Clase 6-Codificación

El objetivo es poder implementar el diseño de la mejor manera posible. Y como afecta al testing y al mantenimientos (Actividades que requieren de la lectura del codigo). En la codificación se busca que el código sea fácil de leer y comprender. Por lo que el objetivo principal del programador es escribir programas simples y fáciles de leer con la menor cantidad de errores (bugs) posibles.

Principios y pautas para la programación

Programación estructurada

Inicia en los 70 en contra del uso indiscriminado de los constructores de control. El objetivo es simplificar la estructura de los programas de manera que sea mas fácil razonar sobre ellos.

Un programa tiene una

- estructura estática: es el orden de las sentencias en el codigo
- estructura dinámica: es el orden en el cual las sentencias se ejecutan

Cuando hablamos de corrección de un programa, hablamos de la estructura dinámica. Es por eso que para determinar que un programa es correcto debemos mostrar que el comportamiento dinámico es el esperado.

Para analizar esto debemos razonar sobre el código del programa (Estructura estatica) y como ambas estructuras no son necesariamente iguales suele hacerlo dificil. Por lo que otro objetivo de la programación estructurada es escribir programas cuya estructura dinámica es la misma que la estática.

Los constructores de la programación estructurada son de una única entrada y única salida. Y ademas no son arbitrarios -> deben mostrar un comportamiento claro.

```
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
```

Ocultamiento de la información

La idea es poder ocultar la información de las estructuras de datos de manera que solo quede expuesta a las pocas operaciones que se realizan sobre esa información para que el programa pueda realizar ciertas funciones.

El ocultamiento **reduce el acoplamiento**. Y es una practica fundamental en OO y uso de componentes.

Algunas prácticas

- Ocultamiento
- Interfaz del modulo

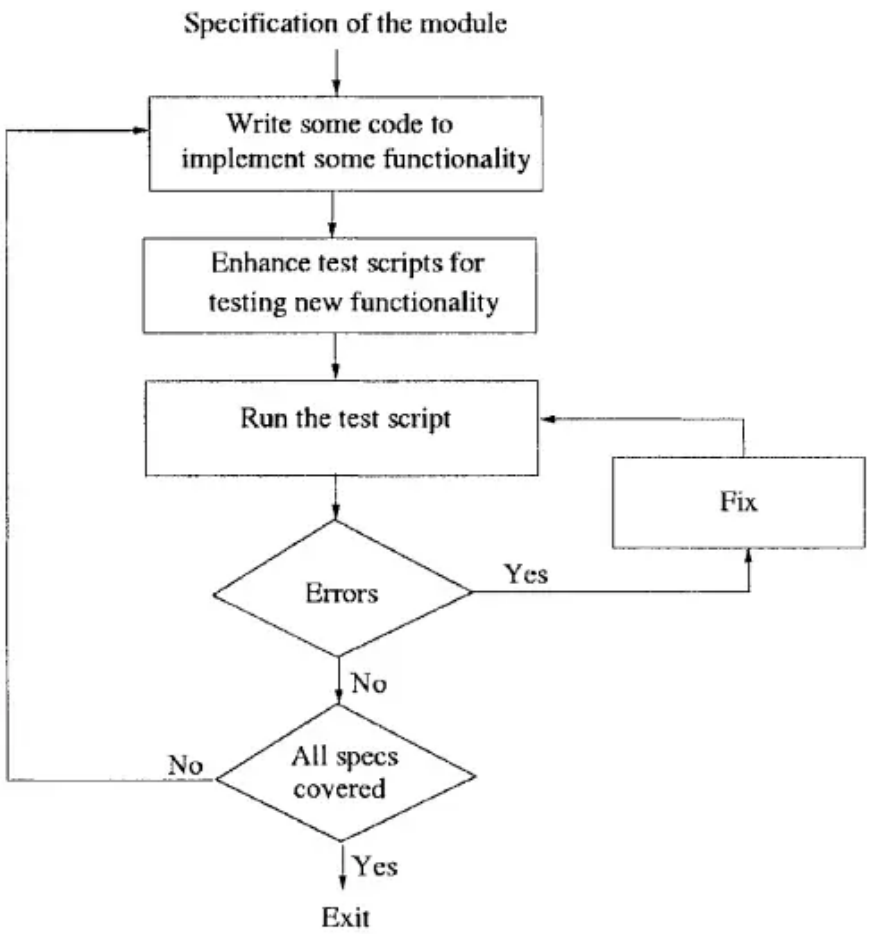
- Tamaño de los módulos no muy largos
- Robustez: valores de retorno en lecturas
- Evitar efectos secundarios
- Fuentes de datos confiables.
- Dar importancia a las excepciones.

Proceso de codificación

La codificación comienza ni bien esta disponible la especificación del diseño de los módulos. Por lo gral los módulos se asignan a programadores individuales.
El desarrollo puede ser:

- top-down: los módulos superiores se desarrollan primero
- bottom-up: los módulos inferiores se desarrollan primero.

Proceso de codificación incremental.



El proceso de codificación incremental toma como entrada la especificación del modulo, el programador implementa alguna funcionalidad, crea scripts de test para esa funcionalidad y los corre.

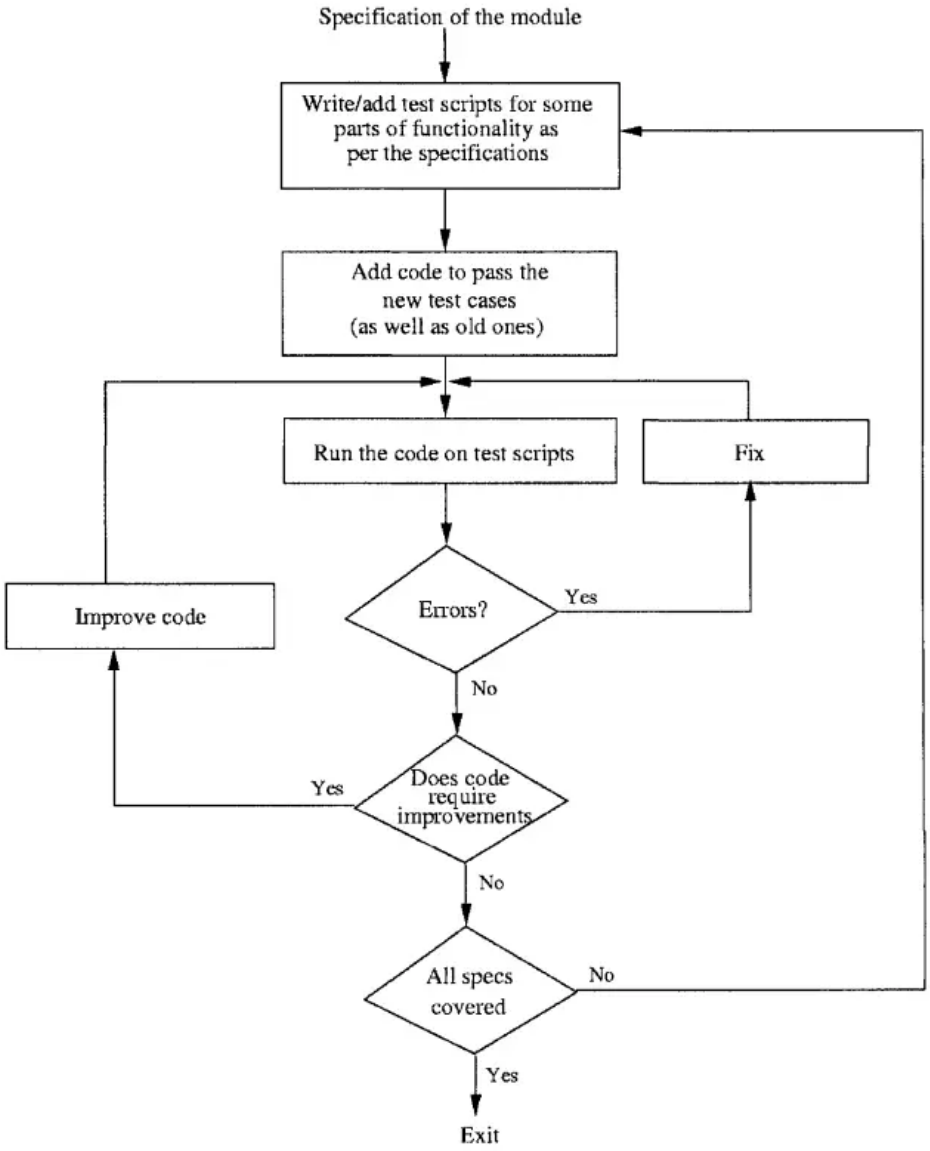
- Si no hay errores y todas las especificaciones estan cubiertas. Entonces termino
- Si no hay errores, pero no estan cubiertas todas las especificaciones, entonces vuelve al paso 1, "escribir codigo para implementar funcionalidades". y repite

- Si hay errores entonces los arregla y vuelve a correr los test hasta que no haya errores. De ahi si estan cubiertas todas las specs termino, sino vuelve a 1 y repite.

Desarrollo dirigido por test

En este proceso se invierte el orden de las actividades. Primero escribe scripts para los test y luego el código para poder cumplir con la funcionalidad que ese test requiere. Es también un desarrollo incremental.

La responsabilidad de asegurar cobertura de toda funcionalidad radica en el diseño de los casos de test y no en la codificación. Ayuda a asegurar que todo código es testeable. El enfoque esta en cómo sera usado el código a desarrollar, ayuda a validar la interfaz del usuario especificada en el diseño.



Toma como entrada la especificación del modulo y se escriben scripts de test para algunas funcionalidad es según la especificación. Se crea código para pasar los casos de test de esas funcionalidades y se corren los test:

- Si hay error, se arreglan y se vuelven a correr hasta que no haya errores
- Si no hay errores, se pregunta si el código requiere de mejoras, si las requiere entonces se lo mejore y se vuelve a correr los test.
- Si no requiere de mejoras se pregunta si todas las especificaciones están cubiertas, si no estan todas cubiertas se vuelve al paso 1, donde se crean nuevos script de test para nuevas funcionalidades dentro de las especificaciones del modulo.

- Cuando todas las especificaciones estén cubiertas se termino.

Programación de a pares

El código es escrito por dos programadores en lugar de uno solo. La pareja diseñan los algoritmos, estructuras de datos, estrategias ,etc.
Una persona tipea el código mientras la otra revisa activamente el mismo(van rotando). Se van marcando los errores y conjuntamente se formulan soluciones.

Este código esta bajo constante revision por lo que deriva en mejor diseño de algoritmos/estructuras de datos/lógica. Ya que es más difícil que se escapen las condiciones particulares.

Refactorización

```
La refactorización es una tecnica que permite realizar cambios en un programa con el fin de simplificarlo y mejorar su comporesion y mejorar el diseño del codigo existente (hacerlo testeable y mantenible), sin cambiar el comportamiento observacional de este.
```

Surge a partir de que al complicarse el diseño, comienza a hacerse mas complicado modificar el código y es más susceptible a errores.

- la estructura interna del software cambia
- el comportamiento externo permanece igual

Conceptos basicos

Busca lograr:

- reducir el acoplamiento
- incrementar la cohesion
- mejorar la respuesta al principio de abierto-cerrado
  Nunca se debe modificar la funcionalidad ni alterarla.

Los cambios por refactorización se realizan separadamente de la codificación normal.

El principal riesgo es introducir nuevos errores. Por lo que se debe realizar en pequeños pasos y se debe disponer de test automatizados para testear la funcionalidad existente. Permite que el diseño del codigo mejore continuamente.

Malos olores

Son algunos indicios de que se deba necesitar refactorización:

- código duplicado, misma funcionalidad en lugares distintos.
- Método largo.
- Clase grande
- Lista larga de parámetros.

- Demasiada comunicación entre objetos.
- Encadenamiento de mensajes.

Las **refactorizaciones más comunes** se enfocan en mejoras de:

- métodos
- clases
- jerarquía de clases.
  Siempre bajo el mismo objetivo: mejorar acoplamiento, cohesion y principio A-C.

Las **mejoras de métodos** pueden ser:

- Extracción de métodos
  - Si es demasiado largo, se busca separar en métodos cortos cuya signatura indique lo que el método hace.
  - Partes del código se extraen como nuevos métodos. Las variables referenciadas se transforman en parámetros.
  - También se realiza si un método retorna un valor y también cambia el estado del objeto.
- Agregar/eliminar parámetros:
  - objetivo: simplificar las interfaces donde sea posible.
  - Agregar solo si los existentes no proveen la info necesaria.
  - Eliminar si no se utilizan.

Las **mejoras de clases** pueden ser:

- Desplazamiento de métodos: mover un método de una clase a otra
  - se realiza cuando el método actúa demasiado con los objetos de otra clase.
- Desplazamiento de atributos: si un atributo se usa más en otra clase, moverlo a esa
  - Mejora cohesion y acoplamiento.
- Extracción de clases: Si una clase agrupa múltiples conceptos, separar cada concepto en una clase
  - Mejora cohesión
- Remplazar valores de datos por objetos:
  - Una colección de atributos se transforman en una entidad lógica
  - Separarlos como una clase y definir objetos para accederlos.

Las **mejoras de jerarquías** pueden ser:

- Remplazar condicionales con polimorfismos:
  - Si el comportamiento depende de algún indicador de tipo, no se está explotando el poder de la OO
  - Remplazar 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 esta duplicado en las subclases, pueden subirse a la superclase.