Modelado y Diseño de Arquitectura de Software

Atributos de Calidad y Arquitectura de Software

Fernando Barraza A. MS.c.

fernando.barraza@gmail.com

Atributos de Calidad de AS

- Son las características esperadas del sistema derivadas del análisis de requerimientos del negocio
- Son ortogonales a la funcionalidad del sistema
- El logro de los atributos de calidad deben ser considerados en el diseño, implementación y despliegue.
- Ningún atributo de calidad es totalmente dependiente del diseño, ni es totalmente dependiente de la implementación o despliegue.

Caso de análisis: Atributos de calidad vs. Funcionalidad

- La usabilidad se refiere a aspectos que incluyen la fabricación de la interfaz de usuario clara y fácil de usar.
  - Es mejor un botón de opción o una casilla de verificación?
  - Qué diseño de la pantalla es más intuitivo?
  - Qué tipo de letra es más clara?
- A pesar de estos detalles son importantes en gran medida al usuario final no son arquitectónicos, ya que pertenecen a los detalles de diseño.
- Pero, si un sistema proporciona al usuario la posibilidad de cancelar las operaciones, para deshacer las operaciones, o volver a utilizar los datos previamente introducidos estos si se refieren a la arquitectura,

Tácticas para alcanzar la calidad en la AS

- Como el arquitecto logra las cualidades particulares esperadas para la Arquitectura?
  - Utilizando tácticas para crear un diseño basado en patrones de diseño, patrones de arquitectura, o las estrategias de arquitectura.
- Las tácticas son entonces las decisiones de diseño que sigue un arquitecto durante la creación de la arquitectura.

Conceptos relacionados a tácticas

- Una táctica es una decisión de diseño que influyen en la respuesta al atributo de calidad.
- Un conjunto de tácticas se refiere a una estrategia de arquitectura.
- Las tácticas se puede combinar con otras tácticas o refinar en un orden jerárquico.
- Un patrón de arquitectura empaqueta un conjunto de tácticas.

Caso de estudio de tácticas

- La redundancia es una táctica para incrementar la disponibilidad del sistema, pero no es la única.
- Una táctica puede refinarse con otras tácticas. En el caso de redundancia, esta puede ser de de datos (base de datos) o de computación (aplicación) que podría necesitar de sincronización entre base de datos o aplicación, lo cual se considera otra táctica.
- Los patrones empaquetan las tácticas: Un patrón que soporta la disponibilidad probablemente usa una táctica de redundancia acompañada de una de sincronización.

Evaluación de Atributos de Calidad

- Un sistema puede evaluarse de dos formas para comprobar que cumple sus atributos de calidad:
  - Atributos observables mediante la ejecución:
    - ¿Cómo se comporta el sistema durante la ejecución?
    - ¿Entrega los resultados esperados?
    - ¿Los entrega en un tiempo razonable?
    - ¿Tienen los resultados una precisión tolerable?
    - ¿Se comporta apropiadamente al interactuar con otros sistemas?
  - Atributos no observables en la ejecución:
    - ¿Cuán fácil es integrar el sistema, probarlo o modificarlo?
    - ¿Cuán caro ha sido su desarrollo?
    - ¿Cuánto se tardó en tenerlo disponible en el mercado?

Atributos de Calidad Observables durante la Ejecución

- Performance
- Seguridad
- Disponibilidad
- Funcionalidad
- Usabilidad

Performance (Desempeño)

- *Tiempo que requiere el sistema para responder a un evento o estímulo (tiempo de respuesta), o bien el número de eventos procesados en un intervalo de tiempo (throughput).*

- La performance de un sistema depende de:
  - Comunicación entre las componentes
  - Asignación de funcionalidad a las componentes
  - Algoritmos que implementan la funcionalidad
  - Codificación de los algoritmos

Dependiente de la arquitectura

No dependiente

Métricas: Tiempo de respuesta

- Se define como la medida de la latencia que una aplicación muestra en una transacción de negocio.
- Está frecuentemente (pero no exclusivamente) asociado con el tiempo que una aplicación toma en responder a una entrada
- **Ejemplo:** En una aplicación de punto de venta para una tienda grande, cuando un artículo es escaneado en la caja, en un segundo o menos tiempo el sistema responde con el precio del ítem, lo que significa que el cliente puede ser atendido rápidamente.

Requerimientos de tiempo de respuesta

- **Tiempo de respuesta garantizado:** Cada una de las peticiones deben ser atendida dentro de un límite de tiempo específico.
  - **Ejemplo:** Ninguna petición debe ser atendida después de 4 segundos.
- **Tiempo de respuesta promedio:** El tiempo promedio para atender un conjunto de peticiones debe observar un promedio, permitiendo latencia más amplia para cuando la aplicación está muy ocupada. Se incluye un tiempo límite para ser atendido.
  - **Ejemplo:** El 95% de todas las peticiones deben ser procesadas en menos de 4 segundos y ninguna petición debe tomar más de 15 segundos.

Métricas: Throughput

- Se define como la cantidad de trabajo que una aplicación debe ejecutar por unidad de tiempo
- Usualmente medida en transacciones por segundo (tps) o mensajes procesados por segundo (mps)
- Ejemplos:
  - Una aplicación de banca en línea debe garantizar que puede ejecutar 1000 transacciones por segundo para los clientes por internet.
  - Un sistema de inventarios para una bodega grande necesita procesar 50 mensajes por segundo desde sus socios de negocio.

Métricas: *Deadlines*

- Una aplicación que tiene una ventana limitada de tiempo para completar una transacción tendrá un requerimiento de *deadline* de desempeño
- Los requerimientos de *deadlines* están comunmente asociados a sistemas en lote (batch-systems)
- **Ejemplo:** Un sistema de predicción de clima toma normalmente un tiempo continuo para pronosticar el clima para el siguiente día. No se puede pasar del tiempo estimado pues de lo contrario el pronóstico sería inoficioso.

Análisis de Performance

- Puede analizarse la performance en la arquitectura mediante:
  - Análisis de las tasas de llegada
  - Distribución de los requerimientos de servicios
  - Tiempos de procesamiento
  - Tamaño de las colas
  - Latencia (tasa de servicio)
- Se puede simular el sistema mediante modelos estocásticos basados en información histórica de carga.
- Históricamente la performance ha sido muy importante, pero últimamente otras cualidades también lo son.

Tácticas para Performance

- Orientadas a controlar el tiempo en que se genera una respuesta a una petición o evento al sistema.
- La latencia es el tiempo entre la llegada de un evento o petición y la generación de una respuesta a la misma.
- Después de un evento ocurre o la petición llega, el sistema lo estará procesando o dicho procesamiento estará bloqueado por alguna razón. Esto lleva a los dos elementos básicos a analizar para el tiempo de respuesta:

  a. El consumo de recursos
  b. El tiempo de bloqueo.

a) El consumo de recursos

- Los recursos incluyen la CPU, almacenamiento de datos, ancho de banda de la red, y memoria, pero también puede incluir a los componentes definidos por el sistema.
- Por ejemplo, los buffers deben ser gestionados y el acceso a las secciones críticas deben ser secuenciales:
  1. Se genera un mensaje de uno de los componentes, se coloca en la red, y llega a otro componente.
  2. Se colocan en un búfer, transformado de alguna manera y procesado de acuerdo con algún algoritmo.
  3. Los datos son transformados para la salida, colocados en un búfer de salida, y enviado a otro componente, otro sistema o al usuario.
- Cada una de estas fases contribuyen a la latencia total de la tramitación de dicho evento.

b) El tiempo de bloqueo

- Se refiere a cuando una computación al usar un recurso se detiene momentáneamente debido a:
  - Contención del recurso: Ocurre cuando varias peticiones llegan al mismo momento. Entre más peticiones más latencia.
  - Disponibilidad del recurso: Cuando el recurso no está disponible por alguna falla momentánea.
  - Dependencia hacia otra computación: Cuando por ejemplo debe esperar los resultados de otro proceso para computarlos en conjunto y dar así una respuesta. Será mas alta la latencia si los procesos están en serie y no en paralelo.

Categorías de Tácticas para performance

- Demanda de recursos:
  - Reducir la cantidad de recursos involucrados
  - Reducir el numero de eventos procesados
  - Controlar el uso de recursos
- Administración de recursos:
  - Procesamiento en paralelo
  - Reducir puntos de acceso a datos comunes
  - Aumentar los capacidad de los recursos
- Arbitramento de recursos:
  - Definir una estrategia de programación de uso de los recursos

Demanda de recursos

- Reducir los recursos requeridos para procesar un evento o petición
  - Aumentar la eficiencia computacional: Un paso en el procesamiento de un evento o un mensaje es la aplicación de algún algoritmo. La mejora de los algoritmos utilizados en las áreas críticas reducirá la latencia.

    A veces, un recurso puede ser cambiado por otro. Por ejemplo, los datos intermedios pueden ser guardados en un repositorio o pueden ser regenerados en función del tiempo y la disponibilidad de los recursos. Esta táctica se aplica generalmente al procesador, pero también es eficaz cuando se aplica a otros recursos, como un disco.

Demanda de recursos (2)

- (cont) Reducir los recursos requeridos para procesar un evento o petición
  - Reducir el *overhead* computacional: Si no hay ninguna solicitud de un recurso, las necesidades de procesamiento se reducen.

    La utilización de intermediarios (tan importante para la modificabilidad) aumenta los recursos consumidos en la elaboración de un flujo de eventos, y por lo tanto la eliminación de ellos mejora la latencia. Este es un clásico caso de compensación entre modificabilidad y rendimiento

Demanda de recursos (3)

- Reducir el numero de eventos procesados

  - Manejar el tipo de evento: Si es posible reducir la frecuencia de muestreo en el que las variables de entorno son monitoreadas, la demanda se puede reducir. A veces esto es posible si el sistema fue 'sobreingenierizado'.

    En otras ocasiones, una velocidad de muestreo innecesariamente alta se utiliza para establecer los períodos de armonía entre varios flujos de procesos. Es decir, un conjunto de flujos de eventos son muestreados para que puedan ser sincronizados.

Demanda de recursos (4)

- (cont) Reducir el numero de eventos procesados
  - Control de frecuencia de muestreo: Si no hay control sobre la llegada de los eventos generados en el exterior, las solicitudes en cola se deberán muestrear en una frecuencia más baja, resultando potencialmente en la pérdida de las solicitudes.

Demanda de recursos (5)

- Controlar el uso de recursos
  - Limitar los tiempos de ejecución: Poner un límite en la cantidad de tiempo de ejecución que se utiliza para responder a un evento.

    A veces esto tiene sentido y a veces no. Cuando los datos dependen de algoritmos, limitar el número de iteraciones es un método de delimitación de tiempos de ejecución.

  - Limitar el tamaños de la cola: Esto controla el número máximo de entradas en la cola y por lo tanto los recursos utilizados para procesar a los recién llegados.

Gestión de recursos

- A pesar de que la demanda de recursos no se pueden controlar, la gestión de estos recursos afecta a los tiempos de respuesta. Algunas de las tácticas de manejo de recursos son:
  - Introducir la concurrencia
  - Mantener múltiples copias de datos o cálculos
  - Aumentar los recursos disponibles.

Gestión de recursos (2)

- Introducir concurrencia:
  - Si las solicitudes se pueden procesar en paralelo, el tiempo de bloqueo se puede reducir
  - La concurrencia se puede introducir mediante el procesamiento de diferentes series de eventos en diferentes hilos o mediante la creación de hilos adicionales para procesar diferentes conjuntos de actividades.
  - Una vez que la concurrencia se ha introducido, es importante una adecuada asignación de los temas a los recursos (balanceo de carga) a fin de aprovechar al máximo dicha concurrencia.

Gestión de recursos (3)

- Mantener múltiples copias de datos o cálculos:
  - El propósito de las réplicas es la reducción de la contención que se produciría si todos los cálculos se llevan a cabo en un servidor central
  - El almacenamiento en caché es una táctica en la que los datos se replican, ya sea en depósitos de diferente velocidades o en repositorios separados, para reducir la contención
  - Dado que los datos se almacenan en caché suele ser una copia de los datos existentes, manteniendo las copias coherentes y sincronizadas lo cual se convierte en una responsabilidad que debe asumir el sistema.

Gestión de recursos (4)

- Aumentar los recursos disponibles:
  - Procesadores más rápidos, más procesadores, más memoria y más ancho de banda en las redes tienen el potencial para reducir la latencia.
  - El costo es por lo general una consideración en la elección de los recursos, pero el aumento de los recursos es sin duda una táctica para reducir la latencia.

Arbitramiento de recursos

- El objetivo del arquitecto es entender las características de uso de cada recurso y elegir la política de planificación de tareas que sea compatible con ellos.
- Una política de planificación conceptualmente consta de dos partes:
  - Asignación de prioridades
  - Política de despachos.

Política de planificación de tareas

- Todas las políticas de programación asignan prioridades
- En algunos casos la tarea es tan simple como first-in/first-out. En otros casos, puede estar vinculada a la fecha de la solicitud o su importancia para el negocio.
- Criterios para la planificación de la competencia incluyen el uso óptimo de los recursos, la importancia de solicitud, minimizar el número de recursos utilizados, minimizar la latencia, el rendimiento máximo, la prevención del 'hambre' para garantizar la equidad, y así sucesivamente.
- El arquitecto tiene que ser consciente de estos criterios, posiblemente en conflicto y el efecto que la táctica elegida ha de responder a ellos.

Algunas políticas de planificación comunes

- First-in/First-out. Las colas FIFO tratan todas las solicitudes de recursos de igual a igual, por lo que es adecuada siempre y cuando todas las peticiones son realmente iguales.
- Planificación fija: Asigna a cada fuente de las solicitudes de recursos una prioridad particular y asigna los recursos en ese orden de prioridad, lo cual asegura un mejor servicio a las peticiones de mayor prioridad, pero admite la posibilidad de una prioridad baja para una solicitud importante. Tres estrategias de asignación de prioridades comunes son:
  - Importancia para el negocio
  - *Deadline* mono tónico
  - Tasa mono tónica

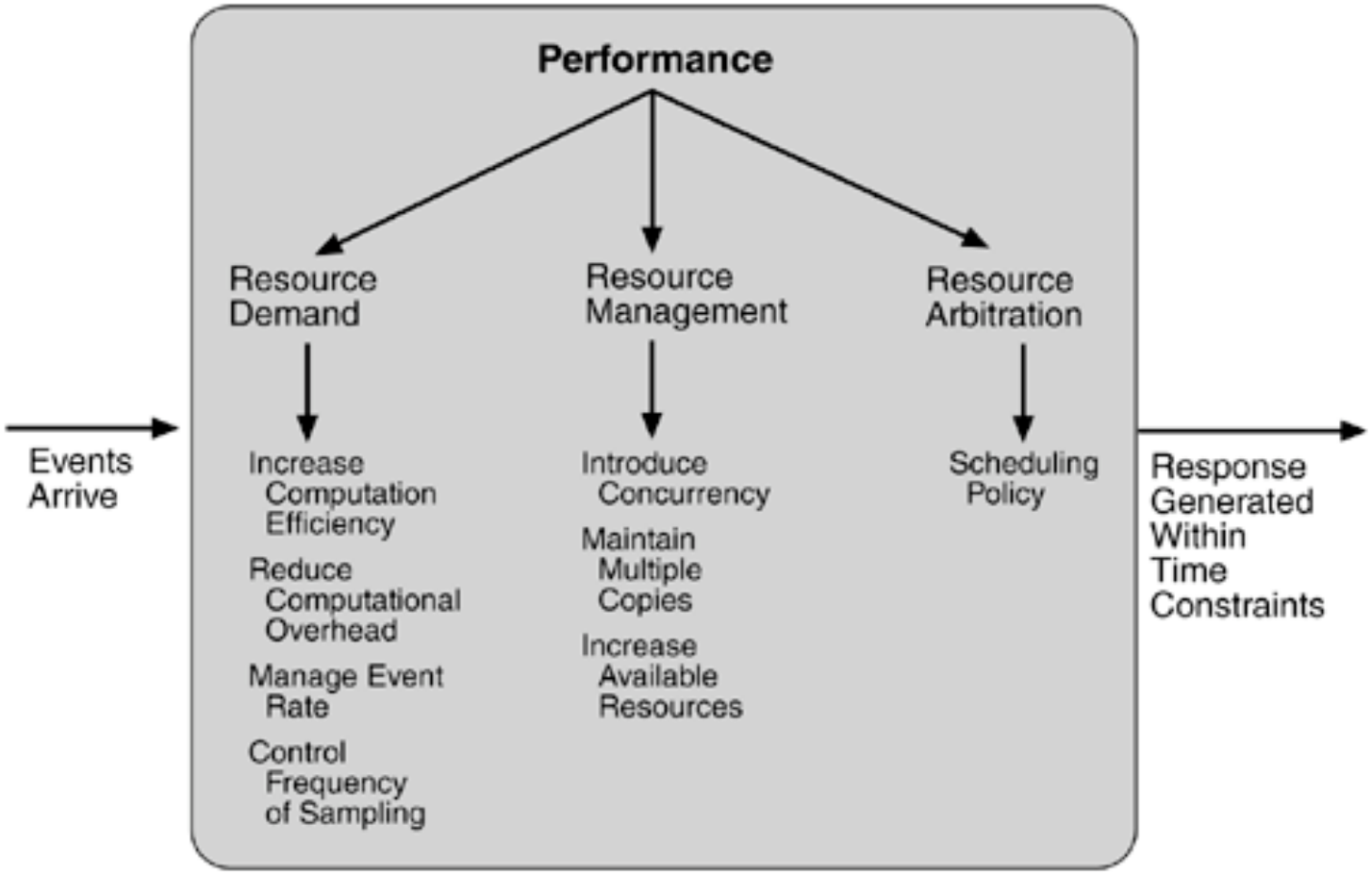
Estrategias de asignación de prioridades en planificación fija

- Importancia semántica. Cada petición tiene asignada una prioridad estática de acuerdo a alguna característica del dominio de la tarea que genera. Común en sistemas tipo mainframe.
- Fecha límite mono tónica. Asigna mayor prioridad a los peticiones con tiempos más cortos. Esta política de planificación se utiliza para peticiones en tiempo real.
- Tasa mono tónica. Es una asignación de prioridad estática para peticiones periódicas que asigna una mayor prioridad a las peticiones con períodos más cortos. Esta política de planificación es tiene más probabilidades de ser compatible con el sistema operativo.

Algunas políticas de planificación comunes (2)

- Planificación de prioridad dinámica:
  - Round-Robin: Es una estrategia de planificación que ordena las solicitudes y luego, según todas las posibilidades de asignación, asigna los recursos para la siguiente petición en ese orden. Una forma especial de round-robin es un ejecución cíclica en el cual las posibilidades de trabajo son en intervalos de tiempo fijos.
  - Fecha más temprana en primer lugar: Asigna prioridades en base a las solicitudes pendientes con el plazo más temprano.
- Planificación estática. Es una estrategia de Planificación en donde los puntos de chequeo y la secuencia de la asignación de los recursos se determinan fuera de línea.

Resumen de tácticas de performance



Seguridad

- *Medida de la capacidad del sistema para resistir intentos de uso y negación de servicios a usuarios no autorizados sin restar servicios a los usuarios autorizados.*
- Típicos ataques:
  - Negación de servicios - impedir que un servidor pueda dar servicios a sus usuarios. Se hace inundándolo con requerimientos o consultas.
  - Impostor de dirección IP - el atacante asume la identidad de un host confiable para el servidor. El atacante usualmente inhibe al host confiable mediante negación de servicios.

Requerimientos de Seguridad

- Non-repudiation: Quien envía el mensaje (sender) tiene prueba de entrega y el receptor (receiver) está seguro de la identidad de quien lo envía. Es decir, no puede refutar su participación en el intercambio del mensaje.
- Confidencialidad: La confidencialidad es la propiedad que los datos o los servicios están protegidos contra el acceso no autorizado.
- Integridad: Es la propiedad que los datos son entregados según lo previsto. Esto significa que un dato en su calidad no cambia desde que ha sido elaborado y durante su tránsito.
- Aseguramiento: Es cuando las partes en una transacción son las que pretenden ser.

Requerimientos de Seguridad (2)

- Disponibilidad: Es la propiedad en la que el sistema está disponible para su uso legítimo. Esto significa que un ataque de denegación de servicio no evitará que la transacción sea completada.
- Auditoría: Es la propiedad en la que el sistema permite realizar un seguimiento de las actividades dentro de el a niveles suficientes para reconstruirlas. Esto significa que, por ejemplo si se hace una transferencia de dinero de una cuenta a otra cuenta, el sistema mantendrá un registro de dicha transferencia.

Estrategias de AS para Seguridad

- Sirven para prevenir, detectar y responder a ataques de seguridad:
  - Disponer un servidor de autentificación entre los usuarios externos y la porción del sistema que da los servicios;
  - Instalar monitores de redes para la inspección y el registro de los eventos de la red;
  - Disponer el sistema detrás de un "firewall" de comunicaciones donde toda comunicación desde y hacia el sistema se canaliza a través de un proxy;
  - Construir el sistema sobre un kernel confiable que proporciona servicios de seguridad.
- Todas estas estrategias implican identificar componentes especializados y coordinar su funcionamiento con las demás componentes.

Tácticas de Seguridad

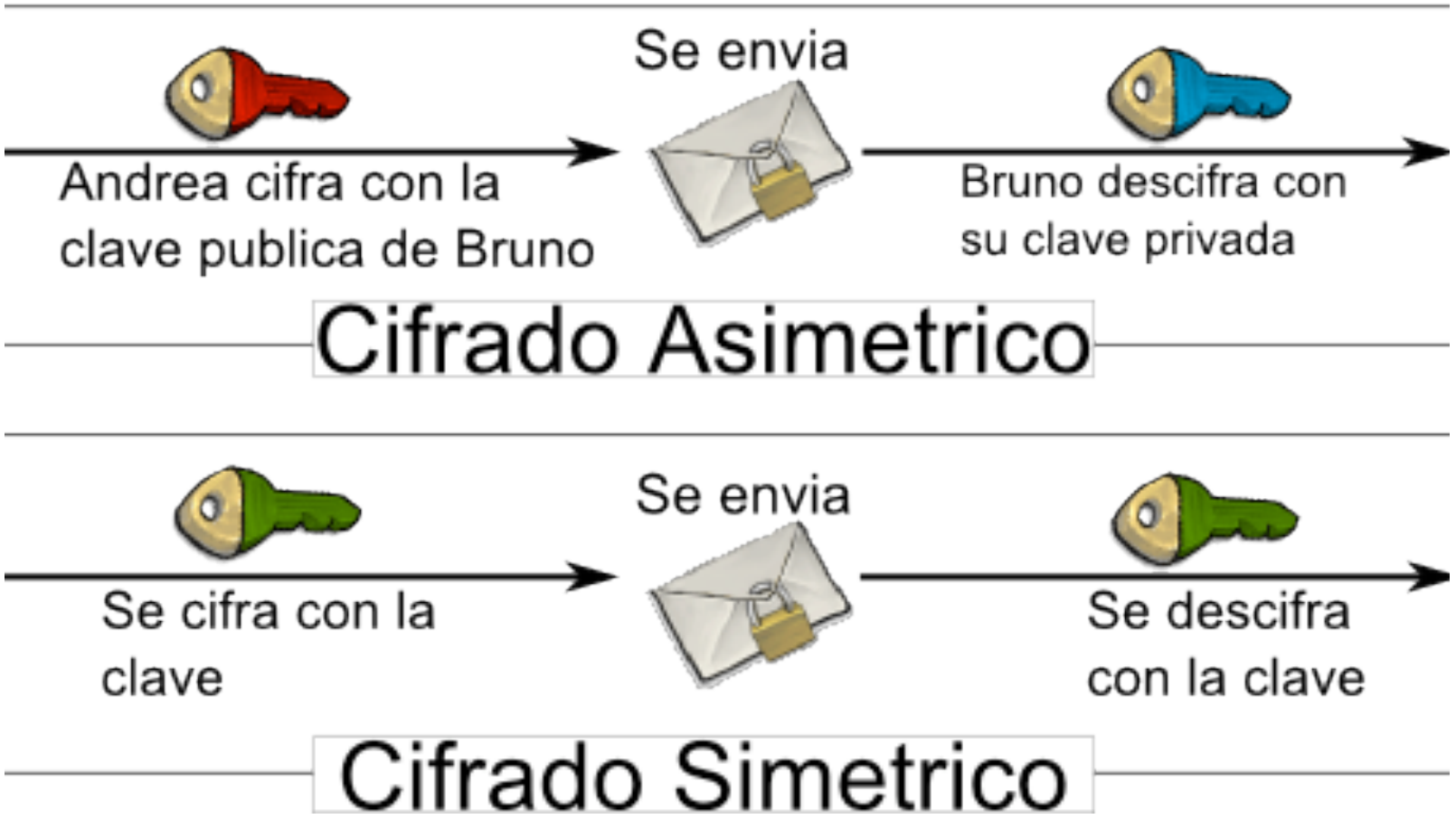
- Resistir a los ataques:
  - Autenticar a los usuarios. La autenticación es asegurar que un usuario o un equipo remoto es realmente quien dice ser. Contraseñas, contraseñas de una sola vez, certificados digitales, y las identificaciones biométricas proporcionan autenticación.
  - Autorizar a los usuarios. La autorización es garantizar que un usuario autenticado tiene los derechos para acceder y modificar datos o servicios. Esto es generalmente administrado ofreciendo algunas pautas de control de acceso dentro de un sistema. Control de acceso puede ser por usuario o por clase de usuario. Clases de usuarios pueden ser definidos por grupos de usuarios, por los roles de usuario, o por listas de las personas.

Tácticas de Seguridad (2)

- (cont) Resistir a los ataques:
  - Mantener la confidencialidad de los datos. Los datos deben estar protegidos contra el acceso no autorizado. La confidencialidad se logra generalmente mediante la aplicación de algún tipo de encriptación de datos y enlaces de comunicación.

    El cifrado es la única protección para pasar datos a través de enlaces de comunicación de acceso público. El enlace puede ser implementado por una red privada virtual (VPN) o por una capa de sockets seguros (SSL) para un enlace basado en la Web. Cifrado puede ser simétrico (ambas partes utilizan la misma clave) o asimétrico (claves públicas y privadas).

Criptografía simétrica y asimétrica



Tácticas de Seguridad (3)

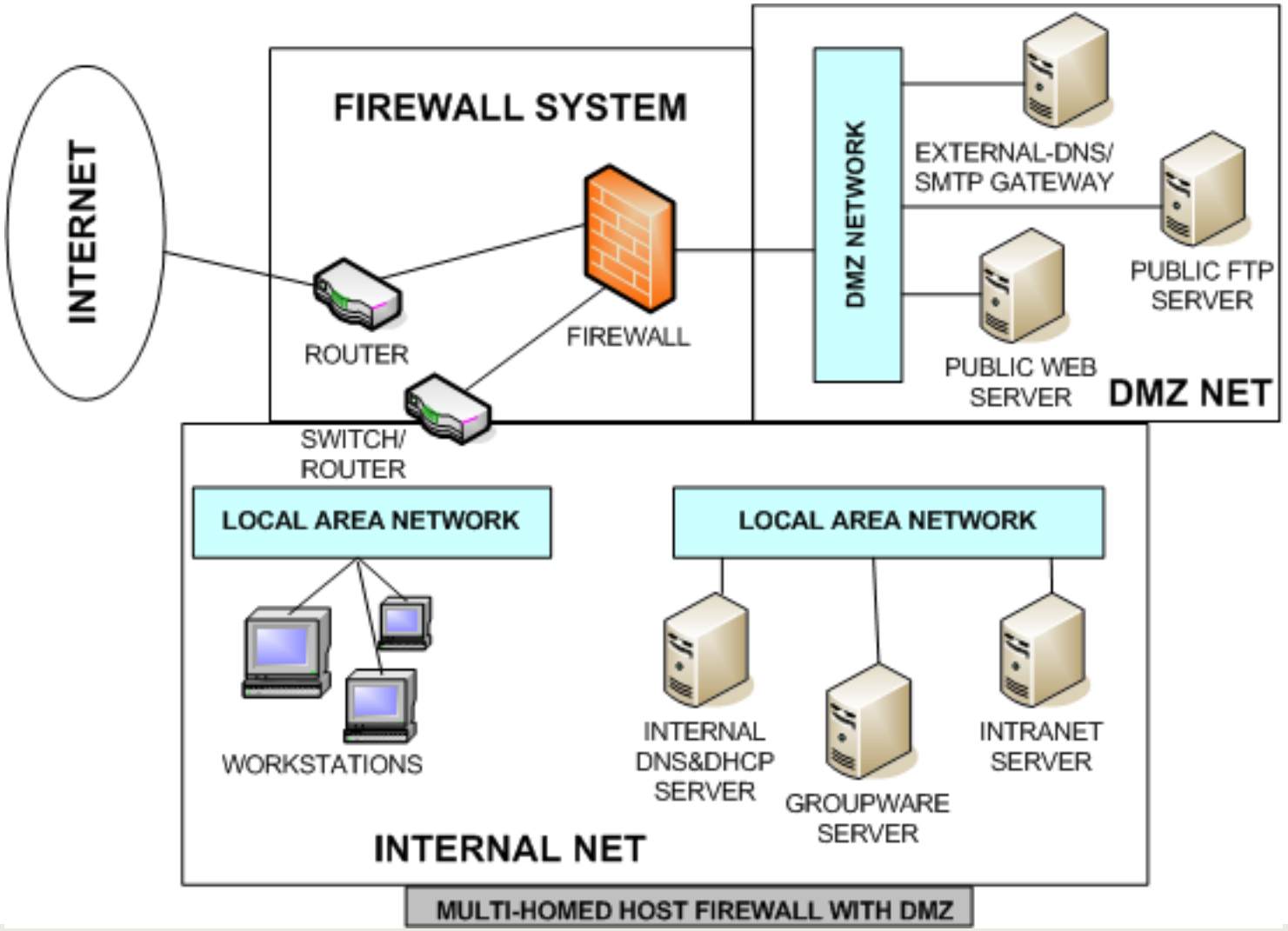
- (cont) Resistir a los ataques:
  - Mantener la integridad. Los datos deben ser entregados según lo previsto. Puede tener información redundante codificada en ella, como sumas o los resultados de hash, que pueden ser encriptados con o independientemente de los datos originales.
  - Límite de exposición. Los ataques suelen depender de la explotación de una sola debilidad para atacar a todos los datos y servicios en un host. El arquitecto puede diseñar la asignación de servicios a los servidores para que los servicios disponibles estén limitados en cada host.

Tácticas de Seguridad (4)

- (cont) Resistir a los ataques:
  - Limitar el acceso. Los firewalls restringen el acceso basado en la fuente del mensaje o puerto de destino. Los mensajes de fuentes desconocidas puede ser una forma de ataque pero no se pueden limitar siempre. Un sitio web público, por ejemplo, puede obtener las solicitudes desde fuentes desconocidas.

    Una configuración que se utilizan en este caso es la llamada zona desmilitarizada (DMZ) cuando se debe proporcionar acceso a los servicios de Internet, pero no a una red privada. Se encuentra entre Internet y un firewall delante de la red interna. La DMZ contiene dispositivos de espera para recibir mensajes de fuentes arbitrarias tales como servicios Web, correo electrónico y los servicios de nombres de dominio (DNS).

DMZ


INTERNET
FIREWALL SYSTEM
ROUTER
FIREWALL
SWITCH/
ROUTER
DMZ NETWORK
EXTERNAL-DNS/
SMTP GATEWAY
PUBLIC FTP
SERVER
PUBLIC WEB
SERVER
DMZ NET
LOCAL AREA NETWORK
LOCAL AREA NETWORK
WORKSTATIONS
INTERNAL
DNS&DHCP
SERVER
GROUPWARE
SERVER
INTRANET
SERVER
INTERNAL NET
MULTI-HOMED HOST FIREWALL WITH DMZ

Tácticas de Seguridad (5)

- Detección de ataques:

La detección de un ataque es generalmente a través de un sistema de detección de intrusos. Estos sistemas trabajan mediante la comparación de los patrones de tráfico de red con una base de datos.

Puede implementarse para detectar:

- Mal uso: El patrón de tráfico se compara con los patrones históricos de ataques conocidos
- Anomalías: El patrón de tráfico se compara con una base histórica de sí mismo.

Tácticas de Seguridad (6)

- Recuperación de ataques:

Tácticas que participan en la recuperación de un ataque se puede dividir en:

- Las relacionados con la restauración de estado
- Las relacionados con la identificación atacante (con fines de carácter preventivo o punitivo).

Recuperación de ataques

- Las tácticas utilizadas en la restauración del sistema o los datos a un estado correcto coinciden con los utilizados para la disponibilidad ya que ambos están preocupados por la recuperación de un estado consistente en un estado incoherente.

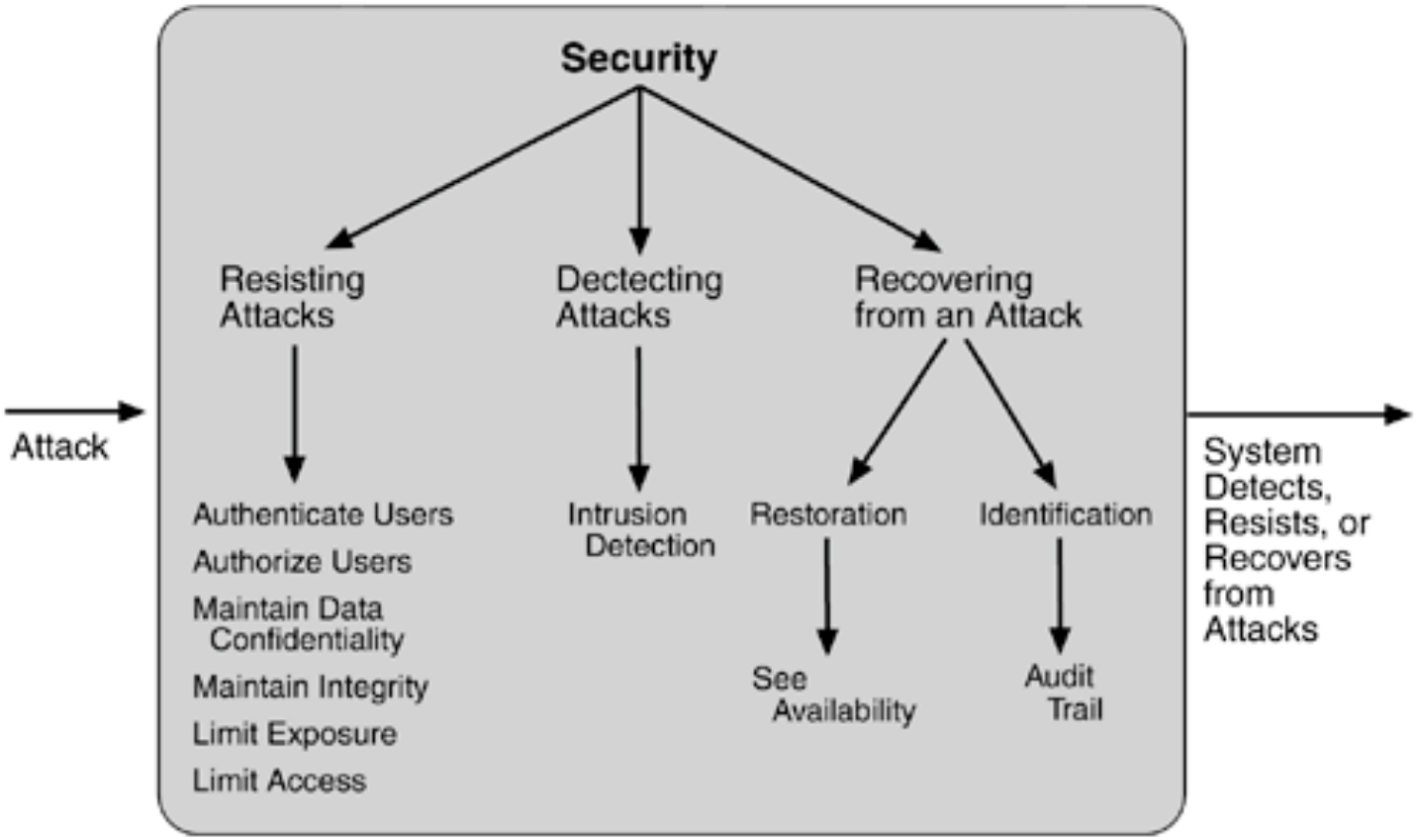
  Una diferencia es que se presta especial atención al mantenimiento de las copias redundantes de datos administrativos del sistema, como contraseñas, listas de control de acceso, servicios de nombres de dominio, y los datos de perfil de usuario.

Recuperación de ataques (2)

- La táctica para la identificación de un atacante es mantener una pista de auditoría.

  Una pista de auditoría es una copia de cada transacción aplicada a los datos en el sistema junto con información de identificación. La información de auditoría puede utilizarse para rastrear las acciones de un atacante, el soporte para non-repudiation (que proporciona evidencia de que una solicitud si se hizo), y la recuperación del sistema de apoyo. Pistas de auditoría son a menudo también objeto de ataques por lo que se debe mantener de manera segura igualmente.

Resumen Tácticas de Seguridad



Disponibilidad

- Proporción del tiempo que el sistema está en ejecución. Ej: 99,98%
- Se mide como el tiempo entre fallas o la rapidez en que el sistema puede reiniciar la operación cuando ocurre una falla.

$$\alpha = \frac{tiempo\_promedio\_entre\_fallas}{tiempo\_promedio\_entre\_fallas + tiempo\_medio\_reparación}$$

Estrategias

- Las arquitecturas redundantes o con alta recuperabilidad son utilizadas para obtener alta disponibilidad. Ej: Canales de comunicación, Componentes.
- Arquitecturas con registro dinámico de componentes facilitan el mantenimiento y favorecen la disponibilidad del sistema
- Las tecnologías como granjas de servidores, arreglos de discos, servidores en cluster y grid computing están orientadas a mejorar la disponibilidad del sistema.

Recuperabilidad y Disponibilidad

- Son dos conceptos altamente relacionados
- Una aplicación es recuperable si tiene la capacidad de restablecer los niveles de desempeño y recuperar los datos afectados después de una falla del sistema.
- **Ejemplo:** Un sistema de base de datos. Cuando un servidor de base de datos falla pasa a un estado de no disponibilidad hasta que se haya recuperado. Esto implica reiniciar el servidor de aplicaciones y resolver cualquier transacción que estaba en ejecución cuando la falla ocurrió.

Tácticas de disponibilidad

- Detección de fallas:

Consiste en implementar algún mecanismo que permita darse cuenta al sistema por si mismo que una falla ha ocurrido.

Son básicamente tres tácticas:

- Ping/echo
- Heartbeat
- Excepciones.

Detección de fallas

- Ping / echo: Un componentes hace un ping y espera recibir de regreso un eco, dentro de un tiempo predefinido, desde el componente bajo escrutinio.

  Puede ser usado:

  - Dentro de un grupo de componentes mutuamente responsables de una tarea.
  - Por un componente cliente para asegurarse de que un objeto de servidor y la ruta de comunicación con el servidor está operando dentro de los límites de rendimiento esperado.

Detección de fallas (2)

- *Heartbeat*: En este caso uno de los componentes emite periódicamente un mensaje de *heartbeat* y otro componente lo escucha. Si un *heartbeat* no llega al que escucha, se supone que el componente de origen ha fallado y un componente de corrección de error es notificado.

  El *heartbeat* también puede transportar datos. Por ejemplo, un cajero automático puede enviar periódicamente el registro de la última transacción a un servidor. Este mensaje no sólo actúa como un *heartbeat*, sino que también tiene los datos a ser procesados.

Tácticas de disponibilidad (2)

- Recuperación de fallas: Consiste en la preparación para la recuperación y hacer la reparación del sistema.

  Algunas tácticas de preparación y reparación son:
  - Votación
  - Redundancia activa
  - Redundancia pasiva
  - Spare (reposición)
  - Operación en la sombra
  - Re-sincronización de estado
  - Checkpoint/rollback.

Tácticas de disponibilidad (3)

- Prevención de fallas:
  - Separación del servicio. Esta táctica elimina un componente del sistema en la operación de algunas de las actividades para prevenir posibles fallos previstos.

    Un ejemplo es el de reiniciar un componente para evitar pérdidas de memoria que podrían causar una falla. Si el retiro del servicio es automático, una estrategia de arquitectura puede ser diseñada para apoyar dicha separación del servicio.

Tácticas de disponibilidad (4)

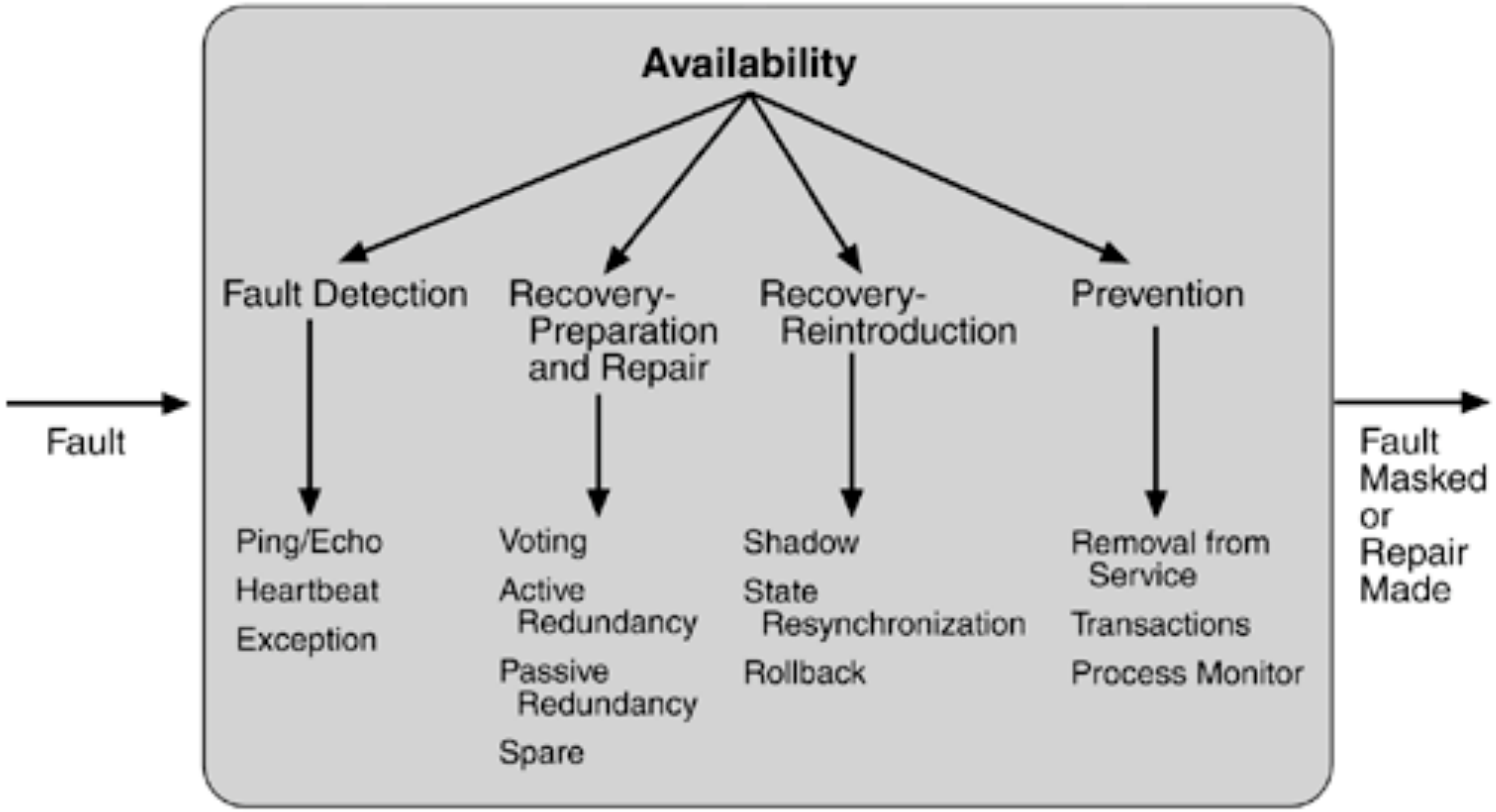
- (cont) Prevención de fallas:
  - Transacciones. Una transacción es la agrupación de varias etapas sucesivas en un paquete de tal manera que todo el paquete se puede deshacer a la vez.

    Las Transacciones se utilizan para evitar que los datos se vean afectados si un paso en un proceso falla, y también para evitar las colisiones entre varios hilos simultáneos que acceden a los mismos datos.

Tácticas de disponibilidad (5)

- (cont) Prevención de fallas:
  - Monitoreo de procesos. Una vez que un fallo en un proceso se ha detectado, un proceso de monitoreo puede eliminarlo y crear una nueva instancia del mismo, inicializado a un estado adecuado.

Resumen tácticas de disponibilidad



Escalabilidad

- Es la capacidad de la aplicación para funcionar correctamente si el tamaño de procesamiento se incrementa.
- La escalabilidad debe ser alcanzada sin modificaciones de la arquitectura subyacente, a parte de los cambios inevitables de configuración.
- Una arquitectura no escalable, cuando el tamaño de procesamiento se incrementa su throughput se decrementa y el tiempo de respuesta se incrementa exponencialmente.

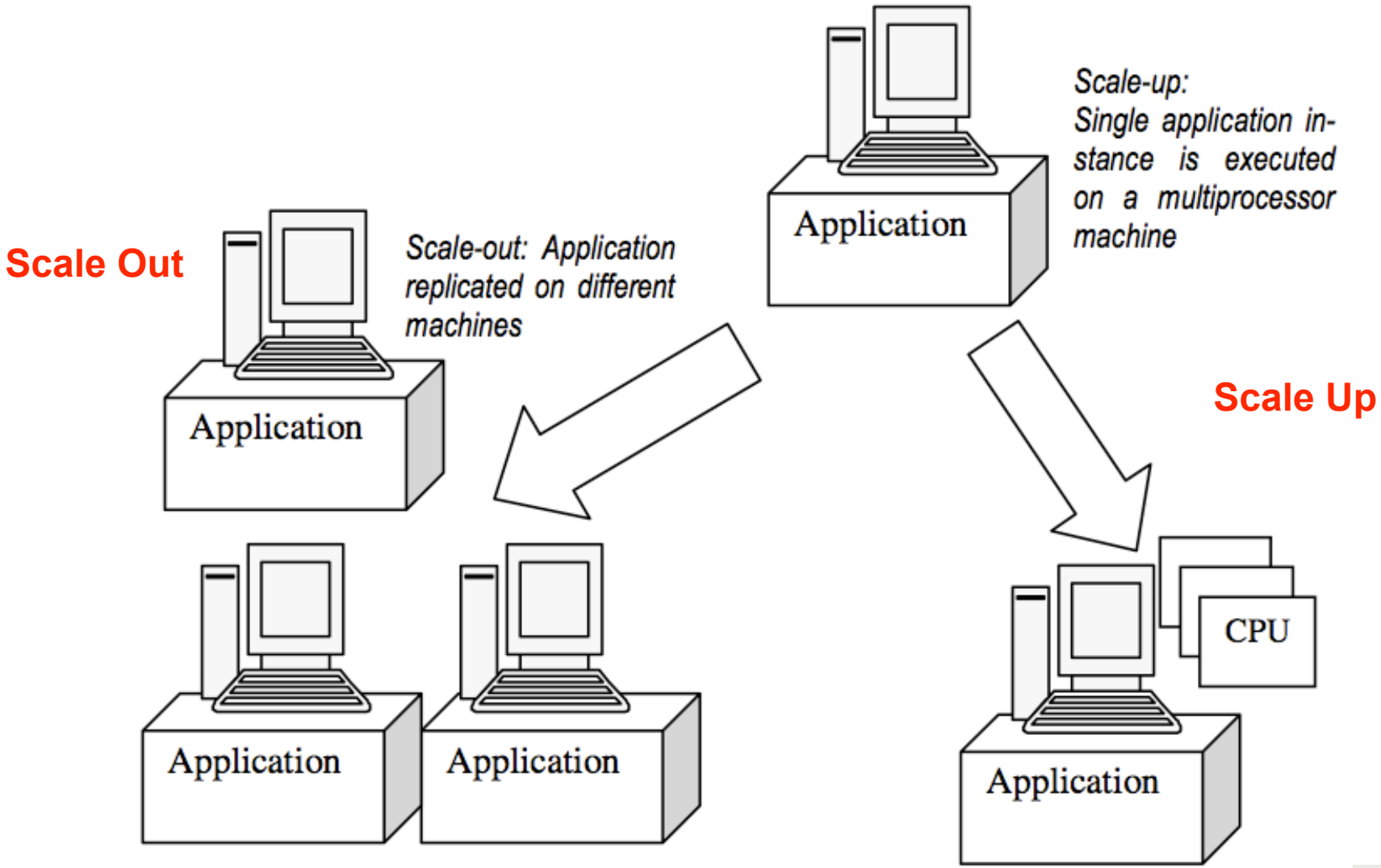
Requerimientos de Escalabilidad

- El tamaño de procesamiento se refiere a requerimientos del tipo de:
  - Requerimientos de carga
  - Conexiones simultaneas
  - Tamaño de los datos
  - Despliegue

Requerimiento de carga

- Una arquitectura escalable debe permitir adicionar capacidad de procesamiento para incrementar el throughput y decrementar el tiempo de respuesta.
- **Ejemplo:** Una aplicación está diseñada para soportar 100 tps como carga pico, con un promedio de 1 segundo de tiempo de respuesta. Si el requerimiento de carga crece en 10 veces entonces el throughput debería permanecer constante (100 tps) y el tiempo de respuesta por petición debería incrementarse de forma línea, es decir 10 segundos.

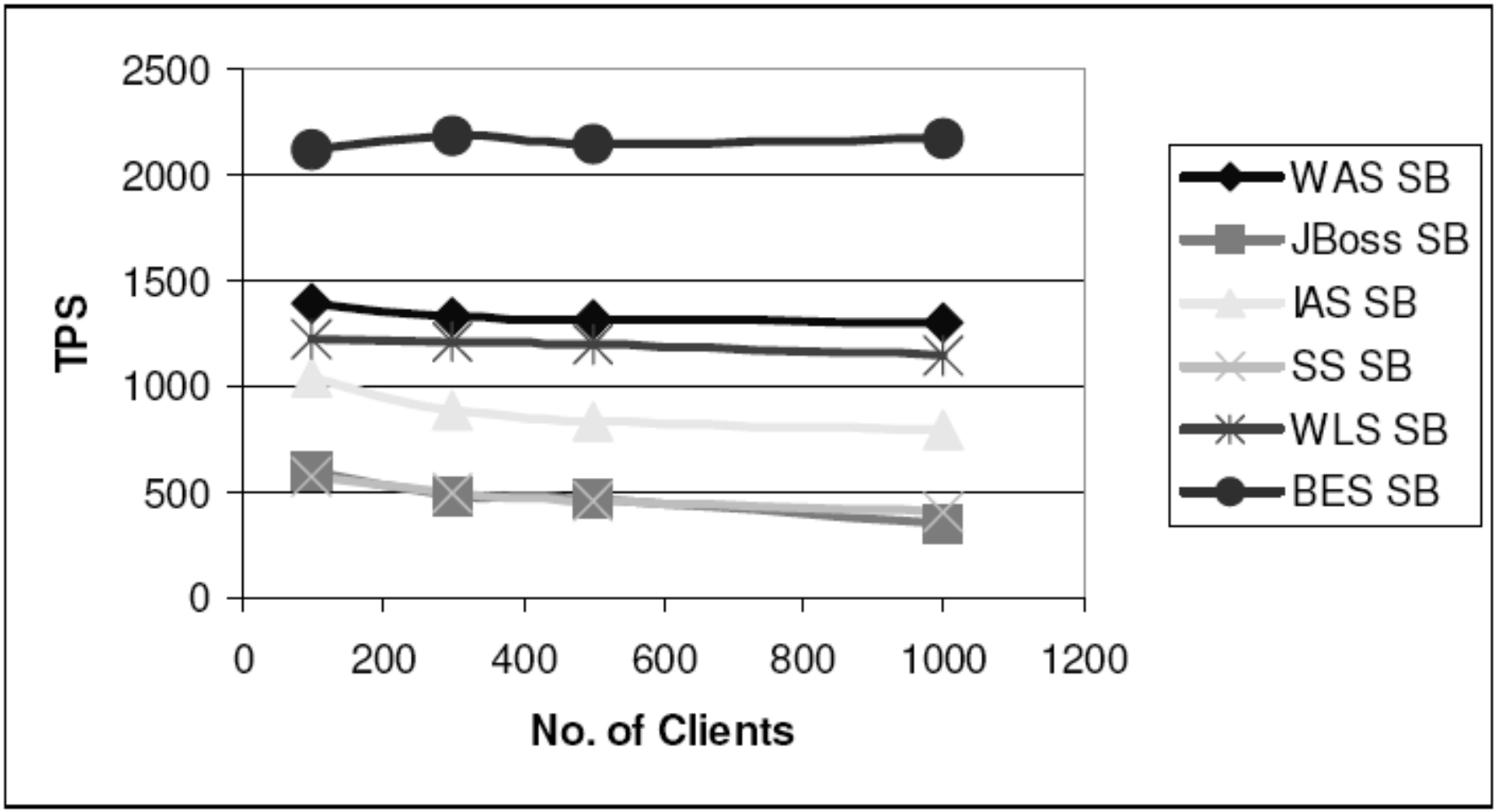
Estrategias de Escalamiento



Conexiones simultáneas

- Se refiere a la necesidad de soportar un numero mayor de conexiones / usuarios al numero habitual de los que de manera concurrente acceden a la aplicación
- Normalmente cada nueva conexión incrementa el consumo de recursos del sistema, por lo que es una mala idea soportar la escalabilidad en el aumento de esos recursos
- Los picos del requerimiento de conexiones simultáneas están fuertemente ligados a los ciclos del negocio

Análisis de escalamiento



Estrategias para escalar conexiones

- Limitar por software el número de conexiones posibles de acuerdo al recurso disponible (naive)
- Favorecer arquitecturas sin estado
- Scale-out
- Algoritmos perezosos

Tácticas de escalamiento

- En general se puede hacer una combinación entre las tácticas de disponibilidad y las tácticas de performance
- Con las tácticas de disponibilidad se garantiza que el sistema seguirá operando ante un incremento de carga
- Con las tácticas de performance se garantiza que el sistema mantendrá un desempeño razonable dentro de los nuevos límites de operación
- La combinación más común es combinar la prevención (disponibilidad) con la gestión de recursos (performance).

Usabilidad

- *Aprendibililidad* - ¿Cuán rápido y fácil es para un usuario el aprender a usar la interfaz del sistema?
- *Eficiencia* - ¿El sistema responde con la rapidez apropiada a las exigencias del usuario?
- *Recordabilidad* - ¿Pueden los usuarios recordar cómo usar el sistema entre dos sesiones de uso?
- *Propenso a errores* - ¿El sistema anticipa y previene los errores comunes de los usuarios?
- *Manejo de errores* - ¿El sistema ayuda a los usuarios a recuperarse de los errores?
- *Satisfacción* - ¿El sistema facilita la tarea de los usuarios?

Usabilidad y Arquitectura

- Gran parte de los mecanismos para lograr usabilidad no tienen relación con la arquitectura:
  - modelo mental del usuario del sistema reflejado en la interfaz usuaria,
  - distribución de elementos y colores en la pantalla.
- Otros elementos sí tienen relación con la arquitectura:
  - la información relevante para el usuario debe estar disponible para una determinada interfaz
    - debe disponerse de un conector que traiga esta información al componente que corresponda
  - la eficiencia tiene implicancias en la usabilidad.

Tácticas para Usabilidad

- En tiempo de ejecución: Dependen de si quién toma la iniciativa en la interacción:
  - User-Initiative: El usuario toma la iniciativa, por ejemplo presiona el botón de cancelar.
  - System-Initiative: El sistema toma la iniciativa, por ejemplo el sistema despliega información adicional de acuerdo al contexto de navegación del usuario.
  - Mixed-Initiative: Combina las anteriores, por ejemplo, el sistema muestra una barra de progreso ante una operación de cancelar del usuario.

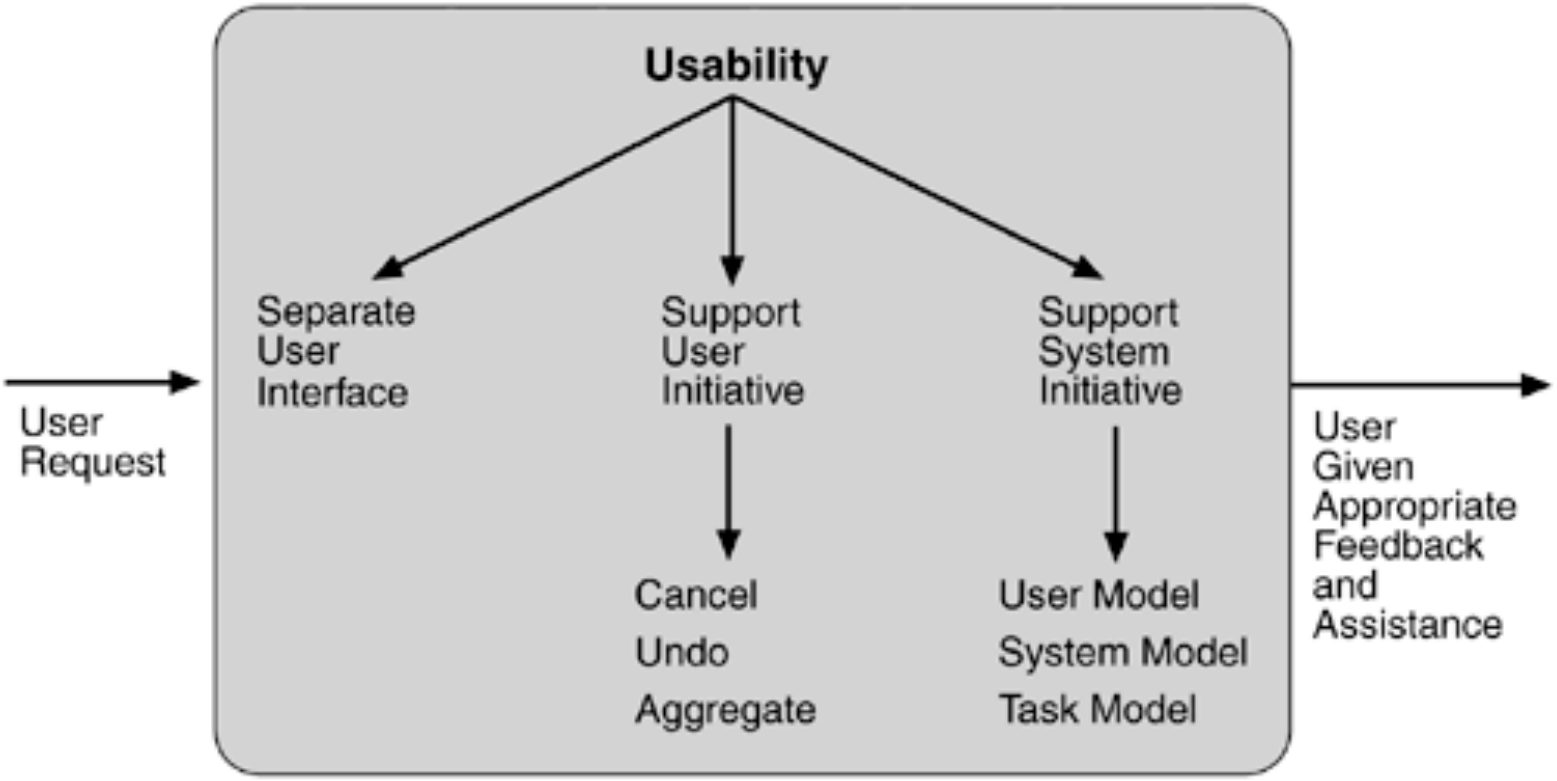
System-Initiative

- Se soporta en un modelo de información que puede ser acerca de:
  - El usuario: Por ejemplo, pasar el scroll de una página de acuerdo al patrón de lectura del usuario.
  - La tarea: Por ejemplo corregir la primera letra en mayúscula para nombres de personas cuando el usuario escribe una carta
  - El sistema: Por ejemplo, determinar el tiempo esperado para completar una actividad en la cual el usuario da ordenes al sistema.

Tácticas de usabilidad (2)

- En tiempo de diseño:
  - Separar la interfaz de usuario del resto de la aplicación. Dado que la interfaz de usuario se espera que cambie con frecuencia, durante el desarrollo y después de la implantación, el mantenimiento del código de interfaz de usuario por separado es una buena idea.
  - Los patrones de arquitectura de software para poner en práctica esta táctica son los siguientes:
    - Modelo-Vista-Controlador
    - Presentation-Abstraction-Control
    - Seeheim
    - Arch / Slinky

Resumen táctica de usabilidad



Atributos de Calidad no Visibles durante la Ejecución

- Modificabilidad
- Portabilidad
- Reusabilidad
- Integrabilidad
- Testabilidad

Modificabilidad

- *Habilidad para hacer cambios al sistema de una forma rápida y poco costosa.*
- Es el atributo de calidad más íntimamente relacionado con la arquitectura.
- Hacer modificaciones en un sistema consiste en dos etapas:
  - localizar el (los) lugar(es) dónde debe aplicarse el cambio
  - aplicar el cambio propiamente

Modificabilidad en el Proceso de Desarrollo

- Ya desde el desarrollo es deseable que un sistema sea mantenible ya que pasa por múltiples etapas:
  - control de versiones,
  - control de configuración,
  - cuanto más locales los cambios, mejor.
- La arquitectura define:
  - las componentes y sus reponsabilidades,
  - las condiciones en que cada componente puede cambiar.

Modificabilidad y Arquitectura

- Las modificaciones suelen venir de cambios en el negocio:
  - extender o cambiar la capacidades - la extensibilidad permite seguir siendo competitivo en el mercado
  - quitar capacidades no deseadas - simplificar el producto para hacerlo más liviano o más barato
  - adaptarse a nuevos ambientes de ejecución - portabilidad hace al producto más flexible para más clientes
  - reestructurar - racionalizar servicios, modularizar, optimizar, o crear componentes reusables para futuros desarrollos

Tácticas de modificabilidad

- Localizar modificaciones: En general entre menos módulos afectados por un cambio menos costoso será este. Las tácticas comunes son:
  - Mantener coherencia semántica: Se refiere a las relaciones entre las responsabilidades en un mismo módulo. El objetivo es asegurar que todas estas responsabilidades se trabajan en conjunto sin excesiva confianza en los demás módulos.
  - Anticipar cambios esperados: Es anticipar los cambios esperados sin preocuparse por la coherencia de las responsabilidades de un módulo, sino más bien de la minimización de los efectos de los cambios. Esta táctica es difícil de utilizar por sí misma, ya que no es posible anticipar todos los cambios. Por esa razón, se suelen utilizar en combinación con la coherencia semántica.

Tácticas de modificabilidad (2)

- (cont) Localizar modificaciones:
  - Generalizar los módulos: Hacer un módulo más general permite calcular una amplia gama de funciones basadas en la entrada. La entrada se puede considerar como la definición de un lenguaje para el módulo, que puede ser tan simple como usar parámetros constantes de entrada o tan complicado como la implementación de un módulo intérprete y que los parámetros de entrada sean un programa en el lenguaje del intérprete.

    Entre más general de un módulo, es más probable que los cambios solicitados se pueden hacer mediante el ajuste del lenguaje de entrada en lugar de modificar el módulo.

Tácticas de modificabilidad (3)

- (cont) Localizar modificaciones:
  - Limitar posibles opciones: Delimitar las capacidades del sistema para distribuirse en diferentes sistemas de plataforma y/o modos de ejecución o con diferentes tipos de tecnologías y componentes. Esto limita el numero de posibles cambios debido a la necesidad de soportar nuevas capacidades de operación. Ejemplo: Limitar los tipos de procesadores soportados para la aplicación.

Tácticas de modificabilidad (4)

- Prevenir el efecto dominó:

  Un efecto dominó de una modificación es la necesidad de hacer cambios a los componentes que no están directamente afectados por ellos.

  Por ejemplo, si el componente A se cambia para llevar a cabo una modificación en particular, entonces el componente B se cambia sólo por el cambio de componente A. B tiene que ser modificada debido a que depende, en cierto sentido, en A.

  El efecto dominó esta determinado por el tipo de dependencias entre los componentes.

Tipos de dependencias entre componentes

- Sintaxis de:
  - Datos: Para compilar B (o ejecutar) correctamente, el tipo de dato (o formato) que es producido por A y consumido por B debe ser consistente con el tipo (o formato) de dato asumido por B.
  - Servicio: Para B compilar y ejecutarse correctamente, la firma de los servicios provistos por A e invocados por B debe ser consistente con lo asumido por B.

Tipos de dependencias entre componentes (2)

- Semántica de:
  - Datos Para B ejecutarse correctamente, la semántica de los datos producidos por A y consumidos por B debe ser consistente con las asunciones de B.
  - Servicio: Para B ejecutarse correctamente, la semántica de los servicios producidos por A y usados por B debe ser consistente por lo asumido por B.

Tipos de dependencias entre componentes (3)

- Secuencia de:
  - Datos: Para B ejecutarse correctamente, debe recibir datos producidos por A en una secuencia fija.
  - Control: Para B ejecutarse correctamente, A debe haberse ejecutado previamente con cierta restricción de tiempo. Por ejemplo, A debe haberse ejecutado no mas de 5ms antes de que B se ejecutara.

Tipos de dependencias entre componentes (4)

- Identidad de una interfaz: A puede tener múltiples interfaces. Para B compilar y ejecutarse correctamente, la identidad (nombre o *handler*) de la interfaz debe ser consistente con lo asumido por B.
- Localización (runtime). Para B ejecutarse correctamente, la localización de runtime de A debe ser consistente con lo asumido por B. Por ejemplo, B puede asumir que A está localizado en un proceso diferente en el mismo procesador.

Tipos de dependencias entre componentes (5)

- QoS/data: Para B ejecutarse correctamente, alguna propiedad que involucra la calidad de los datos o servicios provistos por A debe ser consistente con lo asumido por B. Por ejeplo, datos de un sensor deben tener cierta exactitud para que un algoritmo de B trabaje correctamente.
- Existencia: Para que B trabaje correctamente, A debe existir o ser creado dinámicamente.
- Comportamiento de recursos. Para que B se ejecute correctamente, el comportamiento de los recursos de A deben ser consistente con lo asumido por B.

Tácticas de modificabilidad (2)

- (cont) efecto domino:
  - Esconder información: Descomponer responsabilidades de un componente en partes más pequeñas determinando cuales sean públicas y cuales privadas.
  - Mantener las interfaces existentes: Si B depende de A no cambiar la interfaz de A hacia B, a no ser que haya una dependencia semántica.

    Para mantener una interfaz estable se puede
    - Separar la interfaz de la implementación
    - Utilizar el patrón adaptador

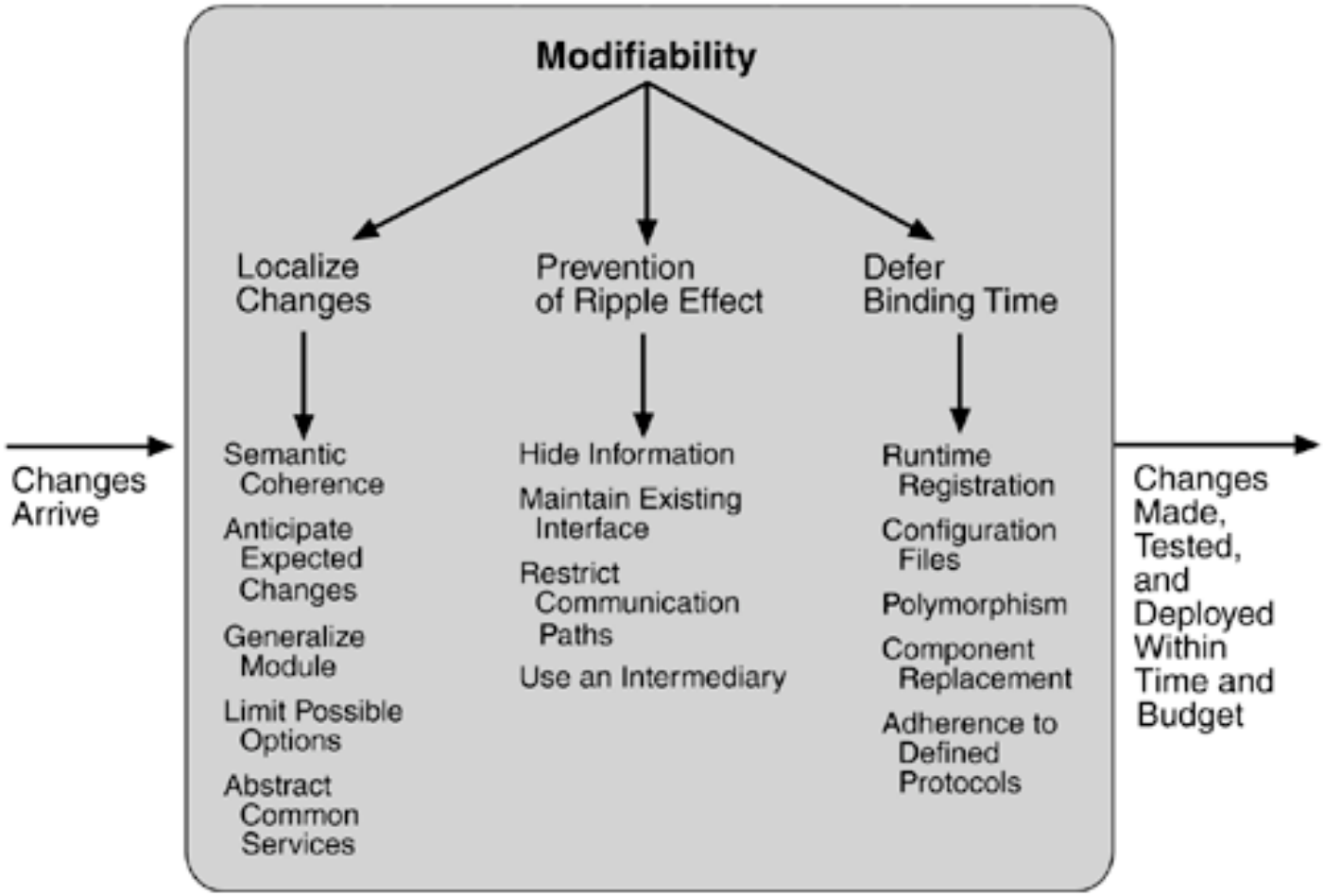
Tácticas de modificabilidad (3)

- (cont) efecto domino:
  - Restringir los caminos de comunicación: Restringir los módulos con los que comparte un determinado módulo de datos. Es decir, reducir el número de módulos que consumen datos producidos por el mismo módulo y el número de módulos que producen los datos consumidos por ellos. Esto reducirá el efecto multiplicador ya que la producción de datos / consumo introduce dependencias.
  - Introducir un intermediario: Si B tiene algún tipo de dependencia que no sea semántica, es posible insertar un intermediario entre B y A que gestione las actividades relacionadas con la dependencia.

Tácticas de modificabilidad (3)

- Diferir el tiempo de enlazado: Son tácticas en tiempo de configuración y ejecución:
  - Registro dinámico de componentes
  - Archivos de configuración separados del ejecutable
  - Polimorfismo
  - Reemplazo dinámico de componentes
  - Protocolos de enlace o gateways

Resumen de tácticas de modificabilidad



Portabilidad

- *Habilidad de un sistema para ejecutar en diferentes ambientes (hardware, software, o una combinación de ambos)*.
- Un sistema es portable si todas las supocisiones acerca del ambiente particular de ejecución se confinan a una única (o unas pocas) componente(s).
- En una arquitectura, el encapsulamiento de las dependencias se hace en una *capa de portabilidad* que da al sistema una serie de servicios independientes del ambiente.

Reusabilidad

- *Es la capacidad que tiene un sistema para que su estructura o alguna de sus componentes puedan ser usadas en futuras aplicaciones.*
- La reusabilidad se relaciona con la arquitectura en que las componentes son las principales unidades de reutilización:
  - la reusabilidad dependerá del acoplamiento de la componente,
  - reusar una componente implica reusar la clausura transitiva de todas las componentes dependientes en la estructura de uso
- La reusabilidad es una forma particular de modificabilidad

Reusabilidad y Modificabilidad

1. Un sistema S está compuesto de 100 componentes (S_1 a S_{100}). Se construye otro sistema T, y se descubre que las *n* primeras componentes son idénticas y pueden reutilizarse.

2. Un sistema S compuesto por 100 componentes está funcionando y se desea modificarlo. Las modificaciones sólo afectan 100-n componentes dejando a las restantes incambiadas. El nuevo sistema modificado recibe el nombre T.

- La reusabilidad y la modificabilidad son dos caras de una misma moneda
- Hacer un sistema que es modificable permite obtener los beneficios de la reutilización

Integrabilidad

- *Habilidad para hacer que piezas de software desarrolladas separadamente trabajen correctamente juntas.*
- La integrabilidad depende de:
  - complejidad de las componentes
  - mecanismos y protocolos de comunicación
  - claridad en la asignación de responsabilidades
  - calidad y completitud de la especificación de las interfaces

Todo depende de la arquitectura

- *La interoperabilidad es una forma de integrabilidad donde las partes del sistema deben trabajar con otro sistema.*

Testabilidad

- *Facilidad con la cual el software puede mostrar sus defectos (típicamente a través de pruebas de ejecución).*
- Probabilidad de que, suponiendo que el software tiene al menos un defecto, fallará en la siguiente prueba.
- Condiciones necesarias:
  - controlabilidad - poder controlar el estado interno de las componentes
  - observabilidad - poder observar las salidas (outputs)

Testabilidad y Arquitectura

- Inciden en la testabilidad:
  - nivel de documentación de la arquitectura
  - separación de intereses
  - uso de ocultamiento de información
  - desarrollo incremental

Atributos de Calidad del Negocio

- Características de costo y tiempo:
  - tiempo para poner en el mercado
  - costo
  - tiempo de vida útil proyectada para el sistema
- Características de comercialización:
  - mercado objetivo
  - planificación
  - uso intenso de sistemas legados

Costo y Tiempo

- Tiempo para poner en el mercado
  - poco tiempo para desarrollo presiona a la reutilización
  - el uso de COTS o componentes ya desarrolladas ayuda
  - el diseño de la nueva arquitectura debe ser capaz de albergar estos componentes.
- Costo
  - el costo de una arquitectura que usa tecnología conocida será menor
  - el aprendizaje de una nueva tecnología podrá reusarse en el futuro
- Vida útil planificada del sistema
  - en una larga vida útil, la modificabilidad y portabilidad son importantes, aunque esto alargue el ciclo de desarrollo

Comercialización

- Mercado objetivo
  - la plataforma y la funcionalidad determina el tamaño del mercado; funcionalidad y portabilidad son esenciales
  - performace, confiabilidad y usabilidad también son importantes
  - para un mercado grande pero específico, debe considerarse el enfoque de líneas de productos
- Planificación
  - si se construye un sistema básico para ser extendido, la flexibilidad y configurabilidad son importantes
- Uso de sistemas legados
  - si se interactúa con sistemas legados, debe proveerse un mecanismo adecuado de integración

Atributos de Calidad de la Arquitectura

- Integridad conceptual
  - visión unificada usada en el diseño del sistema
- Correctitud y completitud
  - el sistema incluye toda la funcionalidad requerida del sistema
- Factibilidad de la construcción
  - posibilidad de ser construido el sistema por el equipo de desarrollo disponible en el tiempo estipulado

Integridad Conceptual

- "Sostendré que la integridad conceptual es *la* consideración más importante en el diseño de un sistema. Es mejor tener un sistema sin ciertas características y optimizaciones, pero que refleje un conjunto único de ideas de diseño más que un sistema que tiene muchas ideas buenas pero independientes y descoordinadas."

[Brooks75]

- En realidad Brooks se refería al diseño de la interfaz usuaria, pero también vale para la arquitectura:
  - Lo que la integridad conceptual de la interfaz es para el usuario, lo es la integridad conceptual de la arquitectura para los demás stakeholders.

20 años más tarde: Estrategia

- "Estoy más convencido que nunca. La integridad conceptual es esencial para la calidad del producto. Tener un arquitecto de sistema es el paso más importante hacia tener integridad conceptual ... Después de dictar un laboratorio de ingeniería de software más de 20 veces, insisto que los grupos de desarrollo de 4 estudiantes deben elegir un administrador de proyecto y un arquitecto separados."

[Brooks95]

¿Cuál Arquitectura Escoger?

- ¿Qué estructura debo emplear para asignar trabajadores, para derivar la estructura de repartición del trabajo, para explotar las componentes ya empaquetadas y para planificar un sistema modificable?

Administración del proyecto

- ¿Qué estructura debo emplear para que el sistema, durante su ejecución, logre sus requisitos de comportamiento y sus atributos de calidad?

Estilos y patrones de arquitectura

Créditos

- **Software Architecture in Practice, Second Edition**. Len Bass, Paul Clements, Rick Kazman
- **Essential Architecture.** Ian Gorton.
- **Documenting Software Architectures, Views and Beyond. Second Edition.** Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, Judith Stafford.
- **Curso de Arquitectura de Software, Universidad de Chile.** Cecilia Bastarrica, Daniel Perovich.