Tácticas

Son decisiones comunes de arquitectura orientadas a lograr o satisfacer determinado atributo de calidad.

Para resolver los requerimientos no funcionales se utilizan estilos, a los cuales se suman mecanismos o tácticas.

Una táctica se puede refinar en varias tácticas.

# Tácticas de disponibilidad

Objetivo: Que un error no se transforme en una falla. Que el error se arregle o se enmascare.

Tácticas

## Detección de errores

### Ping echo

Hacer ping y esperar que el componente conteste. Si no hay respuesta, se asume que el componente falló.

### Latencia

El componente envía cada determinado tiempo una señal de que está operativo. Si no se recibe la señal durante mucho tiempo se asume que falló.

### Excepciones

## Recuperación de errores – preparación y arreglo

### Votación

Varios procesos corriendo en procesadores redundantes y todos toman la misma entrada, pero con algoritmos diferentes, desarrollado por personas diferentes. Todos contestan y se usan algoritmos para ver qué resultado tomar.

### Redundancia activa

Todos reciben la entrada, la procesan y envían la respuesta. Se toma la respuesta del que conteste más rápido. De esta forma todos los componentes se mantienen sincronizados.

### Redundancia pasiva

El componente primario recibe la entrada, la procesa y envía la respuesta. Luego le avisa a los componentes redundantes para que se sincronicen. Si el componente primario falla, se analiza el grado de sincronización de los redundantes.

### Repuesto

Se mantiene un componente de repuesto para sustituir al que falle. Al ocurrir un fallo, el componente debe ser sincronizado con la última versión consistente del componente primario con sus configuraciones y estados.

## Recuperación de errores – reintroducción de componentes

### Modo sombra

El componente que falla se mantiene funcionando en “modo sombra” mientras se recupera. Imita su comportamiento normal durante un breve período.

### Resincronización de estados

Mecanismos para sincronizar los componentes redundantes con el componente primario.

### Checkpoint/rollback

Se crean checkpoints a lo largo de la ejecución para poder volver a un estado consistente (rollback) en caso de que el componente falle.

## Prevención de errores

### Remoción del servicio

Se retira el componente antes de que falle. Si el cambio es automático, la arquitectura debe soportarlo; si el cambio es manual, el sistema debe soportarlo.

### Transacciones

Se trata de empaquetar una serie de pasos secuenciales para deshacer todo el conjunto de una vez, en caso de falla.

### Monitoreo de procesos

Cuando se detecta un error, el monitor mata el proceso defectuoso y lo reinicializa en un estado que pueda mantener el sistema funcional

# Tácticas de modificabilidad

Objetivo: hacer cambios, testearlos y generarlos sin costos de tiempo y presupuesto

## Localización de cambios

### Coherencia semántica

Se trata de mantener las relaciones dentro del módulo, minimizando las dependencias hacia otros módulos para tareas similares o con el mismo significado semántico.

#### Abstraer servicios comunes

### Anticipar cambios

La diferencia entre esta táctica y la anterior es que la anterior asume que los cambios serán semánticamente coherentes. Esta táctica se centra específicamente en los cambios esperados y la minimización de los módulos a tocar, muchas veces armando módulos en torno a la funcionalidad y no a la semántica.

### Generalizar módulos

Hacer que el componente reaccione a las entradas de manera similar, para que los cambios se reflejen solo en la modificación de las entradas y no en el módulo en general.

### Limitar opciones

Se trata de limitar los cambios posibles. Por ejemplo, limitando los cambios de plataforma a una línea de plataformas en particular.

## Prevenir efecto dominó

Limitar las modificaciones a los módulos que no dependen directamente del cambio. Se quiere modificar el módulo A, que tiene relaciones con el B, hay que considerar el tipo de relación:

* **Sintaxis de datos/servicios:** el módulo B consume datos/servicios producidos por A, que tienen una sintaxis determinada.
* **Semántica de datos/servicios:** el módulo B consume datos/servicios producidos por A, que tienen una semántica determinada.
* **Secuencia**
  + **De datos:** El módulo B consume los datos producidos por A en determinada secuencia
  + **De control:** el módulo B ejecuta luego de A, que debe haber ejecutado con restricciones de tiempo.

### Ocultamiento de información

Se centra en ocultar la información a los demás módulos, logrando menos acoplamiento.

### Mantener interfaces existentes

Diseñar interfaces que perduren independientemente de los cambios que se puedan llegar a realizar.

### Restringir caminos de comunicación

Se centra en restringir los módulos que comparten información (reducir módulos que producen y consumen información)

### Utilizar intermediarios

Asignar intermediarios que centralicen la interacción entre 2 módulos

* **Sintaxis de datos: Repositorios** que convierten la sintaxis de los datos
* **Sintaxis de servicios: Facade, bridge, mediator, strategy y proxy**
* **Identidad de interfaz: Brokers** enmascara los cambios de identidad de interfaces
* **Localización: servidor de nombres**
* **Existencia: factory** asegura la creación de instancias

## Diferir tiempo de enlaces

Minimizar tiempo y costo del cambio y habilitar a no desarrolladores a hacer los cambios

### Registro en tiempo de ejecución

Operaciones plug and play para manejar la registración, ej.: suscribirse a servicios.

### Archivos de configuración

Archivos con parámetros de startup

### Polimorfismo

Para aprovechar el enlace tardío

### Reemplazo de componentes

Crear componentes intercambiables

### Adherencia a protocolos

Permitir enlaces de procesos en tiempo de ejecución

# Tácticas de performance

Objetivo: generar respuestas en el tiempo requerido

## Demanda de recursos

### Reducir recursos que hacen falta para procesar un evento

#### Aumentar la eficiencia computacional

Cambiar un recurso por otro o cambiar el algoritmo que procesa el evento

#### Reducir overhead

Eliminar intermediarios o eliminar procesos de control

### Reducir número de eventos procesados

#### Manejar la velocidad de llegadas de eventos

#### Manejar la frecuencia de muestreo de datos

### Controlar el uso de recursos

#### Manejar el tiempo de dedicación a un evento

#### Manejar el tamaño de colas de eventos

## Manejo de recursos

### Introducir concurrencia

Se centra en crear operaciones que procesen en paralelo

### Mantener copias de datos

Se centra en crear copias de datos para que la concurrencia no bloquee procesos

### Aumentar recursos disponibles

Se centra en mejorar los recursos: más y mejor

## Arbitraje de recursos

### Políticas de planificación

#### FIFO

#### Prioridad fija

Por importancia semántica, deadline y tiempo de aparición

#### Asignación dinámica

Round robin y deadline más próximo primero

#### Asignación estática

Se fijan las prioridades cuando el sistema está bajo

# Técnicas de seguridad

Objetivo: detectar, resistir y recuperarse de ataques

## Resistir ataques

### Autenticación de usuarios

Asegura que el usuario (personal o genérico) que entra al sistema sea quien dice ser (psw, one-time psw, certificados digitales, dispositivos biométricos)

### Autorización de usuarios

Control de acceso y modificación mediante grupos, permisos al usuario, roles o listas de usuarios

### Confidencialidad de los datos

Encriptar datos y canales de comunicación

### Integridad de los datos

Entregar los datos que se esperan. Para controlar esto los datos pueden tener información redundante (CRC, MD5) encriptada con los datos o de forma independiente.

### Limitar la exposición de los datos

Diseñar la ubicación de los servicios para exponer lo menos posible los datos

### Limitar el acceso

Firewalls, proxys, dmz (zonas desmilitarizadas)

## Recuperación de ataques

### Restaurar estado

Superponer tácticas de disponibilidad para recuperar el sistema a un estado consistente

### Identificar ataque

Mantener traza de auditoría

## Detección de ataques

A través de sistema de detección de intrusos

# Táctica de testeabilidad

Objetivo: probar de manera fácil cuando se modifica el software y encontrar errores.

## Input / output

### Grabar y reproducir información

Capturar información a través de una interfaz y mostrarla

### Separar interfaz de implementación

Utilizar objetos ficticios para que devuelvan los datos que se necesitan probar, es fácil intercambiarlos. Es una técnica para comenzar desde lo más complicado

### Hacer interfaces para testing (access routines / interfaces)

Monitorear el estado de los componentes y outputs

## Monitoreo interno

### Built-in monitors

Para tomar medidas de estado, carga, capacidad, seguridad o cualquier otra información accesible a través de una interfaz.

# Tácticas de usabilidad

Objetivo: que sea fácil para el usuario completar una tarea deseada y dar soporte.

## Tiempo de ejecución

### Mantener el modelo de la tarea

Determinar el contexto con el fin de que el sistema pueda tener una idea de lo que el usuario está tratando de hacer y proveerle ayuda

### Mantener el modelo del usuario

Determinar el conocimiento del usuario sobre el sistema y su comportamiento, en términos de tiempo de respuesta esperado y otros aspectos específicos a un usuario o grupo de usuarios.

### Mantener el modelo del sistema

Determina el comportamiento esperado del sistema para poder brindar feedback apropiado al usuario

## Tiempo de diseño

### Separar la interfaz gráfica de la implementación

Localizar cambios

### Soporte al usuario

# Tácticas de portabilidad

Objetivo: llevar a un sistema de una organización a otra (productos modularizados y parametrizados)

## Máquina virtual

## Interoperabilidad

### Incluír una capa de interoperabilidad

### Utilizar servicios web, API, mediante un lenguaje como SQL