Introducción

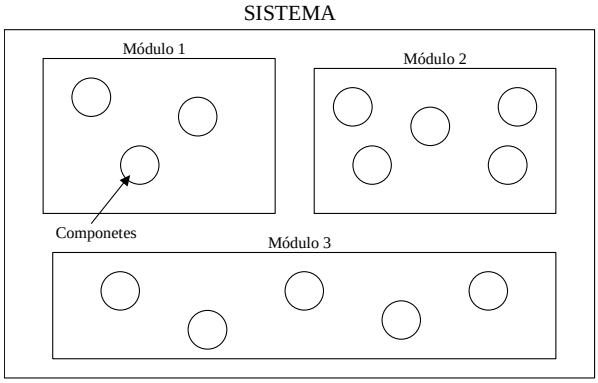
- En el desarrollo de STR, hay que tener en cuenta todas las fases del ciclo de vida del software: análisis, especificaciones, diseño, codificación, verificación, instalación y mantenimiento
- El desarrollo en distintos pasos tiende a reducir la duración de las especificaciones y el diseño.
- Son importantes las primeras fases, pero no se deben alargar demasiado. Conviene empezar la codificación antes de concluir el diseño.
- Es interesante crear prototipos lo antes que se pueda

Especificaciones

- Deben ser completas y consistentes. Se extienden a las fases siguientes
- Es importante la notación. Niveles de notación:
  - Informal: lenguaje natural y diagramas imprecisos
  - Estructurado: representación gráfica y diagramas bien definidos. Manipulación automática
  - Formal: lenguaje matemático. Estudio de propiedades. Difícil de entender
- Máxima seguridad
- Herramientas de diseño automáticas

Diseño (I)

- Necesario en problemas complejos. Varias personas o grupos
- Descomposición en dos aspectos:
  - Diseño estructural
  - Diseño detallado



Diseño (II)

- Interesa obtener un grado elevado de detalle con subsistemas lo más independientes posible
- Técnicas complementarias
  - Descomposición
  - Abstracción

- En una buena encapsulación los módulos forman unidades independientes y aisladas
- Criterios para obtener una buena encapsulación
  - Cohesión: Conexión interna de los componentes
  - Acoplamiento: Interdependencia de los módulos
- Hay que conseguir una cohesión fuerte y un grado de acoplamiento bajo

Metodología HRT-HOOD

- Tiene en cuenta las necesidades de los STR
- Proceso de diseño: Progresión creciente de compromisos específicos
  - Los compromisos definen las propiedades que un módulo debe cumplir
  - Las obligaciones son los requisitos que se exigen a los componentes (especificaciones en el nivel superior)
- El proceso de refinamiento consiste en ir transformando las obligaciones en compromisos

HRT-HOOD. Actividades

- Diseño de la arquitectura lógica
  - Compromisos independientes de las restricciones del entorno de ejecución
- Diseño de la arquitectura física
  - Tiene en cuenta los requerimientos funcionales y las restricciones del entorno
  - Requerimientos no funcionales
  - Tiene en cuenta las restricciones temporales

Codificación

- Clases de lenguajes de programación
  - Lenguajes ensamblador
  - Lenguajes secuenciales
  - Lenguajes concurrentes
- Crisis del software (síntomas)
  - Falta de respuesta
  - Poca fiabilidad
  - Coste imprevisible
  - Mantenimiento complejo
  - Retraso en los plazos
  - Poco transportable
  - Eficiencia

Criterios en el diseño de lenguajes

- Seguridad
- Legibilidad
- Flexibilidad
- Simplicidad
- Portabilidad
- Eficiencia

Ada

Verificación

- En los sistemas tradicionales
  - Verificación por el usuario
  - Versiones beta
  - Compromiso entre coste de desarrollos y fallos en el producto
- Localización de errores
  - No solo al final. Poder verificar cada módulo
  - Dificultad introducida por la concurrencia
  - Problemas en la reproducción de los fallos (entorno)
- Simuladores
  - Necesario si no puede probar en el entorno real
  - Imita las acciones del sistema. Reproduce situaciones
  - Permitir probar todo o parte
  - En el propio sistema o en un sistema independiente de TR
- Herramientas
  - Construcción de herramientas especiales

Prototipos

- Validación temprana de las especificaciones
  - Correctas
  - Completas
- Validación de las expectativas del cliente
  - Experimentar situaciones de forma mas real
- Coste de los prototipos
  - Muy inferior al del producto final
  - Prescindir de las restricciones temporales
  - Reaprovechar parte del diseño