Diseño de Software basado en Arquitecturas

Atributos de Calidad - Tácticas

Juan Carlos Ramos
Natalia Depetris
UTN - FRSF

Atributos de Calidad – Tácticas

- Los requerimientos de calidad (escenarios) especifican las *respuestas* del software a determinados *estímulos*.
- El arquitecto debe aplicar ciertas *tácticas* para poder diseñar una arquitectura de software que permita brindar estas respuestas.

Atributos de Calidad – Tácticas

Estímulo

Artefacto

Respuesta

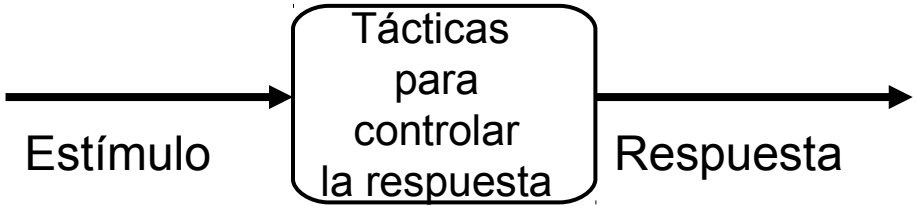
Entorno

Fuente del Estímulo

Medida de la Respuesta

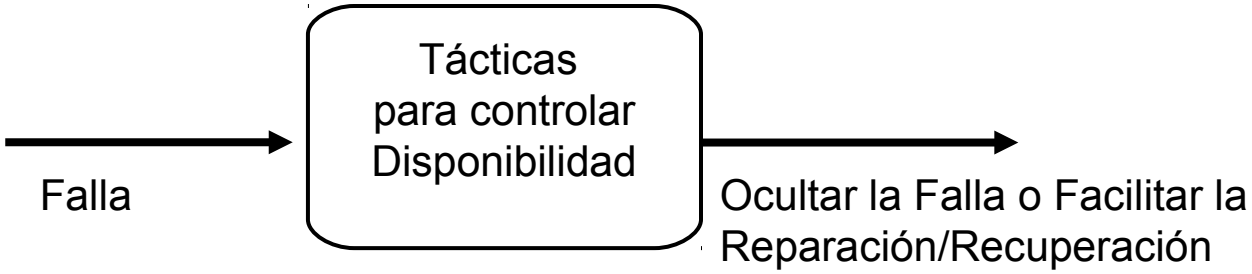
Atributos de Calidad – Tácticas

- *Táctica*
  - Decisión de diseño que afecta el control de una respuesta a un atributo de calidad.
  - *Mecanismo arquitectónico*: sinónimo.
- Estrategia Arquitectónica
  - Conjunto de tácticas.



Tácticas para *Disponibilidad*

- *Disponibilidad (Availability)*: la medida del tiempo en que el sistema está operativo y ejecutando correctamente.
  - *Una falla del sistema ocurre cuando este no entrega más un servicio consistente con sus especificaciones. Esta falla es observable por los usuarios del sistema (humano u otros sistemas).*



Tácticas para *Disponibilidad (2)*

- Detección de Fallas: Monitoreo de actividad y reporte de fallas
  - *Ping/eco*: una componente envía un 'ping' y espera recibir un 'eco' de respuesta de la componente escrutada. Múltiples niveles detectores: componentes hasta procesos.
  - *Heartbeat (latido)*: se emite un mensaje 'heartbeat' periódicamente a otra componente que tiene que estar escuchando, si esto falla se asume que la componente origen falló, se notifica a una componente de corrección de errores.
  - Excepciones: disparadas cuando ocurre un error/falla.
  - Implica restricciones en los patrones de interacción entre las componentes

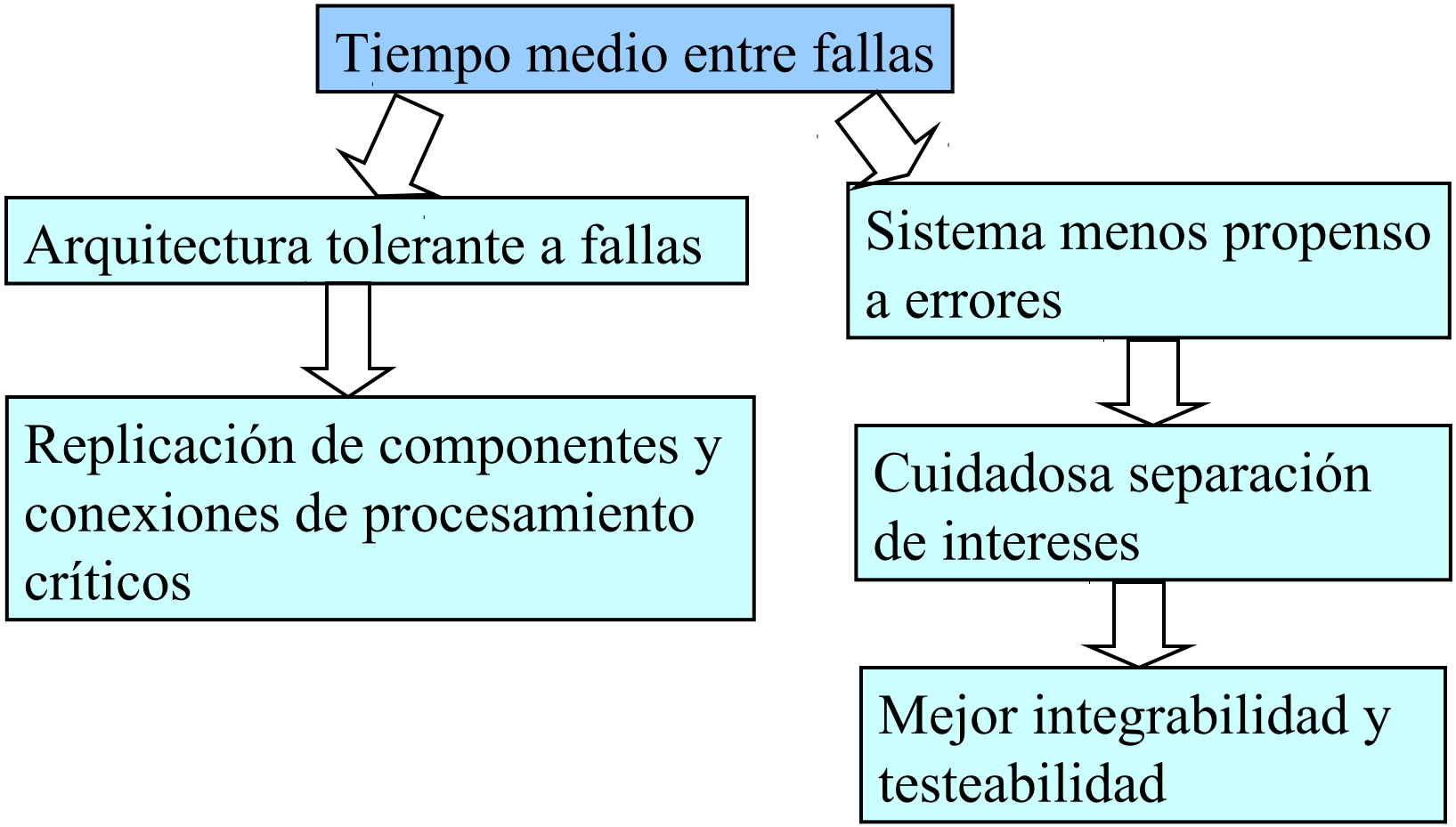
Tácticas para *Disponibilidad (3)*

- Recupero de Fallas
  - Redundancia de componentes críticas activas
    - Todos los componentes responden a los eventos. Solo es considerada una respuesta.
  - Redundancia de componentes críticas pasivas
    - El componente primario responde, actualiza los secundarios.
    - Sincronización: responsabilidad de la componente primaria
  - Plataforma de cómputos standby de repuesto
  - Estrategias de recupero de errores en todo el sistema
  - Prevención para manejar transportes inestables
  - Redundancia de caminos de comunicación críticos
  - Capacidad de intercambio ('switching') en actividad ('live switching', 'hot-swap')
  - Capacidad de recupero o inicio rápido.

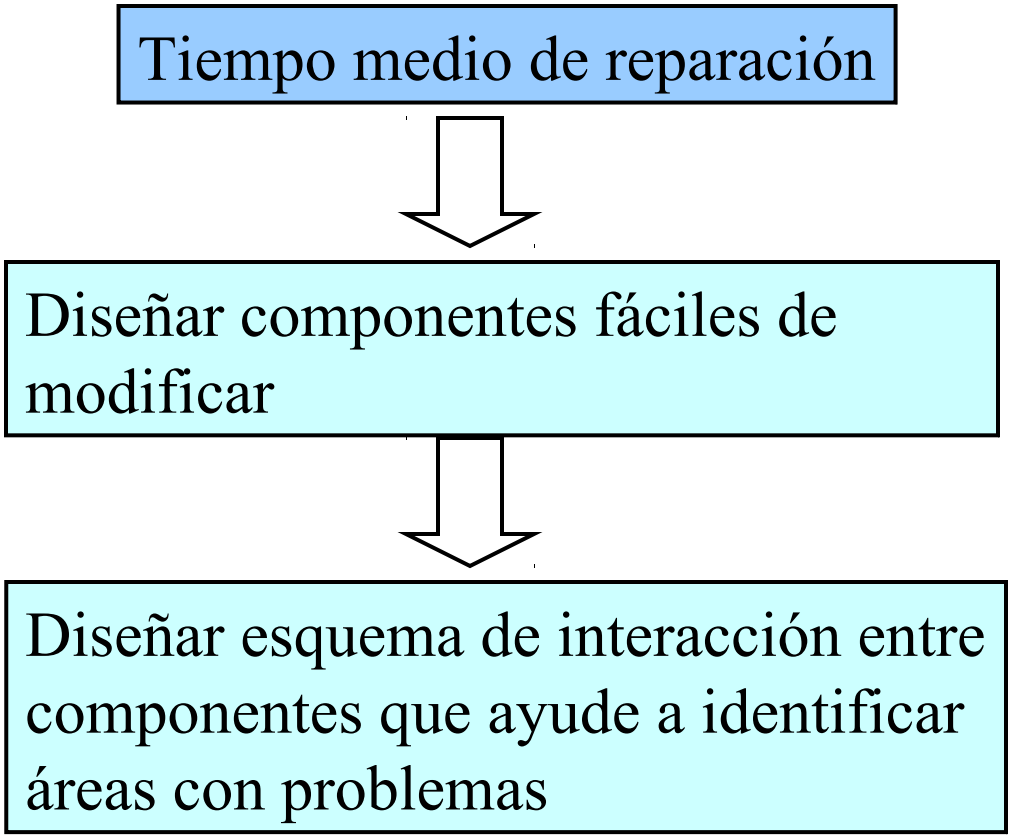
Tácticas para *Disponibilidad (4)*

- Prevención de Fallas
  - Remoción desde el servicio
    - Ejemplo: reiniciar las componentes periódicamente.
  - Transacciones
    - Bloques de varios pasos secuenciales. Se deshacen en bloque. Control de integridad de datos.
  - Monitor de Procesos
    - Detecta fallas en los procesos, y reinicia nuevas instancias.

Tácticas para *Disponibilidad (5)*

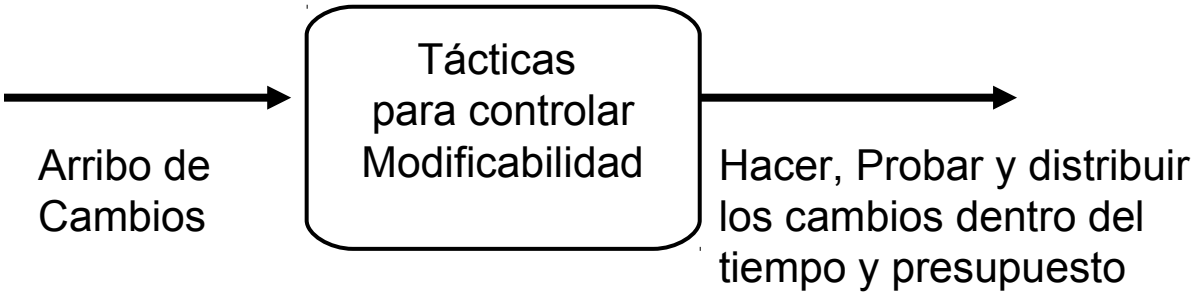


Tácticas para *Disponibilidad (y 6)*



Tácticas para *Modificabilidad*

- *Modificabilidad*: la facilidad con la cual el software puede adecuarse a los cambios.



Tácticas para *Modificabilidad (y 2)*

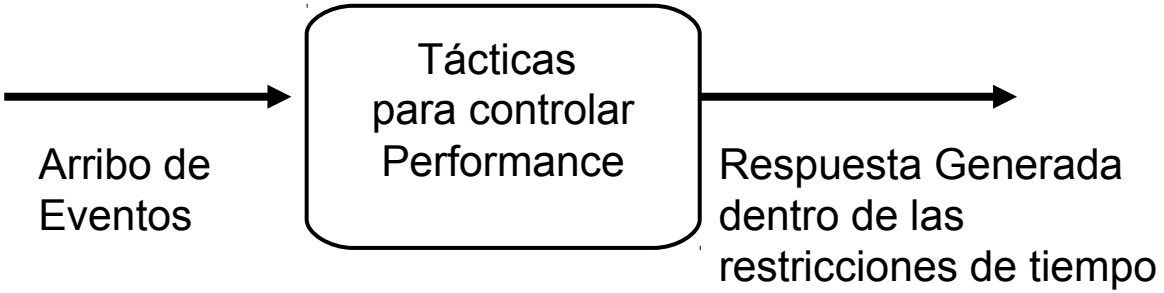
- Descomposición funcional y asignación de funcionalidades consistente (Coherencia Semántica - Cohesión)
- Empaquetado ('packaging') consistente de funcionalidad (Coherencia Semántica - Cohesión)
- Abstraer servicios comunes en un módulo especializado.
- Uso de un pequeño número de patrones de diseño.
- Capas
- Ocultamiento de información (información pública y privada → Interfaces públicas y privadas)
- Mantener las interfaces existentes de una componente (nombre y signatura)
- Desacoplar los productores de los consumidores de datos o servicios (Acoplamiento)
- Separar definición de interfaces de su implementación.

Tácticas para *Performance*

- *Performance*: se refiere a las respuestas del sistema, ya sea el *tiempo requerido* para responder a *eventos específicos* o la *cantidad de eventos* procesados en un intervalo de tiempo dado [Smith, 1993]
- Normalmente expresado por la *cantidad de transacciones por unidad de tiempo* o por el *tiempo que toma completar una transacción* con el sistema.
- *Comunicación* toma más tiempo que los cálculos ⇨ performance es una función de cuanta *comunicación* e *interacción* hay entre las componentes (problema arquitectónico).

Tácticas para *Performance (2)*

- *Tácticas*:



Tácticas para *Performance (3)*

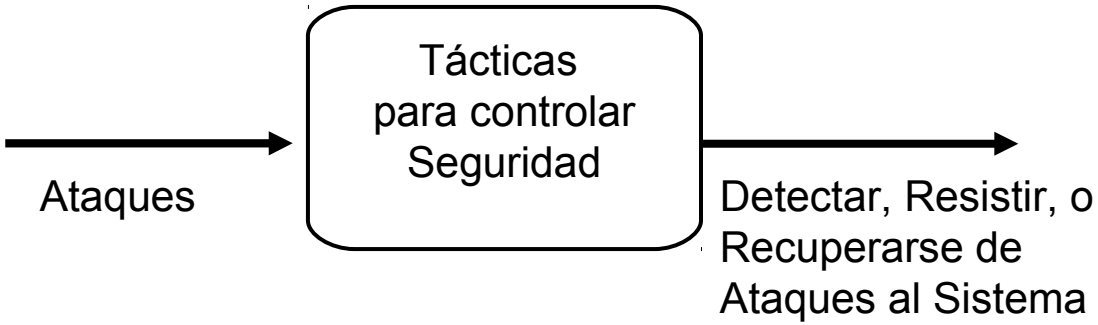
- Incrementar eficiencia computacional
  - Mejorar algoritmo
  - Aumentar procesador/discos disponibles
- Reducir el overhead computacional
  - Reducir los requerimientos de comunicación. Menos 'intermediarios'
- Particionamiento de la funcionalidad.

Tácticas para *Performance (y 4)*

- Introducir Concurrencia
- Mantener múltiples copias de datos o cálculos.
- Incrementar los recursos disponibles
- Flexibilidad de asignación de hardware
- Mecanismos de monitoreo (internas al producto) y herramientas (externas al producto)
- Programación de tiempo real y mecanismos de monitoreo
- Paquetes (de SW) con facilidades para modelar performance.
- Administración de recursos.

Tácticas para *Seguridad*

- *Seguridad*: habilidad del sistema para resistir intentos no autorizados a usar servicios del sistema, mientras permanece proveyendo servicios a usuarios legítimos.



Tácticas para *Seguridad (1)*

- Resistir ataques
  - Usuarios autenticados
  - Usuarios autorizados
  - Mantener confidencialidad de los datos
    - Encriptación y desencriptación
    - VPN (Virtual Private Network) o SSL (Secure Sockets Layer)
  - Mantener integridad
    - Información redundante, checksums o hash
  - Limitar exposición
    - Minimización de los puntos de entrada
  - Limitar acceso
    - Firewalls, DMZ (Demilitarized Zone)

Tácticas para *Seguridad (2)*

- Detección de ataques
  - Sistemas de detección de intrusos
    - Comparar patrones de tráfico de red con una BD
    - Si hay diferencias se compara con patrones históricos de ataques conocidos.
  - Sistemas de monitoreo de los sensores y administración de las acciones a tomar ante la detección de un intruso.

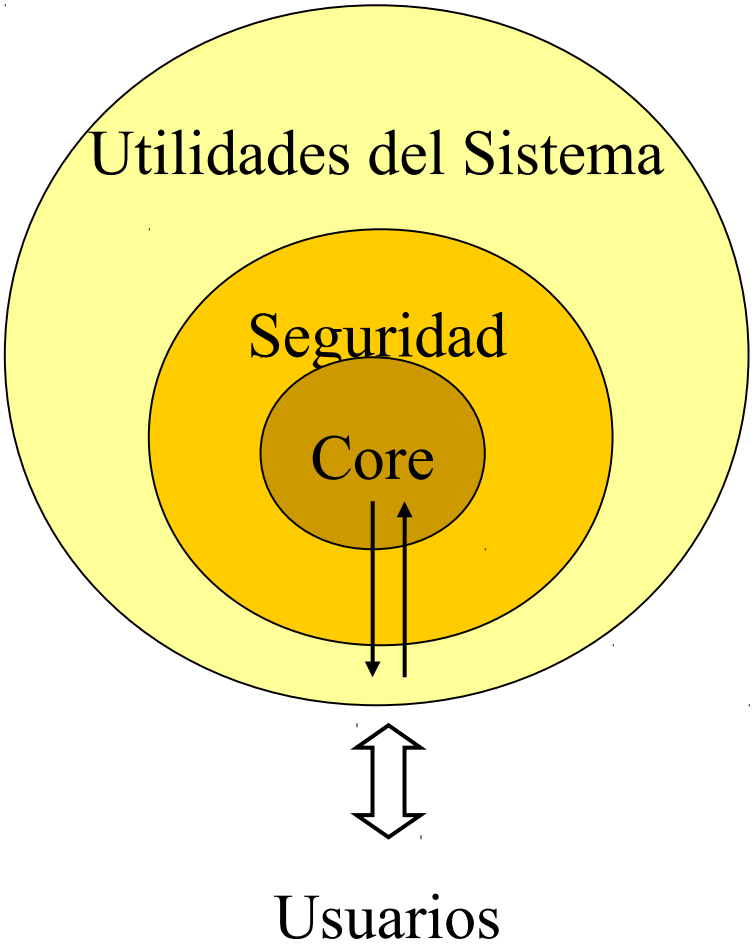
Tácticas para *Seguridad (3)*

- Recupero ante ataques
  - Restaurar el sistema o dato a un estado correcto.
    - Mantener copias redundantes de datos administrativos: claves, listas de control de accesos, servicios de nombres de dominios, y datos de perfiles de usuarios.
  - Identificar un atacante
    - Mantener un registro de auditoría.
      - Una copia de cada transacción aplicada a los datos junto con información de identificación.

Tácticas para *Seguridad (4)*

- *Tácticas* :
  - Mecanismos de monitoreo de actividad para detección de intrusos
    - Monitores de red para inspección y registro de eventos de red.
  - ‘Servidor de autenticación’ ubicado entre los usuarios externos y la parte del sistema que provee los servicios.
    - Autenticación y Autorización.
  - El sistema puede ser ubicado detrás de un “firewall” de comunicaciones a través del cual toda la comunicación desde y hacia el sistema son manejadas por un proxy.
  - Firewall de aplicación
  - Mantener registros de auditoría para recupero.
  - Kernels (núcleos) y shells (capas) de seguridad
    - El sistema puede ser construido sobre un kernel seguro que provea servicios de seguridad.

Tácticas para *Seguridad (4)*

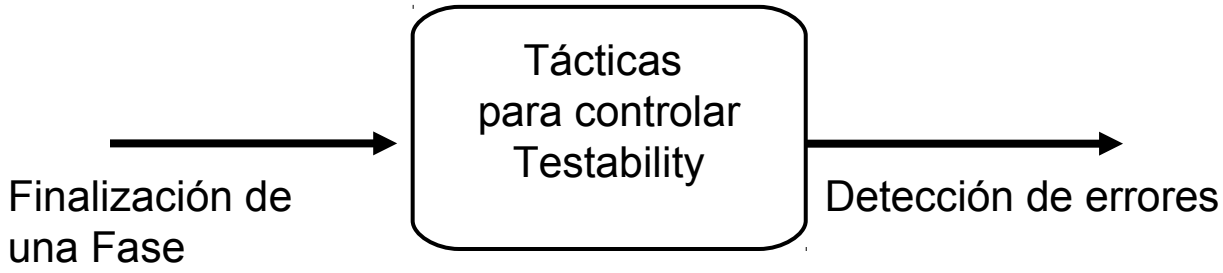


Tácticas para *Seguridad (y 5)*

- Para aplicar las tácticas es necesario:
  - Identificar componentes *especiales* (las que deben ser protegidas),
  - Separarlas del resto de las funcionalidades del sistema,
  - Y establecer la coordinación e interacción de las otras componentes a través de estas.

Tácticas para *Testability*

- *Testability*: la facilidad con la cual se puede demostrar que el software falla.
- Conceptos relacionados:
  - Capacidad de ser controlado (controlability)
  - Capacidad de ser observado (observability).

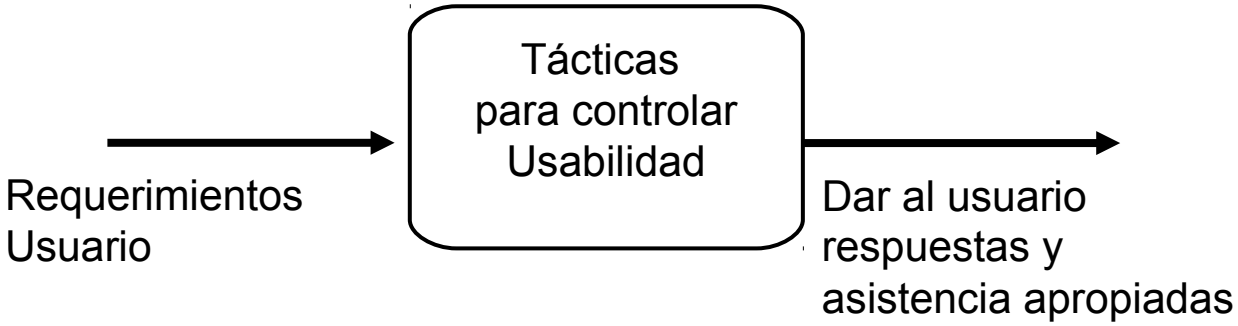


Tácticas para *Testability (y 2)*

- *Tácticas* :
  - Mecanismos para monitoreo interno, captura, registro y reporte
  - Entornos de ejecución especializados (ej.: para sistemas integrados)
  - Herramientas de simulación
  - Herramientas de testeo
  - Recursos para monitoreo y debugging
  - Esquema de manejo de error consistente
  - Nivel de documentación arquitectónica
  - Uso de ocultamiento de información
  - Desarrollo incremental
  - Separar interfaces de implementación

Tácticas para *Usabilidad*

- *Usabilidad*: la facilidad de uso y de entrenamiento de los usuarios finales del sistema.
- Aspectos de la usabilidad:
  - *Aprendizaje*.
  - *Eficiencia*
  - *Memorización*
  - *Evitar errores*



Tácticas para *Usabilidad (2)*

- *Tácticas* en tiempo de ejecución:
  - Deshacer (Undo)
  - Interrupción de la acción actual (Cancelar)
  - Mostrar múltiples vistas
  - Mantener modelos de las tareas (Ej.: las palabras comienzan con mayúsculas → Corrector de Sintaxis)
  - Mantener modelos de usuario (Ej.: Scrolling más lento)
  - Mantener modelos del sistema (Ej.: poder predecir el tiempo necesario para una actividad)

Tácticas para *Usabilidad (y 3)*

- *Tácticas* de diseño:
  - Separar la interface de usuario del resto de la aplicación.
    - Model-View-Controller (MVC)
    - Presentation-Abstraction-Control (PAC)
  - Interfaces de usuarios (UI) con un 'look and feel' común, y 'toolkits' de UI comunes.
  - Guías de estilo y 'frameworks' de soporte comunes.

Tácticas para *Interoperabilidad*

- *Interoperabilidad*: la habilidad de dos o más sistemas de cooperar en tiempo de ejecución.
- *Tácticas* :
  - Minimizar la *complejidad externa* de las componentes (interfaces, pre y post condiciones, etc.)
  - Conjunto limitado de mecanismos de interacción y protocolos
  - Existencia de esquema universal de asignación de nombres
  - Establecer nombres de servicios
  - Especial atención a las interfaces de las componentes

Tácticas para *Portabilidad*

- *Portabilidad*: la habilidad del sistema para ejecutar en diferentes entornos de computación (hardware, software, o ambos).
- Un sistema es portable cuando logra colocar todas las relaciones con un entorno operativo particular en una componente o un conjunto pequeño de componentes fácilmente intercambiables.

Tácticas para *Portabilidad (y 2)*

- *Tácticas* :
  - Capas independientes de la plataforma/redes
    - *Capa de portabilidad:* conjunto de servicios de SW que le brindan a la aplicación una interface abstracta de su entorno.
    - La capa de portabilidad surge de aplicar fuertemente los principios de *ocultamiento de información*.
    - Máquinas virtuales (Java)
    - Runtimes: Cobol, Progress, etc.
    - Compiladores C
  - Uso de interfaces estándares (entre módulos)

Tácticas para *Reusabilidad*

- *Reusabilidad*: el grado en el cual componentes existentes (o el sistema completo) pueden ser reusadas en una nueva aplicación.
- Si un sistema es estructurado para que sus componentes puedan ser elegidas de componentes ya construidas ⇨ *Integrabilidad*
- Depende del grado de *acoplamiento* entre las componentes.
  - Menor *acoplamiento* ⇨ Mayores chances de éxito

Tácticas para *Reusabilidad (y 2)*

- *Tácticas* :
  - Minimizar acoplamiento
  - Regularidad y minimización de patrones
  - Crear un framework de la aplicación
  - Crear una arquitectura de línea de producto

Tácticas para *Integrabilidad*

- *Integrabilidad*: la habilidad de hacer el desarrollo de componentes en forma separada y que el sistema trabaje correctamente al juntarlas.
- Depende:
  - Complejidad externa de las componentes
  - Mecanismos de interacción y protocolos
  - Grado en el cual las responsabilidades ha sido claramente particionadas.
  - Qué tan bien y completamente han sido especificadas las interfaces

Tácticas para *Integrabilidad* (y 2)

- Caso especial: *Interoperabilidad*
  - capacidad de un subconjunto de partes del sistema para trabajar con otro sistema.
- *Tácticas* :
  - Minimización de interface
  - Conjunto limitado de mecanismos/protocolos de interacción
  - Acoplamiento mínimo

Bibliografía

- "Software Architecture in Practice" Second Edition, Len Bass, Paul Clement, Rick Kazman, Addison-Wesley, 2003
  - Chapter 5 – Archieving Qualities
- "Quality Attributes", Mario Barbacci, Mark H. Klein, Thomas A. Longstaff, Charles B. Weinstock, Technical Report, CMU-SEI, 1995