manera que sea fácil razonar sobre ellos y por lo tanto escribir programas cuya estructura dinámica es la misma que la estática. Osea que las sentencias se ejecutan en el mismo orden que las presenta el código.

Entonces, la programación estructurada simplifica el flujo de control, facilitando en consecuencia tanto la comprensión de los programas así como el razonamiento (formal o informal) sobre estos.

Ej. 2. Describa el proceso de codificación: Desarrollo dirigido por test.

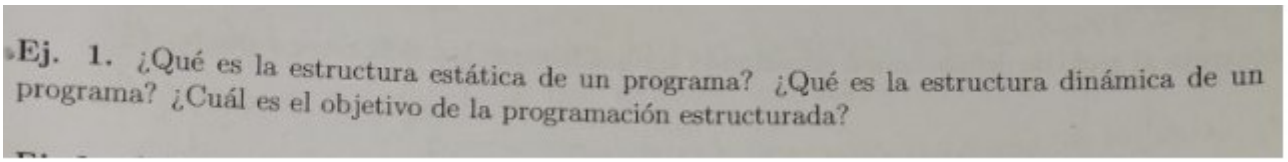
En este proceso de codificación se cambia el orden de las actividades en la codificación, el programador primero escribe los scripts para los tests y luego el código para que estos pasen los casos de tests en el script. Se realiza incrementalmente y se escribe el código suficiente para pasar el test.

No es lo mismo en el modelo de codificación incremental donde los casos de test podrían testear sólo parte de la funcionalidad.

La responsabilidad de asegurar cobertura de toda la funcionalidad radica en el diseño de los casos de test y no en la codificación.

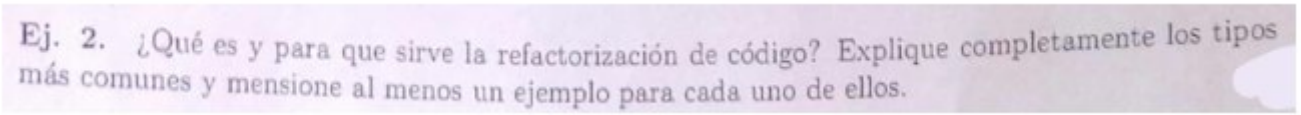
- Se enfoca en cómo será usado el código a desarrollar dado que los tests se escriben primero.
- Ayuda a validar la interfaz del usuario específica en diseño.
- La completitud del código depende de cuan exhaustivo sean los casos de test

El código necesitará refactorización para mejorar el código posiblemente confuso.



Ej. 1. ¿Qué es la estructura estática de un programa? ¿Qué es la estructura dinámica de un programa? ¿Cuál es el objetivo de la programación estructurada?

Con saber la primera ya deberías saber responder esta :D



Ej. 2. ¿Qué es y para que sirve la refactorización de código? Explique completamente los tipos más comunes y mensione al menos un ejemplo para cada uno de ellos.

La refactorización es una técnica utilizada para mejorar el diseño del código existente. Se trata de una tarea que permite realizar cambios en un programa con el propósito de simplificarlo y facilitar su comprensión, todo mientras se mantiene inalterado su comportamiento observacional. Estos cambios se llevan a cabo durante la fase de codificación, pero de manera independiente a la codificación normal.

Tiene varios propósitos que intenta lograr, los cuales son:

- reducir acoplamiento
- incrementar cohesión
- mejorar la respuesta al principio abierto-cerrado.

Y como objetivo
disminuir el costo de hacer cambios en el código, lo cual permite que el diseño de código mejore continuamente en lugar de degradarse con el tiempo

Refactorizaciones mas comunes:

Extracción de métodos :

- Se realiza si el método es demasiado largo.

Objetivo: separar en métodos cortos cuya signatura indique lo que el método hace.
• Partes de código se extraen como nuevos métodos.
• Las variables referenciadas en esta parte se transforman en parámetros.
• Variables declaradas en esta parte pero utilizadas en otras partes deben de unirse en el método original.
• También se realiza si un método retorna un valor y también cambia el estado del objeto. (Dividir en dos métodos).

- Agregar/eliminar parámetros:

• Para simplificar las interfaces donde sea posible.
• Agregar sólo si los parámetros existentes no proveen la información que se necesita.
• Eliminar si los parámetros se agregaron originalmente “por las dudas” pero no se utilizan.

Mejoras de clases:
Desplazamiento de métodos:
• Mover un método de una clase a otra.
• Se realiza cuando el método actúa demasiado con los objetos de la otra clase.
• Inicialmente puede ser conveniente dejar un método en la clase inicial que delegue al nuevo (debería tender a desaparecer).

• **Desplazamiento de atributos:**
• Si un atributo se usa más en otra clase, moverlo a esta clase.
• Mejora cohesión y acoplamiento.

• **Extracción de clases:**
• Si una clase agrupa múltiples conceptos, separa cada concepto en una clase distinta.
• Mejora la cohesión.
• Reemplazar valores de datos por objetos:
Algunas veces, una colección de atributos se transforma en una entidad lógica.
• Separarlos como una clase y de unir objetos para accederlos.

Mejoras de jerarquías
Reemplazar condicionales con polimorfismos :
• Si el comportamiento depende de algún indicador de tipo, no se está explotando El poder de la OO.
• Reemplazar tal análisis de casos a través de una jerarquía de clases apropiada.
• Subir métodos / atributos:
Los elementos comunes deben pertenecer a la superclase.
• Si la funcionalidad o atributo está duplicado en las subclases, pueden subirse a la superclase.

¿Cuáles son las refactorizaciones más comunes?

Mencione al menos un ejemplo para cada una de ellas.

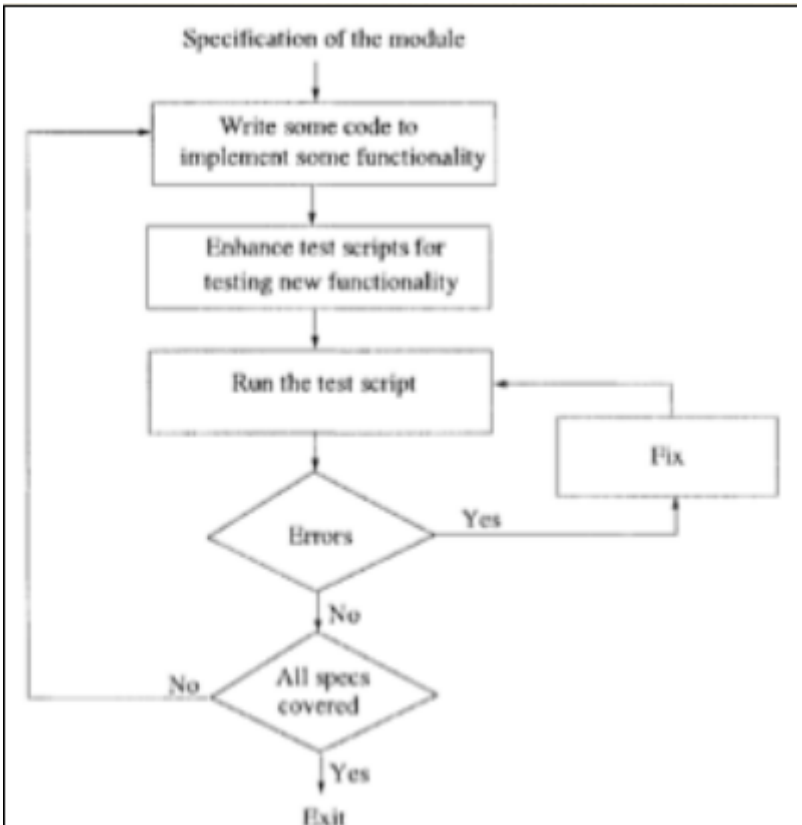
Con la anterior esta ya te deberia salir locura

Ej. 2. Describa los procesos de codificación: incremental, desarrollo dirigido por tests y programación de a pares.

Al principio del archivo, en la segunda resolución está el desarrollo dirigido por tests.

Desarrollo incremental:

- Consta de un proceso básico el cual es: Escribir codigo del modulo
- Realizar test de unidad
- Si hay un error, arreglar bugs y repetir los test



Programación de a pares:
El código se escribe por dos programadores en lugar de uno solo:
Conjuntamente, ambos programadores diseñan los algoritmos, estructuras de datos, estrategias, etcétera.
Una persona tipea el código, la otra revisa activamente el código que se tipea.
Se señalan los errores y conjuntamente formulan soluciones.
Los roles se alternan periódicamente

Explique exhaustivamente el proceso de desarrollo: Iterativo.

Aborda el problema de “todo o nada” del modelo de cascada.
• Combina beneficios del prototipado y del modelo de cascada.
• Desarrolla y entrega el SW incrementalmente.
• Cada incremento es completo en sí mismo.
• Provee un marco para facilitar el testing (el testing de cada incremento es más fácil que el testing del sistema completo).
• Puede verse como una “secuencia de cascadas”.
• El feedback de una iteración puede usarse en iteraciones futuras.

Primer paso:
• implementación simple para un subconjunto del problema completo (sólo aspectos claves y fáciles de entender).
• crear lista de control del proyecto (LCP) que contiene (en orden) las tareas que se deben realizar para lograr la implementación final.

• Cada paso consiste en eliminar la siguiente tarea de la lista haciendo diseño, implementación y análisis del sistema parcial, y actualizar la LCP.
• El proceso se repite hasta vaciar la lista.
• LCP: guía los pasos de iteración y lleva las tareas a realizar.
• Cada entrada en LCP es una tarea a realizarse en un paso de iteración y debe ser lo suficientemente simple como para comprenderla completamente.

Aplicación:
Muy efectivo en desarrollo de productos:
• los desarrolladores mismos proveen la especificación,
• los usuarios proveen el feedback en cada release,
• basado en esto y experiencia previa => nueva versión.
Desarrollo “a gusto del comprador” (customized):
• Las empresas requieren respuestas rápidas.
• No se puede arriesgar el “todo o nada”.

Beneficios:
• pagos y entregas incrementales;
• feedback para mejorar lo desarrollado.
Inconvenientes:

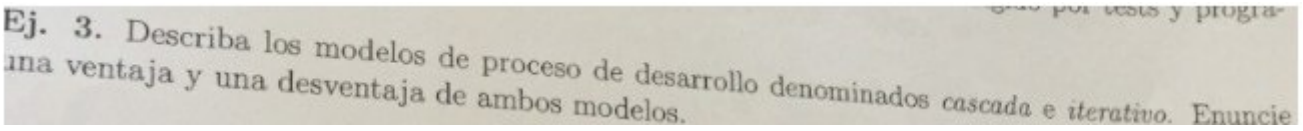
• la arquitectura y el diseño pueden no ser óptimos;
• la revisión del trabajo hecho puede incrementarse;
• el costo total puede ser mayor.
Aplicación:
• cuando el tiempo de respuesta es importante;
• cuando no se puede tomar el riesgo de proyectos largos;
• cuando no se conocen todos los requerimientos.

2) Explique exhaustivamente el proceso de desarrollo: Prototipado.

• El prototipado aborda las limitaciones del modelo de cascada en la especificación de los requerimientos.
• En lugar de congelar los requerimientos sólo basado en charlas y debates, se construye un prototipo que permita comprender los requerimientos.
• Permite que el cliente tenga una idea de lo que sería el SW y así conseguir mejor feedback de él.
=> Ayuda a disminuir los riesgos de requerimientos.
• La etapa de análisis de requerimientos es reemplazada por una ‘’mini-cascada’’.

Desarrollo del prototipo:
• Comienza con una versión preliminar de los requerimientos.
• El prototipo solo incluye las características claves que necesitan mejor comprensión.
• Es inútil incluir características bien entendidas.
• El cliente “juega” con el prototipo y provee feedback importante que mejora la comprensión de los requerimientos.
• Luego del feedback el prototipo se modi ca y se repite el proceso hasta que los costos y el tiempo superen los beneficios de este proceso.
• Teniendo en cuenta el feedback, los requerimientos iniciales se modi can para producir la especi cación nal de los requerimientos.

Ventajas:
• Mayor estabilidad en los requerimientos.
• Los requerimientos se congelan más tarde.
• La experiencia en la construcción del prototipo ayuda al desarrollo principal.
Desventajas:
• Potencial impacto en costo y en tiempo.
Aplicación:
• Cuando los requerimientos son difíciles de determinar y la con anza en ellos es baja (i.e. los requerimientos no se han comprendido).


Ej. 3. Describa los modelos de proceso de desarrollo denominados *cascada* e *iterativo*. Enuncie una ventaja y una desventaja de ambos modelos.

Iterativo está descrito más arriba, con sus ventajas digamos lo deberías saber…
Modelo de cascada
Secuencia inicial de las distintas fases:
1. Análisis de requerimientos
2. Diseño de alto nivel
3. Diseño detallado
4. Codificación
5. Testing
6. Instalación

Una fase comienza sólo cuando la anterior finaliza (en principio, no hay feedback).
Las fases dividen al proyecto; cada una de ellas se encarga de distintas incumbencias.

Productos de trabajos usuales en este modelo:
• Documento de requisitos / SRS
• Plan del proyecto
• Documentos de diseño (arquitectura, sistema, diseño detallado)
• Plan de test y reportes de test
• Código nal
• Manuales del software (usuario, instalación, etcétera)

Fortalezas	Debilidades	Aplicación
Simple. Fácil de ejecutar. Intuitivo y lógico.	"Todo o nada": muy riesgoso. Req. se congelan muy temprano. Puede escoger hw/ tecno. vieja. No permite cambios. No hay feedback del usuario.	Problemas conocidos. Proyectos de corta duración. Automatización de procesos manuales existentes.

Ej. 3. Elija dos modelos de proceso de desarrollo de los estudiados en la asignatura, y compárelos de acuerdo a sus fortalezas y debilidades.

Con lo anterior deberias poder resolver esto re ez

Ej. 4. ¿Cuál es la principal motivación para tener un proceso de desarrollo de software separado en fases? ¿En qué consiste el enfoque ETVX? Describa brevemente las actividades básicas fundamentales de los procesos de desarrollo de software.

¿Por qué utilizar fases?
• Dividir y conquistar.
• Cada fase ataca distintas partes del problema.
• Ayuda a validar continuamente el proyecto
Cada fase sigue el enfoque ETVX (Entry - Task - Verification - Exit): (Aca capaz deberíamos abordar más el tema de entrada y salida primero antes de meter manuela)
• Criterio de entrada: qué condiciones deben cumplirse para iniciar la fase.
• Tarea: Lo que debe realizar esa fase.
• Verificación: Las inspecciones/controles/revisiones/veri caciones que deben realizarse a la salida de la fase (i.e. al producto de trabajo).
• Criterio de salida: Cuando puede considerarse que la fase fue realizada exitosamente.

Usualmente un proceso de software está compuesto por las siguientes actividades:

Análisis de requerimientos y especificación

Objetivo: comprender precisamente el problema.
Forma la base del acuerdo entre el cliente y el desarrollador.
Especifica el "qué" y no el "cómo".
Salida (Producto de trabajo): Especificación de los requerimientos del software (SRS).

Arquitectura y Diseño

Es el paso fundamental para moverse del dominio del problema al dominio de la solución.
Involucra tres tareas:

Diseño arquitectónico: establece las componentes y los conectores que conforman el sistema.

Diseño de alto nivel: establece los módulos y estructuras de datos necesarios para implementar la arquitectura.

Diseño detallado: establece la lógica de los módulos.

La mayoría de los métodos se enfocan en la arquitectura y/o el diseño de alto nivel.

Salida: Documentos correspondientes.

Codificación

Convierte el diseño en código escrito en lenguaje específico.
Objetivo: Implementar el diseño con código simple y fácil de comprender (legible!).
• La fase de codificación afecta tanto al testing como al mantenimiento. Código bien escrito puede reducir el esfuerzo de testing y de mantenimiento.
Salida: el código.

Testing

Cada fase puede introducir defectos.
La fase de Testing debe encontrarlos y eliminarlos para mejorar la calidad.
Objetivo: Identificar la mayoría de los defectos.
• Es una tarea muy cara: debe planearse y ejecutarse apropiadamente.
Salida: Plan de test conjuntamente con los resultados, y el código nal testeado (y con able).

Entrega e instalación

Ej. 5. Describa cuáles son las diferentes fases del proceso de la administración del proyecto.

Fases:
• Planeamiento
• Seguimiento y control
• Análisis de terminación
El planeamiento es la actividad principal que produce un plan el cual forma la base del seguimiento.

Planeamiento: se realiza antes de comenzar el proyecto.
Tareas claves:
• Estimación de costos y tiempos.
• Seleccionar el personal.
• Planear el seguimiento.
• Planear el control de calidad

Seguimiento y control:
• Acompaña al proceso de desarrollo:
• Tareas:
• Seguir y observar parámetros claves como costo, tiempos, riesgo, así como los factores que los afectan.
• Tomar acción correctiva si es necesario.
• Las métricas proveen la información del proceso de desarrollo necesaria para el seguimiento

Análisis de terminación:
• Se realiza al finalizar el proceso de desarrollo.
• El propósito fundamental es analizar el desempeño del proceso e identificar las lecciones aprendidas.
• En procesos iterativos el análisis de terminación se realiza al finalizar cada iteración y se usa para mejorar en iteraciones siguientes.

Ej. 4. ¿Qué es y para qué sirve el oráculo de test? ¿Qué son y para qué sirven los criterios de selección de tests? ¿Cúal es la diferencia entre el testing de caja blanca y el testing de caja negra?

Para verificar la ocurrencia de un desperfecto en la ejecución de un caso de test, necesitamos conocer el comportamiento correcto para este caso.
i.e. necesitamos un oráculo de test.
• Idealmente pretendemos que el oráculo siempre dé el resultado correcto.
• Muchas veces el oráculo es humano y por lo tanto propenso a cometer errores.
• El oráculo humano usa la especi cación para decidir el “comportamiento correcto”.
• Las mismas especificaciones pueden contener errores.
• En algunos casos, los oráculos pueden generarse automáticamente de la especificación.

Criterios de selección este año no entraba :D

Testing de caja negra
• El software a testear se trata como una caja negra
• La especificación de la caja negra está dada.
• Para diseñar los casos de test, se utiliza el comportamiento esperado del sistema.
i.e. Los casos de test se seleccionan sólo a partir de la especificación.
• No se utiliza la estructura interna del código.
Premisa: el comportamiento esperado está especificado.
• Entonces sólo se definen test para el comportamiento esperado especificado.
• Para el testing de módulos: la especificación producida en el diseño de ne el comportamiento esperado.
• Para el testing del sistema: la SRS define el comportamiento esperado.

Testing de caja blanca

El testing de caja blanca se enfoca en el código:
El objetivo es ejecutar las distintas estructuras del programa con el fin de descubrir errores.
Los casos de test se derivan a partir del código.
Se denomina también testing estructural.

4) Explique el criterio de cobertura de sentencia, dando un ejemplo no mencionado en el teórico.

Criterio de Cobertura de sentencia:

Este criterio está basado en el criterio de flujo de control, que es un criterio para seleccionar el conjunto de casos de test en testing estructural (Caja Blanca)

El criterio basado en flujo de control considera al programa como un grafo de flujo de control.

Los nodos representan bloques de código

Una artista (i, j), representa una posible transferencia de control del nodo i a j

Suponiendo la existencia de un nodo inicial y final, un camino es una secuencia del nodo inicial al nodo final

Entonces el criterio de cobertura de sentencia:

Cada sentencia se ejecuta al menos una vez durante el testing.

Osea el conjunto de caminos ejecutados durante el testing debe incluir todos los nodos

Limitación: Si el error está en la falta de sentencias, puede pasarse por desapercibido el error.

además no es posible garantizar 100% de cobertura debido a que puede haber nodos inalcanzables.

¿Qué es el testing de caja negra? Enuncie al menos dos criterios dentro de esta categoría.

Ej. 6. Explique y de un ejemplo del testing de caja negra de análisis de valores límites

Caja Negra:
No se utiliza el funcionamiento interno del software, se trata como una caja negra al cual le damos una entrada y nos entrega una salida esperada.

El comportamiento está especificado:
Para diseñar los casos de test, se utiliza el comportamiento esperado del sistema.

Se seleccionan casos de test sólo a partir de la especificación.

Para el testing en módulos: la especificación producida en el diseño define el comportamiento esperado

Para el testing del sistema: la SRS define el comportamiento esperado.

Criterios

Particionado por clase de equivalencia:
Particionar los datos de entrada en clases de equivalencia para las cuales se especifican distintos comportamientos.
Osea: la especificación requiere el mismo comportamiento en todos los elementos de una misma clase.

Dividir los datos de salida en clases de equivalencia. (Normalmente son eq. válidas y eq. inválidas) (comportamiento esperado del sistema)

Teniendo estas clases de salida generamos casos de test para cada una eligiendo apropiadamente las clases de entradas.
Dos métodos:

Estos deben cubrir tantas clases de entrada como sea posible,
O dar un caso de test que cubra a lo sumo una clase válida para cada entrada

Análisis de valores límites:
Particionar los datos de entrada en clases de equivalencia para las cuales se especifican distintos comportamientos.
Osea: la especificación requiere el mismo comportamiento en todos los elementos de una misma clase.

Dividir los datos de salida en clases de equivalencia. (Normalmente son eq. válidas y eq. inválidas). (comportamiento esperado del sistema)

Luego tomamos un conjunto de datos de entrada que se encuentra en el borde de las clases de equivalencia entrada o salida.

Para cada clase de equivalencia:

Elegir valores en los límites de la clase, los que están justo fuera y dentro de los límites

Con este conjunto más un valor normal se pueden producir casos de test que generen salidas esperadas sobre ellos.

Si tenemos varias variables con casos límites en cada una:

Dos formas:

Ejecutar todas las combinaciones posibles de las distintas variables.
Si tengo n variables y cada una tiene m casos límites entonces tengo m^n
Seleccionar los casos límites para una variable y mantener las otras en casos normales y considerar el caso de todo normal
Si tengo n variables y cada una tiene m casos límites entonces tengo mn + 1

ejemplo:

Si tenemos datos de entrada en clases de equivalencia de la forma que 1<= n <= 6 y 3 <= m <=4

Los casos límite de n en esta clase son 0.9 y 6.1 fuera del rango y 1.1 y 5.9 dentro del rango.
y
Los casos límite de m en esta clase son 2.9 y 4.1 fuera del rango y 3.1 y 3.9 dentro del rango.

Y supongamos que los datos de salida en clases de equivalencia son válidas para n y m dentro del rango y invalido para n y m fuera del rango

por lo que n y m tiene 4 casos límites más un valor normal.

Por lo que la cantidad de test a realizar es:

método 1: $5^2 = 25$ tests

método 2: 5*2 + 1 = 11 tests

Ej. 8. Explique y de un ejemplo del testing de caja negra de particionado por clases de equivalencia.

Particionado por clases de equivalencia:

Es un método de selección para los casos de test en testing de caja negra

Se divide el espacio de entrada en clases de equivalencia para las cuales se especifican distintos comportamientos para cada una.

También se divide los datos de salida en clases de equivalencia con el valor esperado. Estos son los distintos comportamientos de cada clase de equivalencia de el espacio de entrada.

Luego elegimos los casos de test teniendo en cuenta estas equivalencias.

Método 1: Cada test cubre tantas clases de entrada como sea posible

Método 2: Cada test cubre a lo sumo una clase válida para cada entrada

Además de los casos de test separados por cada clase inválida

Ejemplo:
Si tenemos las variables n y m con n 6 datos de entrada que devuelven una equivalencia válida y 4 clase de equivalencia que devuelven una equivalencia invalida.
m 1 clase de equivalencia de entrada que devuelve una equivalencia válida y 1 clase de equivalencia que devuelve una equivalencia invalida.

Entonces con el método 1:

tenemos casó con n que cumple las 6 clases de entrada que devuelven una equivalencia válida y m que cumple con la clase de entrada que la valida y luego un test por cada clase inválida en n y m

En total tenemos 1 + 5 = 6 tests

Método 2: un test por cada clase válida de n (también clase valida de m) y un test por cada clase inválida en n y m

En total tenemos 6+5 = 12 tests

Ej. 4. Explique para qué sirve el modelo COCOMO. Describa sus pasos y enumere además al menos 2 atributos que influyen en su estimación final.

Ej. 9. Describa los pasos básicos asociados a la aplicación del modelo COCOMO para la estimación de esfuerzos. Enumere además al menos 2 atributos que influyen en la estimación final asociada a este modelo.

El modelo COCOMO es un modelo de estimacion de esfuerzo con las siguientes características:
- Enfoque top-down.
- Utiliza tamaño ajustado con algunos factores.
- Procedimiento:

A) Obtener el estimador inicial usando el tamaño;
B) Determinar un conjunto de 15 factores de multiplicación representando distintos atributos;
C) Ajustar el estimador de esfuerzo escalándolo según el factor de multiplicación nal;
D) Calcular el estimador de esfuerzo de cada fase principal.

Descripcion de los pasos