

# INGENIERÍA DEL SOFTWARE

## PARCIAL II

### CODIFICACIÓN (cap. 9)

El objetivo de la codificación es implementar el diseño de la mejor manera posible. Esta implementación afecta al testing y al mantenimiento, por lo que el pronóstico de la codificación es reducir los costos de ambos, ya que estos son muy altos. Para que esto suceda, el código debe ser fácil de leer y comprender (no necesariamente de escribir).

#### Principios y pautas para la programación

Estos principios y pautas ayudan al programador a cumplir su objetivo: escribir código de alta calidad, es decir, fácilmente comprensible, testeable y manejable.

## Programación estructurada:

El objetivo es simplificar la estructura de los programas de manera de ser fácil razonar sobre ellos. Para esto se apunta a escribir programas cuya estructura dinámica sea la misma que la estructura, siendo la estructura dinámica cómo se ejecuta el código y la estructura cómo escribirlo en el código.

Los constructores de este principio son de única entrada y única salida, de manera de la ejecución de las sentencias se realice en el orden en que aparecen en el código.

Entonces => La PE simplifica el flujo de control, facilitando la comprensión de los programas y su razonamiento. Esto a través de igualar estructuras dinámicas y estructurales.

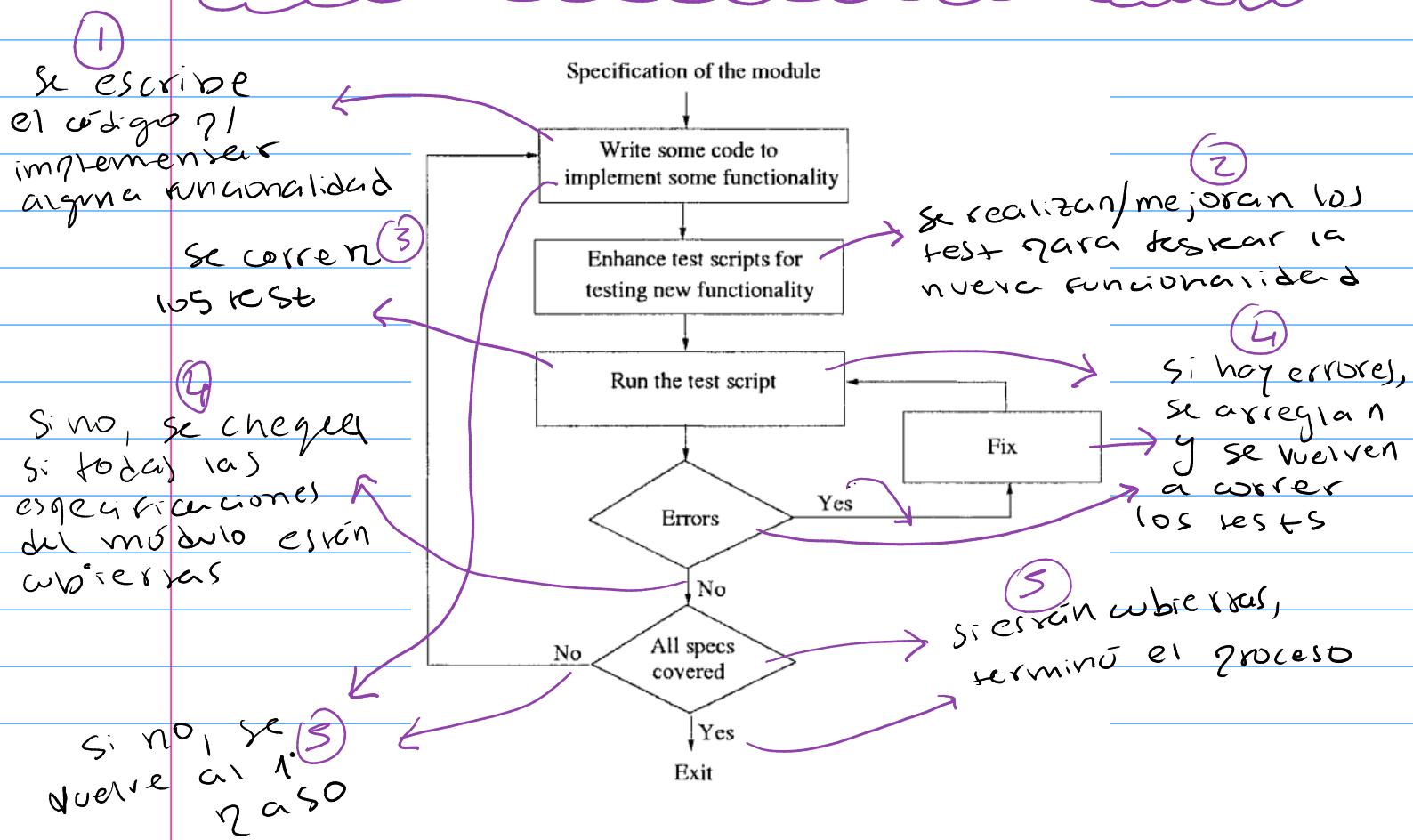
## Ocultamiento de la información

Consiste en ocultar la información que guardan las estructuras de datos consideradas en el código de manera que solo sea expresada a las operaciones que la manipularán. Esta regla o principio reduce el acoplamiento.

## El proceso de codificación

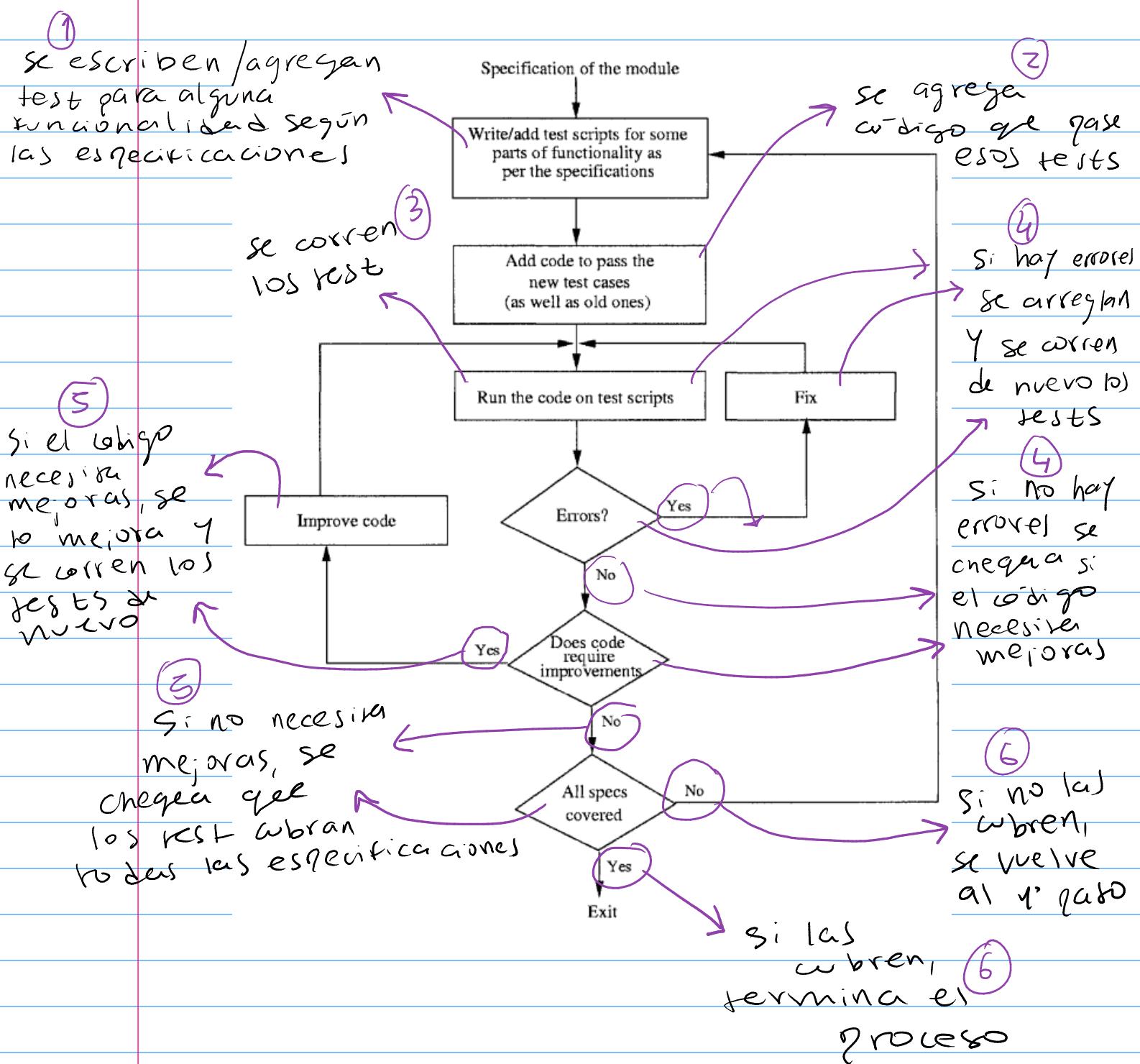
Para la codificación se quedan utilizar diferentes procesos.

### Proceso de codificación incremental



# Desarrollo dirigido por test (TDD)

También se realiza incrementalmente, la diferencia es que la responsabilidad de cubrir todos los especificaciones radica en el diseño de los test y no en la codificación.



# Programación de a pares

El código es escrito por dos programadores:

- Ambos diseñan algoritmos, estructuras de datos, estrategias, etc.
- Uno escribe el código
- Otro revisa aproximadamente el código y se corrige

Si bien se puede decir que mejora el diseño de los algoritmos (estructuras) lógicas y reduce la posibilidad de que se escapen condiciones particulares, la efectividad de este método no es aún bien sabida, se cuestiona si genera ciertas pérdidas de productividad.

## Refactorización

Es una técnica para mejorar el diseño del código existente. No agregar nuevas funcionalidades o características. Se aplica a código que ya está funcionando y que no tiene bugs durante el proceso de codificación, sin embargo, no es lo mismo que codificación normal, se realizan separadamente.

Es la tarea que permite realizar cambios en un programa con el fin de simplificarlo y mejorar su comprensión, es decir, hacerlo testable y comprensible sin cambiar el comportamiento observacional de este.

→ la estructura interna del software cambia.

→ el comportamiento externo permanece igual.

Intenta lograr

- reducción del acoplamiento
- incremento de la cohesión
- mejora de la resguardo al principio abierto-cerrado

Durante la refactorización puede "romperse" la funcionalidad existente, para disminuir este posibilidades podemos:

- refactorizar en pequeños pasos.
- disponer de tests automatizados para testear la funcionalidad existente.

## Refactorizaciones más comunes

Las formas de mejorar el diseño de los programas se enfocan en

### \* métodos:

**extracción de métodos:** Se realiza si el método es demasiado largo, el objetivo es tener métodos cortos cuya firma indique lo que el método hace.

**Agregar/eliminar parámetros:** el objetivo es simplificar las interfaces. Se agregan si los parámetros existentes no proveen la info necesaria y se eliminan si no están siendo utilizados.

### \* Clases:

**desglazamiento de métodos:** Se realiza cuando un método interactúa demasiado con objetos de otra clase.

**desglazamiento de atributos:** Se realiza cuando un atributo se usa más en otra clase que en la actual. Mejora acoplamiento y cohesión.

**extracción de clases:** Se realiza cuando una clase agrupa múltiples conceptos, se separa cada uno en una clase distinta. Mejora cohesión.

### reemplazar valores de datos

**por objetos:** Se separa como una clase una colección de atributos que se transformaron en una entidad lógica. Se definen objetos para accederlos.

## \* jerarquías:

reemplazar condicionales con polimorfismos: encontrará una jerarquía de clases que podrían para reemplazar los análisis de caso) se desglosan de algún indicador de bajo.

### subir métodos/atributos:

Si hay un método/atributo que aparece en subclases, que debe (debe) subirse a la superclase

## MODELOS DE PROCESO DE DESARROLLO (cap. 2 - 2º PT)

Un modelo de proceso provee una estructura genérica de los procesos que puede seguirse en algunos proyectos con el fin de alcanzar sus objetivos.

Especifica un proceso general como un conjunto de etapas.

Cuando se elige un modelo y se lo adapta al proyecto, se produce la especificación del proceso del proyecto.

- **modelo del proceso:** espec. genérica del proc.
- **especificación:** plan de lo que debe ejecutarse
- **proceso:** lo que se ejecuta

modelos comunes...

## Modelo de cascada

Es el modelo más simple, sus fases están organizadas en orden lineal, es decir, una comienza sólo cuando la anterior finaliza. Cada una de ellas se encarga de distintas tareas.

Secuencia inicial de las fases:

1. Análisis de requerimientos, se sugiere con el planeamiento.
2. Diseño de alto nivel
3. Diseño detallado
4. Codificación
5. Testing
6. Instalación

El final de una fase y el comienzo de otra es claramente identificado por un mecanismo de certificación: verificación y validación.

- ↳ Cada fase debe producir una salida que sea evaluada y certificada, suelen llamarse productos de trabajo, algunos son:
- documento de requisitos / SRS
  - plan de proyecto (detail design)
  - documentos de diseño: arqu., sist., DD
  - plan de test y reportes de test
  - código final
  - manuales de software: user, instalación, etc.

## Ventajas:

- concepcionalmente simple, división del problema en fases se pueden realizar independientemente.
- fácil de entender, fronteras bien definidas entre cada fase.
- intuitivo y lógico.

## Desventajas:

- riesgoso porque es "todo o nada", el sistema se desarrolla todo "de una".
- se concretan los requisitos muy temprano.
- posibilidad de elegir tecnologías/hardware viejo.
- no permite cambios.
- no hay feedback del usuario entre fases.

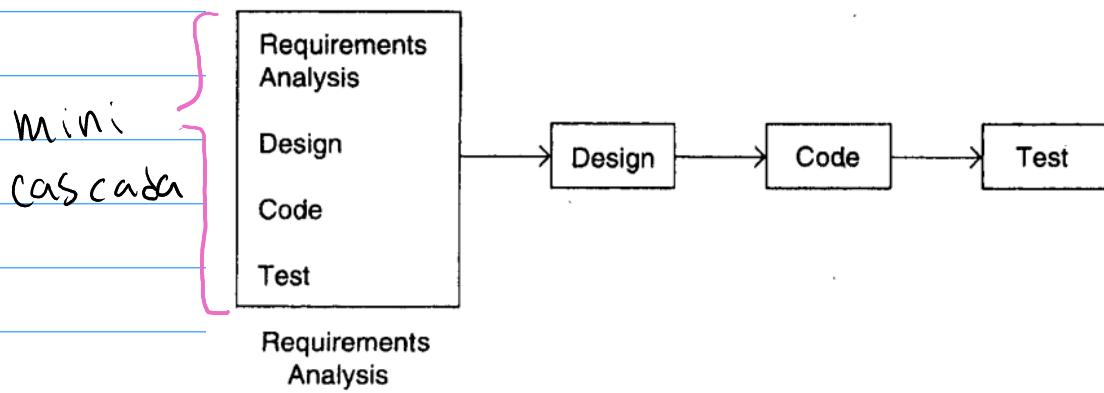
## aplicación: (muy usado)

- problemas conocidos
- proyectos corídos
- automatización de procesos manuales existentes.

# Prototipado

Se construye un prototipo que permite comprender los requerimientos, esto aborda las limitaciones del modelo de cascada en la especificación de los mismos.

⇒ disminuye los riesgos de requerimientos



## desarrollo del prototipo

→ Comienza con una versión preliminar de los requerimientos.

→ Sólo incluye características claves que necesitan mejor comprensión.

→ El cliente usa el prototipo y provee feedback.

→ Se modifican el prototipo hasta que los costos y tiempo superen los beneficios del proceso.

→ Los requerimientos iniciales se modifiquen teniendo en cuenta el feedback.

EL PROTOTIPO DEBE DESCARTARSE

## Ventajas:

- ayuda a la reelección de requerimientos
- reduce el riesgo.
- mayor estabilidad en los requerimientos.
  - requerimientos se congelan más tarde.
  - experiencia en la construcción del producto ayudan al desarrollo principal.
    - ↳ sistemas finales mejores y más estable.

## desventajas:

- potencial impacto en costo y tiempo (aumento).
- comienzo lento.
- no permite cambios tardíos.

## aplicación:

- requerimientos no se han comprendido.
- sistemas con usuarios novatos.
- interfaces con el usuario son muy importantes.

## Desarrollo iterativo

Consiste en desarrollar y entregar el software incrementalmente, abordando el problema de "todo o nada" del modelo de cascada. "Secuencia de cascadas"

Cada incremento es completo en sí mismo, se testeaa y se recibe feedback luego de qd se completa.

MODELO CON MEJORA ITERATIVA:

Primer paso  $\Rightarrow$  se hace una implementación simple para un subconjunto del problema complejo (sólo aspectos clave y fáciles de entender)

Se crea una lista de control del proyecto (LCP) qd contiene, en orden, las tareas qd se deben realizar para lograr la implementación final.

En cada paso se elimina la siguiente tarea de la lista haciendo diseño, implementación y análisis del sistema parcial. Luego se actualiza la lista.

Se repite hasta vaciar la lista.

## MODELO EN ESPIRAL

1. Identificar objetivos, alternativas y restricciones
2. Evaluar alternativas en base a eso, considerando riesgos. Desarrollar estrategias para resolverlos / reducirlos.
3. Desarrollar y verificar el SW.
4. Planear próximo paso.

en gral...

### Ventajas:

- Entregas regulares y rápidas.
- Reduce riesgos.
- acelera cambios naturalmente.
- permite feedback del usuario.
- Prioriza requisitos.

### desventajas:

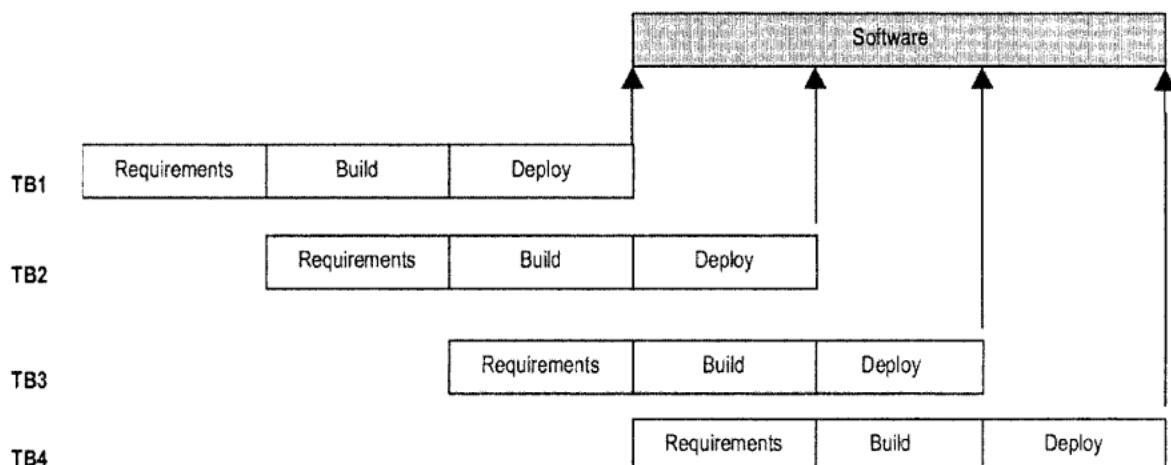
- Sobrecarga de planeamiento en cada iteración
- posible incremento del costo total (el trabajo hecho en una iteración puede deshacerse en otra)
- arquitectura y diseño suelen ser afectados con tantos cambios.

## aplicación

- en empresas donde el tiempo de respuesta es esencial
- cuando no quede enfrentarse el riesgo de proyectos largos
- cuando los requerimientos son desconocidos y sólo se comprendrán con el tiempo

## Timeboxing

Es una secuencia lineal de iteraciones, en donde cada iteración es una mini cascada. Cada fase de esta mini cascada tiene la misma duración y se usa zigzeling para ejecutar las iteraciones en paralelo.



**timeboxer**: el iteración con mini cascadas adentro. Todas tienen la misma duración.

Cada etapa de la timebox puede desarrollarse independientemente.

Hay un equipo para cada etapa.

Un equipo finaliza y le pasan la salida al equipo de la siguiente etapa.

El cronograma no es negociable.

### Ventajas:

- todos los del iterativo
- facilita el planeamiento y la negociación
- ciclo de entrega muy corto

### desventajas:

- la administración del proyecto se vuelve compleja.
- posibilidad de incremento de los costos
- equipos de trabajo muy grandes

### aplicación:

- donde son necesarios tiempos de entrega muy cortos.
- cuando hay flexibilidad en agrupar características.

# EL PROCESO DE SOFTWARE

(Cap. 8-1-nt)

producto  $\Rightarrow$  resultado de ejecutar un proceso

en ese se enfoca  
la IS

Uno adecuado ayuda a lograr los objetivos del proyecto con menor C&P.

Un modelo de proceso especifica un proceso general, con fases en las que el proceso debe dividirse, conjuntamente con otras restricciones y condiciones para la ejecución de las fases.

$\hookrightarrow$  el proceso real se adapta al modelo de proceso.

## Dos procesos fundamentales:

• Desarrollo: se enfoca en las actividades para el desarrollo y para garantizar la calidad necesaria para la ingeniería del SW.

• Administración del proyecto:  
se enfocan en el planeamiento y control del proceso de desarrollo con el fin de cumplir los objetivos.

El proceso es un conjunto de fases. Cada una realiza una tarea bien definida y produce una salida

→ producto de trabajo

entidad formal y tangible capaz de ser verificada

## Enfoque "ETVX"

Cada fase del proceso de desarrollo del SW sigue el enfoque ETVX que sirve para resolver el problema de cuándo una fase debe empezar y terminar.

ETVX es por Entry, Task, Verificación y exit, con la especificación del criterio de entradas y salidas sabemos el momento de inicio y fin de las fases.

- Criterio de entradas (Entry): qué condiciones deben cumplirse para iniciar la fase
- Tarea (task): qué debe realizarse en la fase
- Verificación (Verification): inspecciones/controles que deben realizarse al producto de trabajo
- Criterio de salida (exit): cuándo puede considerarse que la fase fue realizada exitosamente.

ii) El criterio de entrada de una fase debe ser **coherente** con el criterio de salida de la fase anterior. !!

## Otros procesos del Software

El proceso de desarrollo es el **proceso principal** sobre el cual rinden los otros procesos, esto influye en los métodos que usan estos otros procesos.

### Proceso para la administración del proyecto

Se encarga de **asegurar** la ejecución eficiente de las **fases** del proceso de desarrollo. Consiste en **3 fases**:

**Planeamiento:** actividad principal que produce un plan el cual forma la base del seguimiento. Se realiza antes de comenzar el proyecto.

**Tareas claves:**

- estimación de costos y tiempos
- selección de personal
- planeamiento del seguimiento
- " del control de calidad

## Seguimiento y control:

### Tareas:

- seguir y observar parámetros clave como: costo, tiempo, riesgos, así como factores que los afecten
- tomar acciones correctivas de ser necesario

Son las métricas del proceso de desarrollo que proveen la info para esta fase.

**Análisis de terminación:** se realiza al finalizar el proceso de desarrollo.

**Prognóstico fundamental:** analizar el desempeño del proceso, identificar lo aprendido

En procesos iterativos se realiza al finalizar cada iteración.

## Proceso de inspección

**Objetivo:** detectar defectos en los productos de trabajo.

Mejora calidad y productividad.  
Los defectos deben eliminarse en cada etapa.

**Proceso estructurado con roles**

realizado por personal técnico

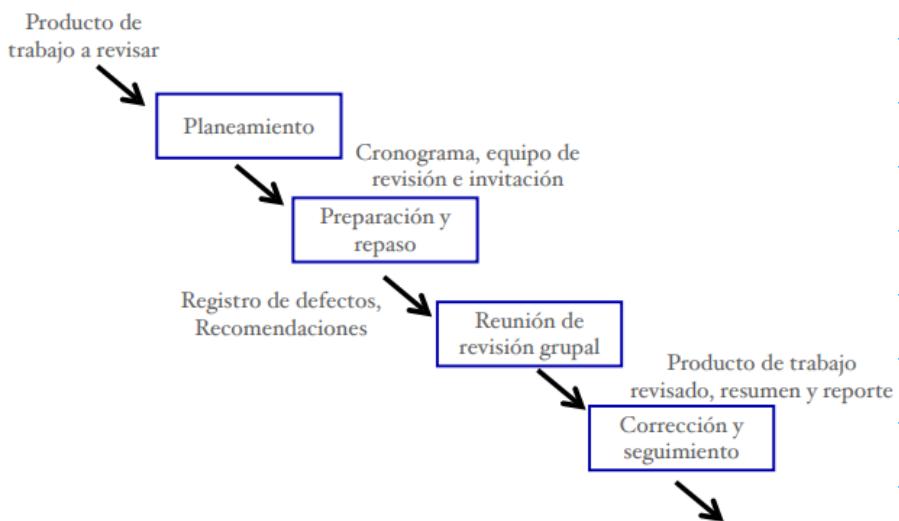
definidos; para personal técnico

agregar  
nuevos!!  
=

- moderador: responsabilidad gral.
- autor: quién realizó el producto de trabajo
- revisor: quién identifica los defectos
- Tector: lee linea a linea el producto de trabajo para enfocar el progreso de la reunión
- escriba: registra las observaciones indicadas.

Foco en encontrar problemas, no resolverlos

FASES:



### Planeamiento:

entra  
salida tareas:

- Seleccionar equipo de revisión.
- Identificar al moderador.
- Preparar el paquete para la distribución:
  - Producto de trabajo a revisar
  - Sus especificaciones
  - Lista de control
  - Estándares



## Preparación y regreso (overview):

Consiste en una breve reunión (opcional) donde se entrega el paquete, se explican el propósito de la revisión y se introducen las áreas de acuerdo.

Todos los miembros revisan individualmente el producto de trabajo para identificar defectos; se usan listas de control, gautas y estándares.

## Reunión de revisión grupal:

Propósito: definir la lista final de defectos.

Criterio de entrada: revisiones individuales apropiadas, estas son revisadas por el moderador.

### Reunión:

- Se lee línea a línea el producto de trabajo.
- Se efectúa cualquier observación que hubiere (que no se ha visto).
- Se discute para identificar el defecto.
- El escritor registra la decisión.

### A final de la reunión:

- Escritor presentar la lista de defectos y observaciones.
  - Pocos defectos => el producto se acepta sin que se realice otra revisión
  - Se prepara un resumen de inspección. Se usa para analizar la efectividad de la revisión.

## Corrección y seguimiento:

Los ~~defectos~~ son corregidos por el autor. Y luego obtiene la aprobación del moderador o el producto se somete a una nueva revisión.

La revisión finaliza cuando los defectos/observaciones fueron satisfactoriamente procesados.

## Proceso de administración de configuración (SCM)

La administración de configuración del software controla sistemáticamente los cambios producidos en los ítems producidos en el proyecto de su tales como programas, documentos, datos, etc. Se enfoca en los cambios durante la evolución, los cambios de req. se manejan a parte.

A medida que estos ítems van cambiando generan nuevas versiones, la SCM debe asegurar que estos se combinen sin pérdidas.

### Funcionalidades necesarias:

- Recolección de fuentes, documentos y otra info del sistema actual.
- Evitar cambios o eliminaciones desautorizadas.

- Deshacer cambios o revertir a una versión específica
- Hacer disponible la última versión del programa.

### Mecanismos principales:

- Control de acceso
- Control de versiones
- Identificación de la configuración

### Ítems de la configuración:

Cada ítem es una unidad de modificación y se modifica intensamente, esas modificaciones se siguen rigurosamente.

Las distintas versiones deben combinarse apropiadamente de manera periódica.

Baseline: arreglo apropiado de ítems de configuración. Es establece puntos de referencia a lo largo del desarrollo del sistema. Captura el estado lógico del sistema y forma la base de cambios posteriores.

### Control de versiones:

Auxilia a preservar viejas versiones y a deshacer cambios.

### Control de acceso:

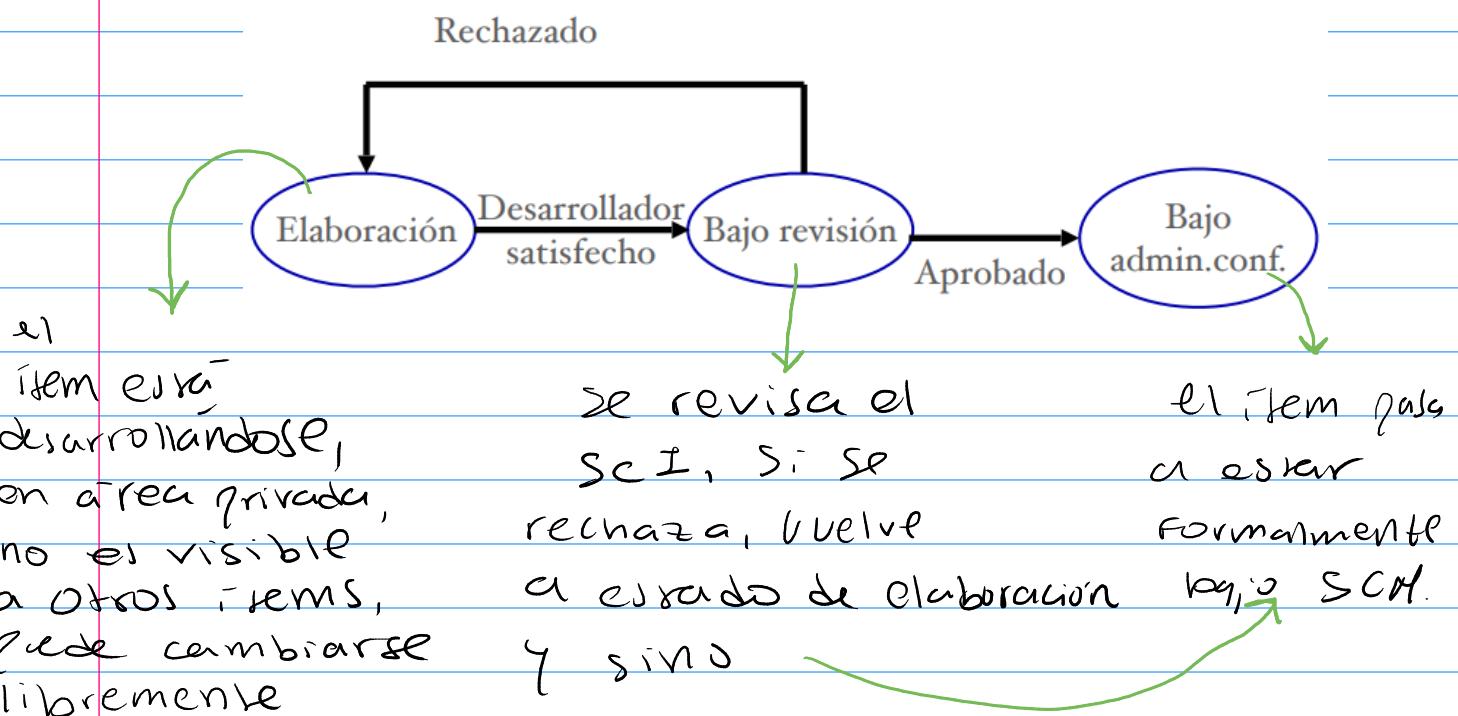
Auxiliado por herramientas que limitan el acceso a personal electrónico. Procedimientos check-in, check-out.

ejemplo: el programador desarrolla el código en área privada.

Para ponerlo en disposición de otro se lo ingresa a un repositorio con acceso controlado.

Para modificar un ítem del repositorio se lo toma de allí (check-out) Los cambios realizados se incorporan con check-in.

• Ciclo de vida de un ítem de configuración:



### Proceso de admin. de config.

Se definen las actividades que regirán control de cambio.

fases:

## Planeamiento:

- identificar ítems
- definir la estructura del rego
- definir control de acceso, niveles de referencia, reconciliación, procs.
- definir procedimientos de publicación.

## Ejecución:

- realizar procedimientos según lo establecido en planeamiento

**Auditoría:** Se verificar que no se cometieron errores.

## Proceso de administración de cambios de requisitos

Busca controlar / administrar los cambios en los requerimientos.

ya qd  
Estos cambios producen impactos en los productos de trabajo y en los ítems de configuración.

### Proceso:

- registrar los cambios, estos se inicián a través de un requerimiento de cambio.
- realizar el análisis de impacto sobre los productos de trabajo y los ítems (identificar cambios necesarios en c/u y su naturaleza).

- estimar impacto en esfuerzo y cronograma.
- analizar impacto con las personas involucradas.
- regreses los productos de trabajo y los ítems.

El impacto del cambio es analizado para decidir si hacerlo efectivo o no.

## Proceso de administración de procesos

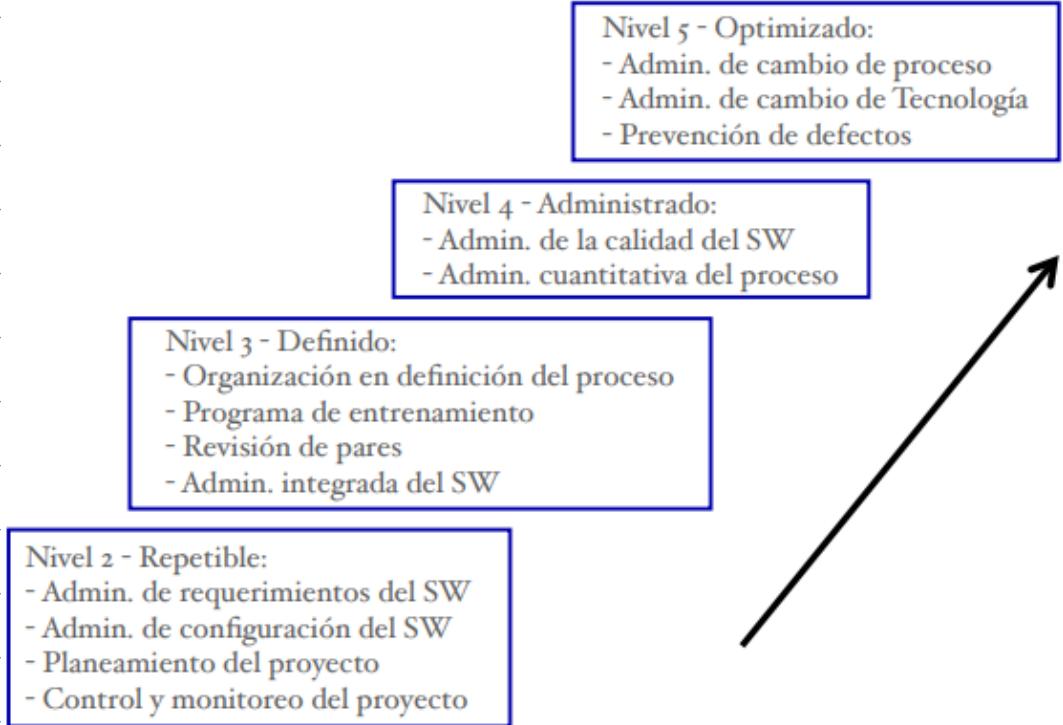
Se enfoca en la evaluación y mejora del proceso.

Los cambios realizados al proceso para su mejora deben hacerse incrementalmente, de a pequeños pasos y deben seleccionarse cuidadosamente.

Para esto existen marcos que sugieren formas de proceder en esa mejora del proceso.

## CMM (Capability Maturity Model)

5 niveles, el 1º es ad-hoc. En cada uno el proceso tiene ciertas capacidades y establece las bases para pasar al siguiente nivel, para ello CMM especifica áreas en las cuales enfocarse para mejorar el proceso.



Primero se evalúa en ~~se~~ nivel este el ~~proceso~~ actual y a partir de eso se chequea ~~se~~ ~~áreas~~ mejorar del proceso para pasar al siguiente nivel.

## TESTING (cap. 10)

El principal objetivo de un proyecto es desarrollar un producto de alta calidad con alta productividad. Para que tenga alta calidad, el producto entregado debe tener la menor cantidad de defectos posible.

Un **desperfecto** ocurre cuando el comportamiento del software no es el esperado, mientras que un **defecto** es lo que causa el desperfecto.

- Desperfecto implica la presencia del defecto.
- Defecto no implica la ocurrencia del desperfecto, pero si tiene potencial para causarlo.
- Qué se considera desperfecto depende del proyecto.

### Rol del testing:

Garantizar calidad mediante la identificación de defectos.

### ¿Cómo?

Un programa se ejecuta siguiendo un conjunto de casos de test. Desperfectos en la ejecución

→ defectos en el SW.

Como estos defectos se detectan a través de los desperfectos, para detectarlos, debemos **causar** desperfectos durante el **testing**.

Para identificar el defecto debemos recurrir al **debugging**.

## Oráculo de test

El oráculo de test es aquello que conoce el comportamiento correcto para algún caso de test, se ejecute; idealmente, de ese siempre dé el resultado correcto.

En algunos cursos estos quedan generarse automáticamente de la especificación.

### Casos de test y criterios de selección

Datos con los que voy a correr el test (gralmente, valores de variables)

queremos que la ejecución satisfactoria

de ellos implique la ausencia de defectos.

Tmb queremos que sea un conjunto reducido

Especifican las condiciones que el conjunto de casos de test debe satisfacer con respecto al programa y/o a la especificación.

ideas

contradicciones

para ello usamos

Propiedades fundamentales que esperamos de los criterios de selección:

Confiabilidad y validez (casi imposible)

Enfoques para diseñar casos de test  
Estos deben ser **destructivos**.

Hay **dos** enfoques para diseñarlos.  
Ambos son complementarios.

## Testing de caja negra (funcional)

El SW se trata como una **caja negra**, es decir, no se utiliza la **estructura interna** del código.

Los **casos de test** se seleccionan **solo** a partir de la **especificación**, se utiliza el **comportamiento esperado** del **sistema**.

Para el **testing de módulos**, este se define en la **especificación** producida en el **diseño**.

Para el **testing del sistema** el **comportamiento** es **definido** en la **SRS**.

Testear el SW con todos los elementos del espacio de entrada es muy caro. Hay **diferentes enfoques** para seleccionar los **casos de test**.

## Partitionado por clase de equivalencia

Se divide el espacio de entradas en clases de equivalencia. Si  $c_1$  funciona para un caso de test en esta clase, muy probablemente funcione de la misma manera para todos los elementos de la misma clase.

Para partitionar, debemos identificar las clases para las cuales se especifican distintos comportamientos

↳ La base lógica es que la especificación requiere el mismo comportamiento en todos los elementos de la misma clase.

Para lograr robustez, deben de armarse clases de equivalencias para entradas inválidas.

También deben considerarse las clases de equivalencia de los datos de salida.

Una vez elegidas estas clases, hay dos métodos para seleccionar los casos de test.

1- Seleccionar cada caso cubriendo tantas clases como sea posible

2- Dar un caso que abra a lo sumo una clase válida que entraña

A estos se le suman los casos de test ~~segurados~~ y ~~el~~ clase inválida.

## Análisis de valores límites

Los ~~programas~~ generalmente fallan sobre ~~valores especiales~~, estos usualmente se encuentran en los ~~límites~~ de las ~~clases de equivalencia~~.

También denominados casos extremos, los casos de test con valores límites tienen ~~alto~~ rendimiento.

⇒ Un caso de test de valores límites es un conjunto de datos de entradas que se encuentra al borde de las clases de equivalencia (de la entrada o salida)

Para cada clase

- Elección de valores en los límites
- " " " justo fuera y dentro de los límites

- Elección de un valor normal

También deben considerarse las salidas y generar casos de test que produzcan señales sobre los límites.

Primero se determinan los valores a utilizar para cada variable Entrada con rango definido

↳ 6 valores límite + un valor normal = 7

Múltiples entradas

|| ↳ Todas las combinaciones de las distintas variables ( $n$ ) =  $7^n$

↳ Casos límite en una variable y casos normales en las otras y caso normal en todo =  $6 \cdot n + 1$

## Grafo de causa-efecto

Ayuda a seleccionar las combinaciones de las clases de equivalencia como condiciones de entrada.

Para ello:

1- Identificar causas y efectos en el sistema:

• Causa: condiciones en la entrada que pueden ser verdaderas o falsas

• Efecto: condiciones de salida tmb V/F.

2- Identificar cuáles causas quedan producir gé efectos. Las causas se pueden combinar.

Causas y efectos son nodos.

Aristas determinan dependencia (pueden ser + o -).

Hay nodos  $\wedge$  y  $\vee$  para combinar causas.

A partir del grafo se debe armar una tabla de decisión, se listan las combinaciones que hacen efectivo el efecto. Esta tabla queda usarse para armar los casos de test.

## Testing de a pares

Usualmente muchos parámetros determinan el comportamiento del sistema, estos deben ser entradas o salidas se deben tomar distintos valores.

defectos  
verificando  
distintos  
valores de  
los parámetros

Así como muchos defectos involucran un sólo parámetro (defectos de modo simple), hay otros que involucran combinaciones de ellos (modo múltiple)

casos de  
test q  
combinaciones  
estos no  
fascible  
es muy raro  
y costoso

Se investigó q la mayoría de estos defectos se revelan con la interacción de pares de valores o sea, la mayoría de los defectos son de modo simple o doble.

Entonces, el testing de a zares consiste en ejercitarse con uno de estos casos.

El menor conjunto de casos de test se obtiene cuando cada zar es cubierto sólo una vez. Existen eficientes algoritmos para la generación de este conjunto de forma que provea cobertura completa.

## Testing basado en estados

se enfoca  
en el  
testing  
de  
transi-  
ciones

En muchos sistemas el comportamiento y la salida del mismo dependen tanto de las entradas como del estado actual del sistema.

Este estado refleja el impacto acumulado de las entradas pasadas. Este testing está dirigido a tales sistemas.

El modelo de estados del sistema se construye, generalmente, a partir de los estados lógicos de interés del mismo. Este modelo tiene cuatro componentes:

**Conjunto de estados:** estados lógicos representando el impacto acumulativo del sistema.

**Conjunto de transiciones:** cambio

de estado en respuesta a algún evento de entrada.

**Conjunto de eventos:** entradas al sistema.

**Conjunto de acciones:** salidas producidas en respuesta a los eventos de acuerdo al estado actual.

Este modelo muestra la ocurrencia de las transiciones y las acciones.

El desafío es identificar el conjunto de estados que captura las propiedades claves y que es lo suficientemente pequeño para modelarlo e garantir la especificación/requerimientos.

Los casos de test se seleccionan con este modelo, hay varios criterios para generarlos, algunos:

- **Cobertura de transiciones:** el conjunto debe asegurar que se ejecuten todas las transiciones

- **Cobertura de par de transiciones:** todo par de transiciones adyacentes que entran y salen de un estado.

- **Cobertura de árbol de transiciones:** todos los caminos simples, del estado inicial al final o a uno visitado.

## Testing de caja blanca

(estructural)

Se enfoca en el código, los casos de test se derivan de ahí. El objetivo es ejecutar las distintas estructuras del programa con el fin de descubrir errores.

Hay varios criterios para seleccionar los conjuntos de casos de test:

Criterio basado en el flujo de control

Se considera al programa como un grafo de flujo de control.

Nodos  $\Rightarrow$  bloques de código, i.e., conjuntos de sentencias que siempre se ejecutan juntas

(i,j) Aristas  $\Rightarrow$  posible transferencia de control del nodo i al j

Se sugiere la existencia de un nodo inicial y uno final. Un camino es una secuencia de nodos inicial al nodo final.

Criterio de cobertura de sentencia:

- Cada sentencia se ejecuta al menos una vez en el testing  $\Rightarrow$

El conjunto de caminos ejecutados incluye todos los nodos

Puede haber nodos inalcanzables.

## Criterio de cobertura de ramificaciones:

- El arista debe ejercitarse al menos una vez en el testing
- La decisión debe ejercitarse como verdadera y como falsa

Ramificaciones  $\Rightarrow$  Sentencial  
Cob. Cob.  
implican

Todo test suite se cubre ramificaciones cubre sentenciales, pero quedan haber algunos se abran ramificaciones que no encuentren el defecto mientras se hagan test suites se cubren sentenciales que si lo detecten.

## Criterio de cobertura de caminos:

- Todos los posibles caminos del estado inicial al final deben ejercitarse

Problema: la cant. de caminos puede ser infinita; puede haber caminos no realizables

Se usa más para evaluar el nivel de testing se ?/ seleccionar casos de uso.

## Criterio basado en flujo de datos

Se construye un grafo de definición-uso extrayendo el grafo de flujo de control.

Una sentencia en el grafo de flujo de control puede ser:

- en nodos  $\leftarrow$  def: definición de una variable (la var está a la izq. de la asignación)
- en aristas  $\leftarrow$  uso-c: la var se usa para cómputo (derecha de la asig.)
- $\leftarrow$  uso-q: la var se utiliza en un predicado de transfección de control.

Se asocian variables a nodos y aristas del grafo de flujo de control.

- def(i): conjunto de variables para el cual hay una definición en el nodo i
- c-use(i): conjunto de variables para el cual hay un uso-c en el nodo i
- q-use(i,j): conjunto de variables para el cual hay un q-use en la arista i,j

Un camino se dice libre de definiciones con respecto a una variable  $x$  si no hay definiciones de  $x$  en los nodos del camino intermedios.

### Criterios q/ los casos de test:

- todas las definiciones: por cada nodo  $i$  y cada  $x$  en  $\text{def}(i)$  hay un camino libre de definiciones con respecto a  $x$  hasta un uso -c o ? de  $x$ .
- todos los usos -? : se desejan todos los usos -? de todas las definiciones.
- todos los usos -c; algunos usos -?; algunos usos -c.

Una vez elegido el criterio, para determinar la cobertura se tendrá el test suite se deben usar herramientas que dicen q/ sentencias o ramificaciones y dejan sin cubrir.

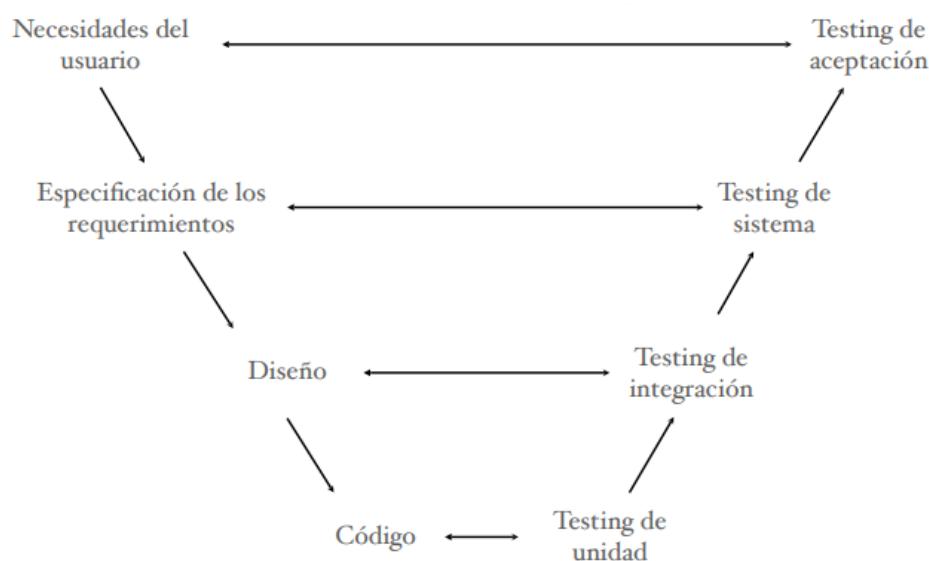
### Comparación y uso de caja negra y caja blanca

Deben usarse ambos, son técnicas complementarias.

- útiles a bajo nivel  
a alto nivel
- útiles a alto nivel
- Caja blanca => bueno para detectar errores en la lógica del programa, errores estructurales
  - Caja negra => bueno para detectar errores de entrada/salida, errores funcionales.

# Niveles de testing

Un solo tipo de testing sería incapaz de detectar los distintos tipos de defectos, ya que la naturaleza de estos es diferente para cada etapa del proyecto. Se utilizan distintos niveles de testing para revelar los distintos tipos de defectos.



Otra forma de testing es el test de regresión, se realiza cuando se introduce algún cambio al software. Verifica que funcionalidades previas continúen funcionando bien.

Si el test suite completo no puede realizarse cada vez que se introduce un cambio, se deben priorizar los casos de test necesarios.

# Registro de defectos y seguimiento

Como los SW grandes pueden tener miles de defectos encontrados por muchas personas distintas, su registro y corrección no puede realizarse informalmente. Esto generalmente se registran en un sistema **seguimiento de defectos** ("tracking") que permite rastrearlos hasta que se cierran.

Los defectos tienen su propio **ciclo de vida**.

Alguien lo encuentra y lo registra



Se asigna la tarea de corrección, esa persona hace el debugging y lo corrige

Durante este ciclo se registran info del defecto y ayudar al debugging y análisis. Se categorizan en distintos tipos. Una posible es "Orthogonal defect classification"

CatS: funcional, lógicas, standards, asignación, interfaz de usuario, de componente, diseño, documentación, etc.

También se registra la severidad del defecto en términos de su impacto en el SW.

### Possible categorización:

- Crítico: Puede demorar el proceso; afecta a muchos usuarios.
- Mayor: Mucho impacto pero que se solucionan provisionalmente; mucho esfuerzo corregirlo, pero tiene menor impacto en el cronograma.
- Menor: Defecto aislado que se manifiesta raramente y tiene poco impacto.
- Cosmético: Pequeños errores sin impacto en el funcionamiento correcto del sistema.

Idealmente todos los defectos deben cerrarse. Sin embargo, algunas veces, las orgs. entregan software c/ defectos. Hay estándares p/ determinar cuándo un producto se considera entregar.

# PLANEAMIENTO DEL PROYECTO DE SOFTWARE (cap. 5)

Es la primera fase del proceso de administración del proyecto.

Se realiza antes de comenzar el desarrollo, se planean todas las tareas de la administración del proyecto necesaria realizar.

Los tópicos más importantes son:

## 1. Planeamiento del proceso

Se planean cómo se estructurará el proyecto, esto incluye:

- determinar el modelo de proceso a seguir
- adecuado al proyecto
- definir las etapas y los criterios de entrada y salida de C/U
- actividades de verificación de c/ etapa
- definir mejores pasos (milestones)

se utilizan para analizar el progreso del proyecto

2.

## Estimación del esfuerzo

Es necesario/deseable saber cuánto costará en tiempo y dinero el desarrollo del SW.

El esfuerzo se mide, usualmente, en personalmes.

Esta estimación es clave para el planeamiento, de ello dependen tiempo costo y recursos (humanos) particularmente)

La estimación mejora mientras más info del proyecto tenemos.

## Construcción de modelos

Un modelo intenta determinar la estimación del esfuerzo a partir de valores de ciertos parámetros, que dependerán del proyecto, reduciendo el problema de estimar el esfuerzo a estimar/determinar ciertos parámetros claves del proyecto.

El parámetro/factor principal es el humano del proyecto.

Hay dos enfoques a la hora de estimar el esfuerzo

**top-down:** se determina el esfuerzo total y luego se calcula el esfuerzo de cada parte del proyecto

**bottom-up:** • Se identifican los módulos del sistema y se clasifican en simples, medios o complejos.

- Se determina el esfuerzo de codificación de cada módulo
- Se determina el esfuerzo total de codificación en base a los dos pasos anteriores
  - Se estima el esfuerzo de cada tarea y finalmente el esfuerzo total en base a proyectos similares
  - Se refinan los estimadores anteriores en base a factores específicos del proyecto

## Modelo CoCoRo

Es un modelo para la estimación del esfuerzo con enfoque top-down.  
Utiliza tamaño ajustado con algunos factores.

## Procedimiento:

A) Obtenir el estimador inicial usando el tamaño (en KLOC)

$$E_i = a * \text{tamaño}^b$$

a y b son constantes que dependen del tipo de proyecto. Escribir que debe ser

- orgánico: es relativamente simple y puede ser desarrollado por pequeños equipos. (mayor a 1 menor b)
- semi-rígido
- rígido: más ambicioso y novedoso, con restricciones estrictas impuestas por el entorno y altos requerimientos en cuanto a interfaces y confiabilidad. (menor a y mayor b)

B) Determinar un conjunto de 15 factores de multiplicación que regresen en distintos atributos

Estos atributos pueden ser de software, como la confiabilidad (RELY), de hardware, como la volatilidad de la máquina virtual (VIRT); de personal, como la calificación de los programadores (PCAP) y del proyecto, como el uso de herramientas de desarrollo de software (TOOL).

Cada atributo tiene una escala de calificación (muy bajo, bajo, normal, alto, muy alto, extra alto; no todos tienen todas las calificaciones) y para cada calificación se provee un factor multiplicativo.

### c) Ajustar el estimador de esfuerzo

Una vez que tenemos el factor multiplicativo de cada atributo, podemos ajustar el estimador inicial

$$\text{Estimación del esfuerzo final} = E_i \cdot \prod_{i=1}^{IS} f_i$$

Con  $f_i$ : el factor mult. de c/atributo

### D) Calcular el estimador de esfuerzo de cada fase

Dependiendo el tamaño en kloc del proyecto, el estimador de esfuerzo de cada fase será un porcentaje del estimador del esfuerzo final.

### 3. Planificación y recursos humanos

Dos niveles:

#### Planificación global

Abusar las metas parciales y la fecha final. Depende fuertemente del esfuerzo estimado.

Un método es estimar el tiempo programado del proyecto M (meses) en función del esfuerzo en P/M.

En efecto esto se calcula

$$M = 2,5 * E^{0,38}$$

Una rule of thumb es  $M = \sqrt{E}$

Luego debe determinarse la duración de cada meta parcial principal.

Para ello se usa la distribución de los RPHH, que sigue approx una curva de Rayleigh, y la distribución de esfuerzos.

#### Planificación detallada

Se deciden y asignan las tareas de bajo nivel (tareas realizadas por una persona en no más de 2/3 días) preservando la planificación de alto nivel.

Esta no se realiza de manera completa, si no se evoluciona.

Para ceder tarea se asigna nombre, esfuerzo, fecha, duración, recurso, etc. Estas son subdivisiones de las metas, su esfuerzo debe sumar preservando la duración total.

Para asignar estas tareas es necesario un equipo de trabajo estructurado, algunas estructuras son:

- Organización jerárquica: más común
  - Un administrador del proyecto con la responsabilidad global.
  - Este tiene programadores, testers y administradores de configuración que ejecutan las tareas.
- Equipos democráticos:
  - Funciona en pequeños grupos
  - El liderazgo es roativo
- Alternativa: desarrollo de grandes productos
  - Tres tareas principales: desarrollo, testing y administración del programa.
  - Desarrollo y testing se esperan independientes
  - Administradores proveen las especificaciones y aseguran que coding y testing estén coordinados.
  - Ceder tarea tiene su equipo y su líder.
  - Todos respondan a un líder general
  - Utilizado en grandes corporaciones.

4. Planeamiento de la administración de la configuración del software

5. Planeamiento del control de calidad

Su objetivo es especificar las actividades que se necesitan realizar para identificar y eliminar defectos. Se incluyen también herramientas y métodos que se usan a tal efecto.

Estos defectos se introducen en cualquier etapa, su eliminación es fundamental para alcanzar el objetivo de calidad (baja densidad de defectos). Esto se realiza fundamentalmente mediante <sup>esas</sup> actividades de control de calidad (QC) incluyendo revisiones y testing.

Hay ciertos enfoques para la administración del QC

- Enfoque ad-hoc: se hacen tests y revisiones de acuerdo a cuándo y como se necesite.

- Enfoque de procedimiento:

- el plan define qué tareas de QC se realizarán y cuándo
- principales tareas: revisión y testing

- provee procedimientos y lineamientos a seguir en ellas
- durante la ejecución del proyecto se asegura el seguimiento del plan y los procedimientos

- Enfoque cuantitativo:

No solo requiere que se ejecute el proc. sino que analizar los datos recolectados de los dificultades y establece juicios sobre la calidad (métricas, calidad de defectos)

- predice defectos usando info del pasado y compara la cantidad real contra las estimadas

El plan de calidad establece qué actividades deben realizarse. Tmb que se especifican los niveles esperados de defectos que cada tarea de QC debe encontrar.

Permite el seguimiento de la calidad del proyecto.

## 6. Administración de Riesgos

Un riesgo es cualquier condición o evento de ocurrencia incierta y no común que puede causar la falla del proyecto. El objetivo de la administración de riesgos es minimizar el impacto (en costo, calidad y tiempo) de la materialización de los riesgos.

Se realiza durante el planeamiento del proyecto

Evaluación de Riesgos

Identificación de riesgos

Análisis de riesgos

Administración del riesgo

Definición de prioridades de los riesgos.

Planeamiento de la administración de riesgos

Control de Riesgos

Resolución de riesgos

Se lleva a cabo durante la realización del proyecto

Seguimiento de riesgos

## Evaluación del riesgo

- Identificación del riesgo: Se identifican los posibles riesgos del proyecto mediante listas de control, experiencias pasadas, brainstorming, etc.

9 factores de riesgo más importantes:

- personal: insuficiente, inapropiadamente entrenado
- tiempo y costos irrealistas
- componentes externos: incontrolables o de baja calidad
- discrepancia en los requerimientos
- discrepancia con la interfaz del usuario
- arqu, diseño, calidad: inadecuada o insuficiente evaluación
- flujo continuo en los cambios de reqs.
- software legado
- tareas desarrolladas externamente: inadecuadas o demoradas

## ● Análisis de riesgos y definición de prioridades:

Como la cant. de riesgos que se tiene grande, deben priorizarse aquellos con mayor probabilidad de ocurrir y mayor impacto si ocurrieran.

→ Pueden ordenarse de acuerdo al valor de exposición al riesgo (RE):

$RE = \text{Prob. ocurrancia} * \text{impacto ocurrencia}$   
y realizar planes para tratar con aquellos de mayor RE

→ ó pueden clasificarse las probabilidades de ocurrencia en Bajas, Medianas o Altas, al igual que los impactos e identificar aquellos que son AA y AM/MA y enfocarse en ellos. (funciona bien si proyectos chicos y medianos)

## Control de riesgos

Si se debe evitarse un riesgo  $\Rightarrow$  evitarlo, Sino, planear y ejecutar los pasos para mitigarlo, i.e., definir las acciones a seguir en caso de materialización del riesgo, 1) q el su impacto sea mínimo.

El plan de mitigación de riesgos incluye los pasos a realizar a costo extra. Son distintos q los q se realizan si se materializa el riesgo.

7.

## Planificación del seguimiento del proyecto

El plan de seguimiento implica planificar qué medir, cuando y cómo y cómo analizar y reportar estos datos. Las principales medidas son:

- Tiempo: Més importante. ¿Se están cumpliendo los plazos?
- Efuerzo: ¿Se está dentro del presupuesto?
- Defectos: determinan calidad.
- Tamaño: mucha info se normaliza respecto al tamaño.

En resumen ⇒ el objetivo del seguimiento es hacer visible la ejecución del proyecto de manera de realizar acciones correctivas cuando sea necesario con el fin de asegurar el éxito del proyecto.

### Niveles de seguimientos:

#### • nivel de actividad:

→ asegura que se realizan las actividades en tiempo y forma

→ se realiza diariamente por los admin. del proyecto

#### • reporte de estado:

→ se genera semanalmente

→ contiene resumen de los act.

complejidades y pendientes desde el último reporte y cuestiones que necesiten aclaración o ser resueltas.

- Análisis de metas parciales:

- Se realiza una mayor revisión con orden meta parcial
- Se analizan los esfuerzos y tiempos reales vs. los estimados. Si la desviación es grande => medidas correctivas.
- Se revisan los riesgos.