



# Principios fundamentales



# Principios que guían el proceso

- Ser ágil
- En cada etapa centrarse en la calidad
- Estar listo para adaptar
- Formar un equipo eficaz
- Establecer mecanismos para la comunicación y coordinación
- Administrar el cambio
- Evaluar el riesgo
- Crear productos del trabajo que agreguen valor para otros



# Principios que guían la práctica

- Divide y vencerás
- Entender el uso de la abstracción
- Buscar la coherencia
- Centrarse en la transferencia de información
- Construir software que tenga modularidad eficaz
- Buscar patrones
- Cuando sea posible representar el problema y su solución desde varias perspectivas diferentes
- Tener en mente que alguien dará mantenimiento al software



# Principios que guían toda actividad estructural

# Principios de comunicación

- Escuchar
- Antes de comunicarse, prepararse
- Alguien debe facilitar la actividad
- Es mejor la comunicación cara a cara
- Tomar notas y documentar las decisiones
- Perseguir la colaboración
- Permanecer centrado; hacer módulos con la discusión
- Si algo no está claro, hacer un dibujo
- Una vez que se acuerde algo, avanzar. Si no es posible  
Lograr el acuerdo en algo, avanzar. Si una característica o  
función no está clara o no puede aclararse en el momento,  
avanzar.
- La negociación no es un concurso o un juego. Funciona  
mejor cuando ambas partes ganan.



# Principios de planeación

- Entender el alcance del proyecto
- Involucrar en la actividad de planeación a los participantes del software
- Reconocer que la planeación es iterativa
- Estimar con base en lo que se sabe
- Al definir el plan, tomar en cuenta los riesgos
- Ser realista
- Ajustar la granularidad cuando se defina el plan
- Definir cómo se trata de asegurar la calidad
- Describir cómo se busca manejar el cambio
- Dar seguimiento al plan con frecuencia y hacer los ajustes que se requieran

# Principios de modelado

- El equipo de software tiene como objetivo principal elaborar software y no crear modelos.
- Viajar ligero, no crear más modelos de los necesarios
- Tratar de producir el modelo más sencillo que describa al problema o al software
- Construir modelos susceptibles al cambio
- Ser capaz de enunciar un propósito explícito para cada modelo que se cree
- Adaptar los modelos que se desarrollan al sistema en cuestión
- Tratar de construir modelos útiles, pero olvidarse de elaborar modelos perfectos
- No ser dogmático respecto a la sintaxis del modelo. Si se tiene éxito para comunicar contenido, la representación es secundaria
- Si su instinto dice que un modelo no es el correcto a pesar de que se vea bien en el papel, hay razones para estar preocupado.
- Obtener retroalimentación tan pronto como sea posible



# Requerimientos de los principios del modelado

- Debe representarse y entenderse el dominio de información de un problema
- Deben definirse las funciones que realizará el software
- Debe representarse el comportamiento del software (como consecuencias de eventos externos)
- Los modelos que representen información, función y comportamiento deben dividirse de manera que revelen los detalles en forma estratificada (o jerárquica)
- El trabajo de análisis debe avanzar de la información esencial hacia la implementación en detalle.





# Principios de modelado de diseno

- El diseño debe poder rastrearse hasta el modelo de requerimientos
- Siempre tomar en cuenta la arquitectura del sistema que se va a construir
- El diseño de los datos es tan importante como el de las funciones de procesamiento
- Las interfaces tanto internas como externas deben diseñarse con cuidado
- El diseño de la interfaz de usuario debe ajustarse a las necesidades del usuario final. Sin embargo, en todo caso debe resaltar la facilidad de uso.
- El diseño en el nivel de componentes debe tener independencia funcional
- Los componentes deben estar acoplados con holgura entre si y con el ambiente externo
- Las representaciones del diseño (modelos) deben entenderse con facilidad
- El diseño debe desarrollarse en forma iterativa. El diseñador debe buscar mas sencillez en cada iteración.

# Principios de la construcción

## ● Principios de codificación

- Principios de preparación
  - Entender el problema que se trata de resolver
  - Comprender los principios y conceptos básicos del diseño
  - Elegir un lenguaje de programación que satisfaga las necesidades del software que se va a elaborar y el ambiente en el que operará
  - Seleccionar un ambiente de programación que disponga de herramientas que hagan más fácil su trabajo
  - Crear un conjunto de pruebas unitarias que se aplicarán una vez que se haya terminado el componente a codificar



## ● Principios de programación:

- Restringir sus algoritmos por medio del uso de programación estructurada [Boh00].
- Tomar en consideración el uso de programación por parejas.
- Seleccionar estructuras de datos que satisfagan las necesidades del diseño.
- Entender la arquitectura del software y crear interfaces que son congruentes con ella.
- Mantener la lógica condicional tan sencilla como sea posible.
- Crear lazos anidados en forma tal que se puedan probar con facilidad.
- Seleccionar nombres significativos para las variables y seguir otros estándares locales de codificación.
- Escribir código que se documente a sí mismo.
- Crear una imagen visual (por ejemplo, líneas con sangría y en blanco) que ayude a entender.




## ● Principios de validación:

- Realizar el recorrido del código cuando sea apropiado.
- Llevar a cabo pruebas unitarias y corregir los errores que se detecten.
- Rediseñar el código.

## ● Principios de la prueba

- La prueba es el proceso que ejecuta un programa con objeto de encontrar un error.
- Un buen caso de prueba es el que tiene alta probabilidad de encontrar un error que no se ha detectado hasta el momento.
- Una prueba exitosa es la que descubre un error no detectado hasta el momento.

- 
- Todas las pruebas deben poder rastrearse hasta los requerimientos del cliente
  - Las pruebas deben planearse mucho antes de que den comienzo
  - El principio de Pareto se aplica a las pruebas de software
  - Las pruebas deben comenzar “en lo pequeño” y avanzar hacia “lo grande”
  - No son posibles las pruebas exhaustivas



# Principios de despliegue

- Deben manejarse las expectativas de los clientes
- Debe ensamblarse y probarse el paquete completo que se entregará
- Antes de entregar el software, debe establecerse un régimen de apoyo
- Se deben proporcionar a los usuarios finales materiales de aprendizaje apropiados
- El software defectuoso debe corregirse primero y después entregarse

# Preguntas a resolver

- Cada vez que la búsqueda de la calidad reclama recursos y tiempo, ¿es posible ser ágil y centrarse en ella?
- De los ocho principios fundamentales que guían el proceso ¿cuál cree que sea el más importante?
- Un principio de comunicación importante establece que hay que “prepararse antes de comunicarse”. ¿Cómo debe manifestarse esta preparación en los primeros trabajos que se hacen? ¿Qué productos del trabajo son resultado de la preparación temprana?
- Haga algunas investigaciones acerca de cómo “facilitar” la actividad de comunicación (use las referencias que se dan u otras distintas) y prepare algunos lineamientos que se centren en la facilitación.
- ¿En qué difiere la comunicación ágil de la comunicación tradicional de la ingeniería de software? ¿En qué se parecen?
- ¿Por qué es necesario “avanzar”?
- Investigue sobre la “negociación” para la actividad de comunicación y prepare algunos lineamientos que se centren sólo en ella.
- Describa lo que significa granularidad en el contexto de la programación de actividades de un proyecto.
- ¿Por qué son importantes los modelos en el trabajo de ingeniería de software? ¿Siempre son necesarios? ¿Hay calificadores para la respuesta que se dio sobre esta necesidad?
- ¿Cuáles son los tres “dominios” considerados durante el modelado de requerimientos?
- Trate de agregar un principio adicional a los que se mencionan para la codificación.
- ¿Qué es una prueba exitosa?
- Diga si está de acuerdo o en desacuerdo con el enunciado siguiente: “Como entregamos incrementos múltiples al cliente, no debíamos preocuparnos por la calidad en los primeros incrementos; en las iteraciones posteriores podemos corregir los problemas. Explique su respuesta.
- ¿Por qué es importante la retroalimentación para el equipo de software?