

# Principios de Arquitectura Empresarial

Fabio Castro Rozo



# Contenido

1. Generalidad acerca de Calidad
2. Atributos de Calidad



# Generalidades acerca de Calidad



# Arquitectura de Software

## Atributos de Calidad



Hola Felipe, Maria de nuevo. Tengo una pregunta acerca del nuevo sistema de empleados que programaste. Como sabes, este sistema corre en nuestro mainframe y cada departamento tiene que pagar por su almacenamiento en disco y por el consumo de CPU cada mes. Al parecer el nuevo sistema de archivos está utilizando por lo menos el doble de disco que el anterior. Peor aún, los costos por CPU se han triplicado. ¿Puedes decirme que ha pasado?

Claro que si María, *dice Felipe*. Recuerda que requerías que el sistema almacenara mucho más información acerca de cada empleado respecto al sistema anterior, por lo tanto la base de datos es más grande. Y por lo tanto es natural que pague más por mes. También especificaste que requerías que el sistema fuera más sencillo de usar, por lo tanto le adicionamos una interfaz de usuario gráfica amigable. Sin embargo, la interfaz gráfica consume muchos más recursos de CPU que el anterior sistema, que era solo de caracteres. Por eso es que tu costo por sesión de usuario es más alto, pero el nuevo sistema es más fácil de usar que el anterior, ¿cierto?



# Arquitectura de Software

## Atributos de Calidad



Si, lo es, *replicó María*, pero la verdad no pensaba que fuera tan costoso de usar. Estoy en problemas por eso. Mi jefe se está poniendo nervioso, ya que a este paso todo nuestro presupuesto de cómputo se acabará en Abril. ¿Puedes **arreglar** el sistema para que cueste mucho menos?

*Felipe estaba frustrado:* No hay nada que arreglar, realmente. El nuevo sistema de empleados es exactamente lo que tu pediste que fuera. **Asumí** que si tu tenías claro que al almacenar más datos o trabajar más con el computador tus costos iban a subir. **Posiblemente debimos haber hablado de esto antes, porque en este momento no hay mucho que yo pueda hacer.** Lo siento.



# Arquitectura de Software

## Atributos de Calidad



Se enfocan en el comportamiento y en la funcionalidad (las cosas que el software debe hacer)



El éxito del software va mucho más allá de que ofrezca la funcionalidad correcta

# Arquitectura de Software

## Atributos de Calidad



Facilidad de uso

Qué tan rápido corre

Qué tanto falla

Cómo maneja condiciones inesperadas

Estas (y otras) características se conocen colectivamente como “atributos de calidad” (o factores de calidad) y forman parte de los requerimientos no funcionales (o no comportamentales)

# Arquitectura de Software

## Atributos de Calidad

### ¿Cómo clasificarlos?

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

<https://jegadeesansite.files.wordpress.com/2018/01/sei-series-in-software-engineering-len-bass-paul-clements-rick-kazman-software-architecture-in-practice-addison-wesley-professional-2012.pdf>

Distinguir entre aquellos que son observables mediante la ejecución, y entre aquellos que no lo son

OBSERVABLES EN EJECUCIÓN	NO OBSERVABLES EN EJECUCIÓN
¿Cómo se comporta el sistema durante la ejecución?	¿Cuán fácil es integrar el sistema, probarlo o modificarlo?
¿Entrega los resultados esperados?	¿Cuán caro ha sido su desarrollo?
¿Los entrega en un tiempo razonable?	¿Cuánto se tardó en tenerlo disponible en el mercado?
¿Tienen los resultados una precisión tolerable?	
¿Se comporta apropiadamente al interactuar con otros sistemas?	



# Arquitectura de Software

## Atributos de Calidad

### Clasificación según observabilidad en ejecución

Distinguir entre aquellos que son observables mediante la ejecución, y entre aquellos que no lo son

OBSERVABLES EN EJECUCIÓN	NO OBSERVABLES EN EJECUCIÓN
Performance	Modificabilidad
Seguridad	Portabilidad
Disponibilidad	Reusabilidad
Funcionalidad	Integrabiilidad
Usabilidad	Testabilidad

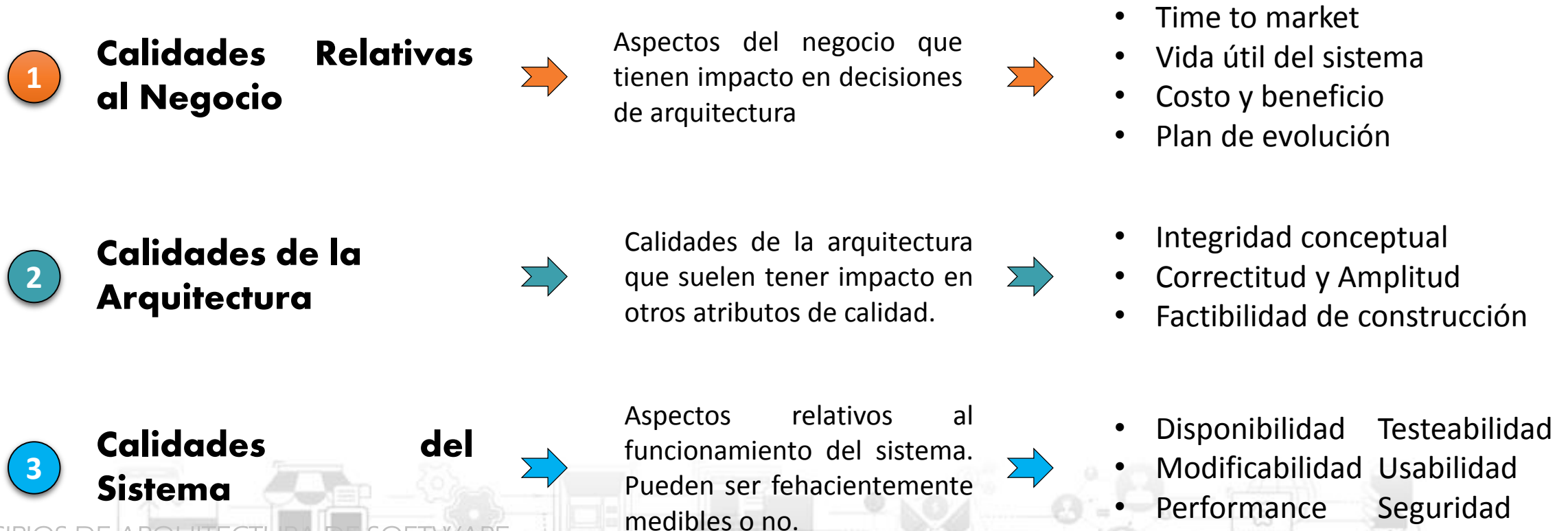


# Arquitectura de Software

## Atributos de Calidad

### ¿Hay otra forma de clasificarlos?

De acuerdo a la “Clase de Calidad” a la que pertenecen:



# Arquitectura de Software

## Atributos de Calidad

### Clasificación según Clase de Calidad



# Arquitectura de Software

## Atributos de Calidad



Cada Sistema exhibe el máximo valor posible para todos los atributos

El Sistema está disponible todo el tiempo

El Sistema nunca falla

El Sistema brinda resultados instantáneos, y siempre son correctos

El Sistema es intuitivamente fácil de utilizar



# Arquitectura de Software

## Atributos de Calidad



- Es necesario determinar cuales son los atributos más importantes para el éxito del proyecto en particular.
- Hay que entender que diferentes partes del producto necesitan diferentes combinaciones de atributos de calidad.
- La eficiencia puede ser crítica para algunos componentes, mientras la usabilidad es determinante para otros.



# Arquitectura de Software

## Conflictos entre Atributos de Calidad (Trade-Offs)

Algunas combinaciones de Atributos de Calidad inevitablemente generan conflictos.

Se deben evaluar cuantitativamente y/o cualitativamente el conjunto de atributos de calidad para determinar:

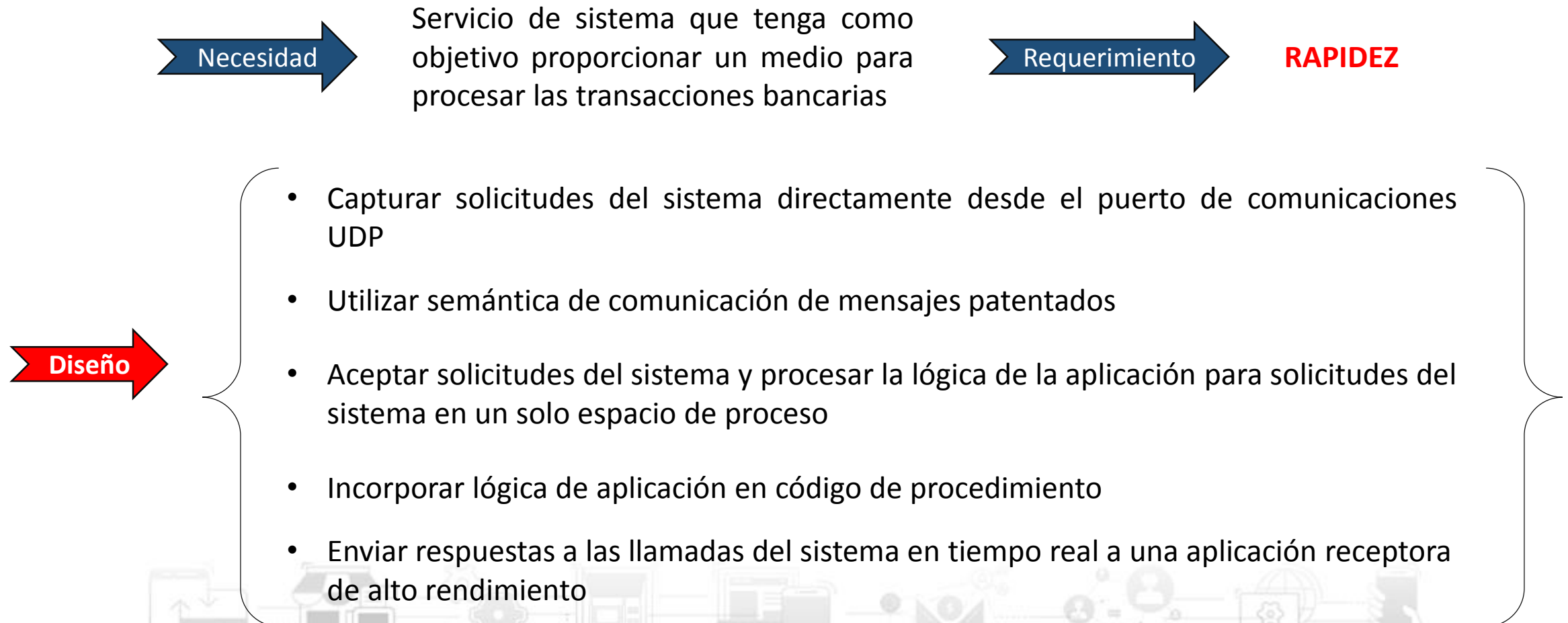
- Priorización de los atributos de calidad de acuerdo a los intereses involucrados
- El diseño de un sistema lo suficientemente bueno (y balanceado) para todos los interesados.

	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability								+		+		
Efficiency			-		-	-	-	-		-	-	-
Flexibility		-		-		+	+	+		+		
Integrity		-			-				-		-	-
Interoperability		-	+	-			+					
Maintainability	+	-	+					+			+	
Portability		-	+		+	-			+		+	-
Reliability	+	-	+			+				+	+	+
Reusability		-	+	-				-			+	
Robustness	+	-						+				+
Testability	+	-	+			+		+				+
Usability		-								+	-	

# Arquitectura de Software

## Conflictos entre Atributos de Calidad (Trade-Offs)

### Ejemplo



# Arquitectura de Software

## Conflictos entre Atributos de Calidad (Trade-Offs)

### Ejemplo



Servicio de sistema que tenga como objetivo proporcionar un medio para procesar las transacciones bancarias



**RAPIDEZ**



Puntos de Conflicto

- **Interoperabilidad.** Será difícil interoperar debido al protocolo de comunicación UDP no estructurado y poco confiable, así como a los formatos de mensajes no compatibles no basados en los estándares de la industria.
- **Flexibilidad.** La descomposición de la aplicación no se presta para ser cambiada y reutilizada para otros fines debido a los protocolos de propiedad, la mensajería acoplada y la lógica de la aplicación, así como el almacenamiento de memoria de propiedad.
- **Fiabilidad.** Los paquetes de comunicación UDP no son confiables ya que los paquetes podrían perderse, los datos en memoria no se conservan en el disco y podrían perderse si la memoria se libera a través de una falla de proceso, reinicio, etc.



# Arquitectura de Software

## Escenarios de Calidad

*Entre estímulo y respuesta, hay un espacio.*

*En ese espacio está nuestro poder para elegir nuestra respuesta. En nuestra respuesta yace nuestro crecimiento y nuestra libertad.*

Viktor E. Frankl, Man 's Search for Meaning

- Un requerimiento de Atributo de Calidad debe ser inequívoco y comprobable.
- Se debe utilizar una forma común para especificar todos los requisitos de atributos de calidad.
- Tiene la ventaja de enfatizar los puntos en común entre todos atributos de calidad

### Entonces ... ¿Qué es un Escenario de Calidad?

Mecanismo que permite caracterizar/capturar aspectos de atributos de calidad de una forma que puede ser evaluado y utilizado en diseño

### ¿Qué tipos de Escenarios de Calidad existen?

	GENERALES	CONCRETOS
<b>Definición</b>	Son independientes del sistema y, potencialmente pueden pertenecer a cualquier sistema.	Aquellos que son específicos al sistema en consideración.
<b>Ejemplo</b>	Llega un requerimiento para hacer un cambio en la funcionalidad, y el cambio debe ser hecho en un momento particular dentro del proceso de desarrollo y dentro de un periodo de tiempo específico.	Llega un requerimiento para agregar soporte a un nuevo browser para un sistema web y el cambio debe ser hecho en, como máximo, 2 semanas.

# Arquitectura de Software

## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

<https://jegadeesansite.files.wordpress.com/2018/01/sei-series-in-software-engineering-len-bass-paul-clements-rick-kazman-software-architecture-in-practice-addison-wesley-professional-2012.pdf>

### ¿Cuáles son los componentes de un Escenario de Calidad?

COMPONENTE	DESCRIPCIÓN
FUENTE DE ESTÍMULO	Entidad que generó el estímulo (un ser humano, un sistema, u otro)
ESTÍMULO	El estímulo es una condición que requiere una respuesta cuando llega a un sistema.
AMBIENTE	El estímulo ocurre bajo ciertas condiciones. El sistema puede estar en una condición de sobrecarga o en operación normal, o en algún otro estado relevante. Para muchos sistemas, la operación "normal" puede referirse a uno de varios modos. Para este tipo de sistemas, el entorno debe especificar en qué modo se está ejecutando el sistema.
ARTEFACTO	Se estimula algún artefacto. Esto puede ser una colección de sistemas, todo el sistema, o una parte o partes de él.
RESPUESTA	La respuesta es la actividad realizada como resultado de la llegada del estímulo.
MEDIDA DE LA RESPUESTA	Cuando se produce la respuesta, debe ser medible de alguna manera para que el requisito pueda ser probado.

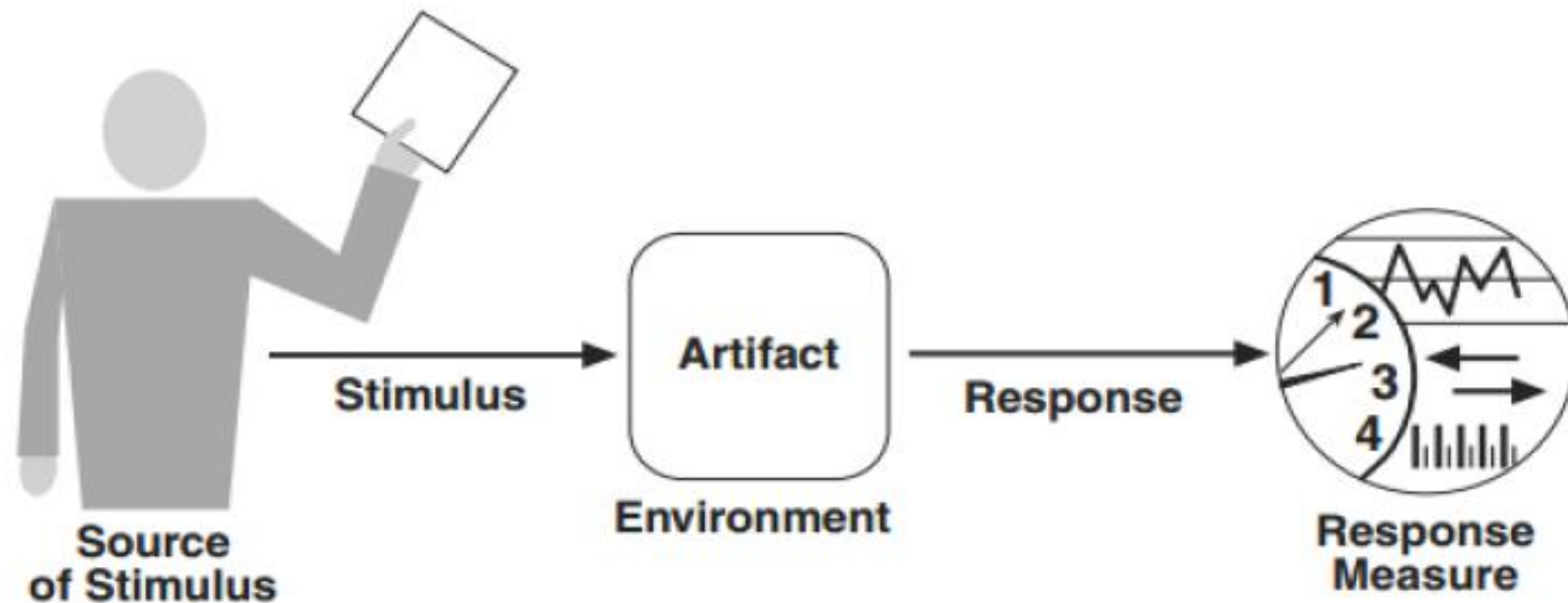
# Arquitectura de Software

## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

<https://jegadeesansite.files.wordpress.com/2018/01/sei-series-in-software-engineering-len-bass-paul-clements-rick-kazman-software-architecture-in-practice-addison-wesley-professional-2012.pdf>

### Componentes de un Escenario de Calidad



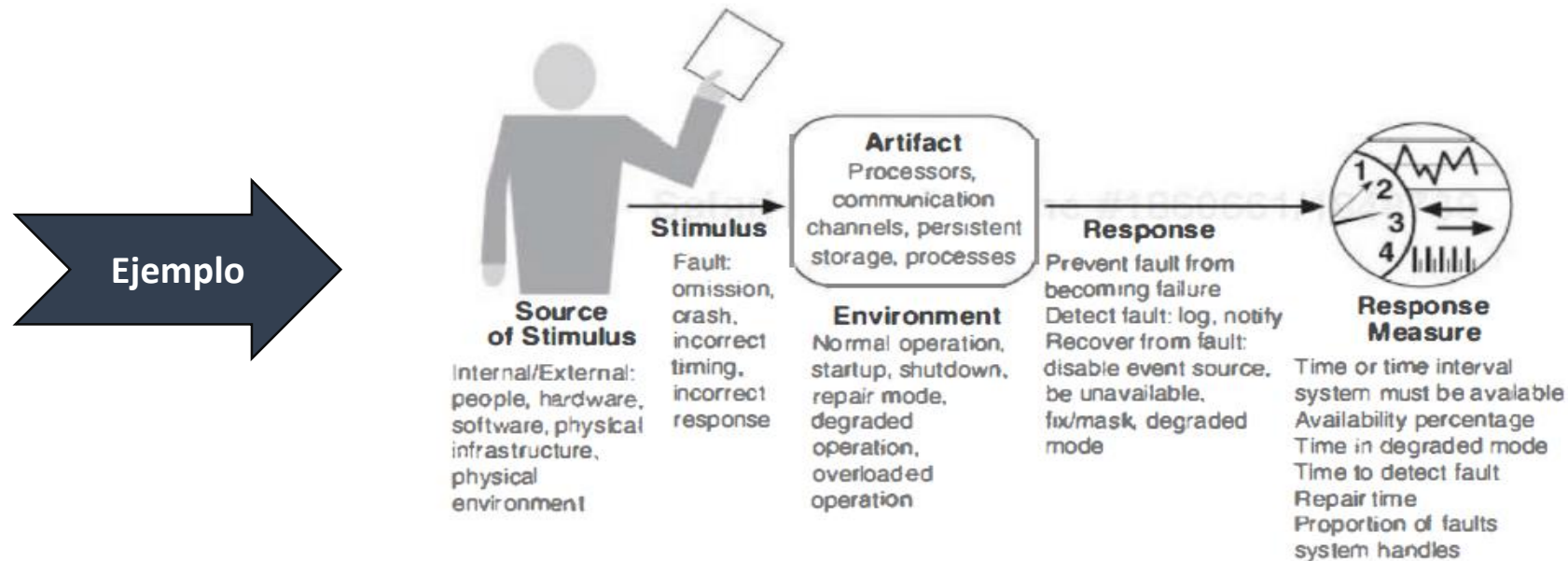
# Arquitectura de Software

## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

<https://jegadeesansite.files.wordpress.com/2018/01/sei-series-in-software-engineering-len-bass-paul-clements-rick-kazman-software-architecture-in-practice-addison-wesley-professional-2012.pdf>

- Se pueden caracterizar los atributos de calidad como una colección de escenarios generales.
- Para traducir el atributo de calidad en requisitos para un sistema en particular, los escenarios generales deben traducirse en concretos.



**Figure 4.2. A general scenario for availability**

# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Lograr Atributos de Calidad a través de Tácticas

El Arquitecto requiere técnicas para lograr alcanzar los atributos de calidad requeridos.  
Estas técnicas se conocen como “tácticas”

### Pero entonces... ¿Qué es una táctica?

Una táctica es una **decisión de diseño** que influencia el control de la respuesta de un atributo de calidad

### Consideraciones:

- 1 El enfoque de una táctica está en una respuesta de un atributo de calidad único. No se preocupa de los *trade-offs* entre atributos.
- 2 Las tácticas están destinadas a controlar las respuestas a los estímulos.
- 3 Las tácticas pueden y deben ser refinadas.
- 4 La aplicación de una táctica depende del contexto

# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

La táctica representa la relación entre estímulo y respuesta. Actúa como controlador de la respuesta.



# Atributos de Calidad



# Arquitectura de Software

## Atributos de Calidad

### Performance

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*).

#### ¿De qué depende?

Dependiente de la Arquitectura	No dependiente de la Arquitectura
Comunicación entre los componentes	Algoritmos que implementan la funcionalidad
Asignación de funcionalidades a los componentes	Codificación de los algoritmos
Asignación de los recursos compartidos	



# Arquitectura de Software

## Atributos de Calidad

### Performance

#### Métricas

#### 1 **Tiempo de Respuesta**

Medida de la latencia que una aplicación muestra en una transacción de negocio.

Frecuentemente (pero no exclusivamente) asociado con el tiempo que una aplicación toma en responder a una entrada

##### Tiempo de Respuesta Garantizado

Cada una de las peticiones deben ser atendida dentro de un límite de tiempo específico.

##### 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.

# Arquitectura de Software

## Atributos de Calidad

### Performance

#### Métricas

#### 2 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)



# Arquitectura de Software

## Atributos de Calidad

### Performance

#### Métricas

#### 3 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 comúnmente asociados a sistemas en lote (batch-systems)

# Arquitectura de Software

## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Performance

### Ejemplo de Escenario General

COMPONENTE	DESCRIPCIÓN
FUENTE DE ESTÍMULO	Una de un número independiente de fuentes posibles
ESTÍMULO	<ul style="list-style-type: none"> <li>• Eventos periódicos arriban</li> <li>• Eventos estocásticos arriban</li> <li>• Eventos esporádicos arriban</li> </ul>
AMBIENTE	<ul style="list-style-type: none"> <li>• Modo normal</li> <li>• Modo sobrecargado</li> <li>• Modo emergencia</li> </ul>
ARTEFACTO	El sistema, uno de sus componentes, un conjunto de componentes
RESPUESTA	<ul style="list-style-type: none"> <li>• Procesar el estímulo</li> <li>• Cambiar el nivel de servicio</li> </ul>
MEDIDA DE LA RESPUESTA	<ul style="list-style-type: none"> <li>• Latencia</li> <li>• Deadline</li> <li>• Throughput</li> <li>• Varianza de latencia</li> <li>• Tasa de pérdida de requerimientos</li> <li>• Tasa de pérdida de datos</li> </ul>

# Arquitectura de Software

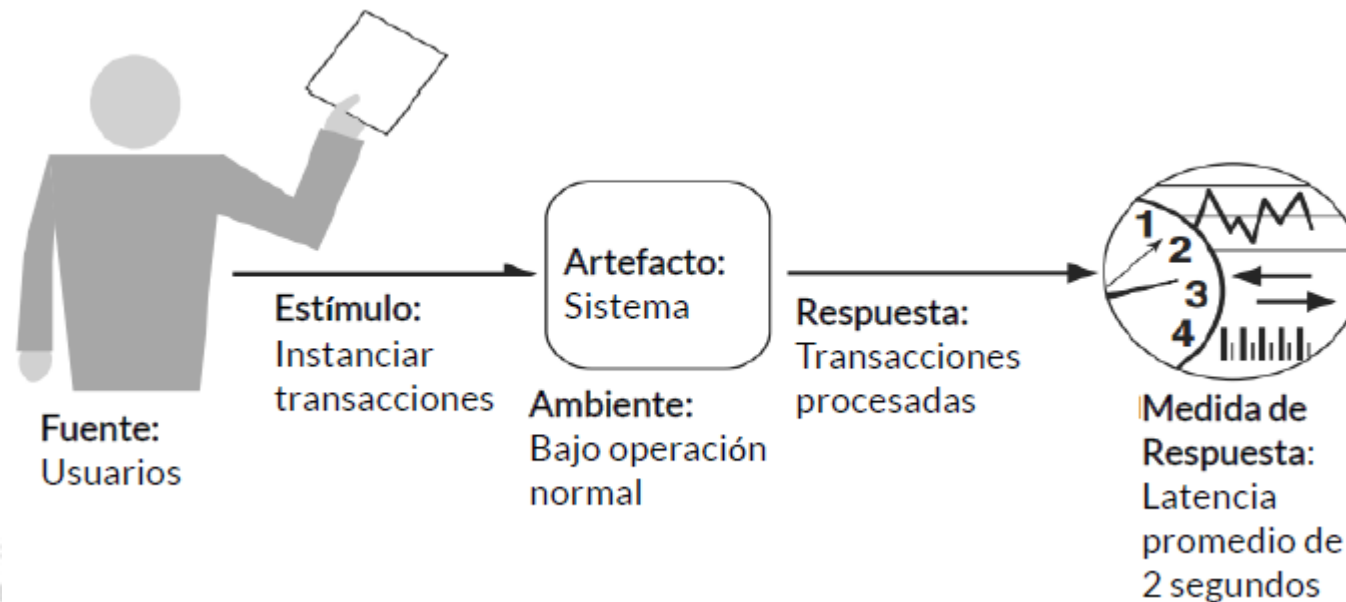
## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Performance

### Ejemplo de Escenario Concreto

Los usuarios inician transacciones en operaciones normales. El sistema procesa las transacciones con una latencia promedio de dos segundos.



# Arquitectura de Software

## Tácticas de Calidad

### Performance

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

- El objetivo de las tácticas de rendimiento (performance) es generar una respuesta a un evento que llega al sistema dentro de un límite de tiempo. El evento puede ser único o continuo y es el desencadenante para realizar el cálculo.
- Las tácticas de rendimiento controlan el tiempo dentro del cual se genera una respuesta.



# Arquitectura de Software

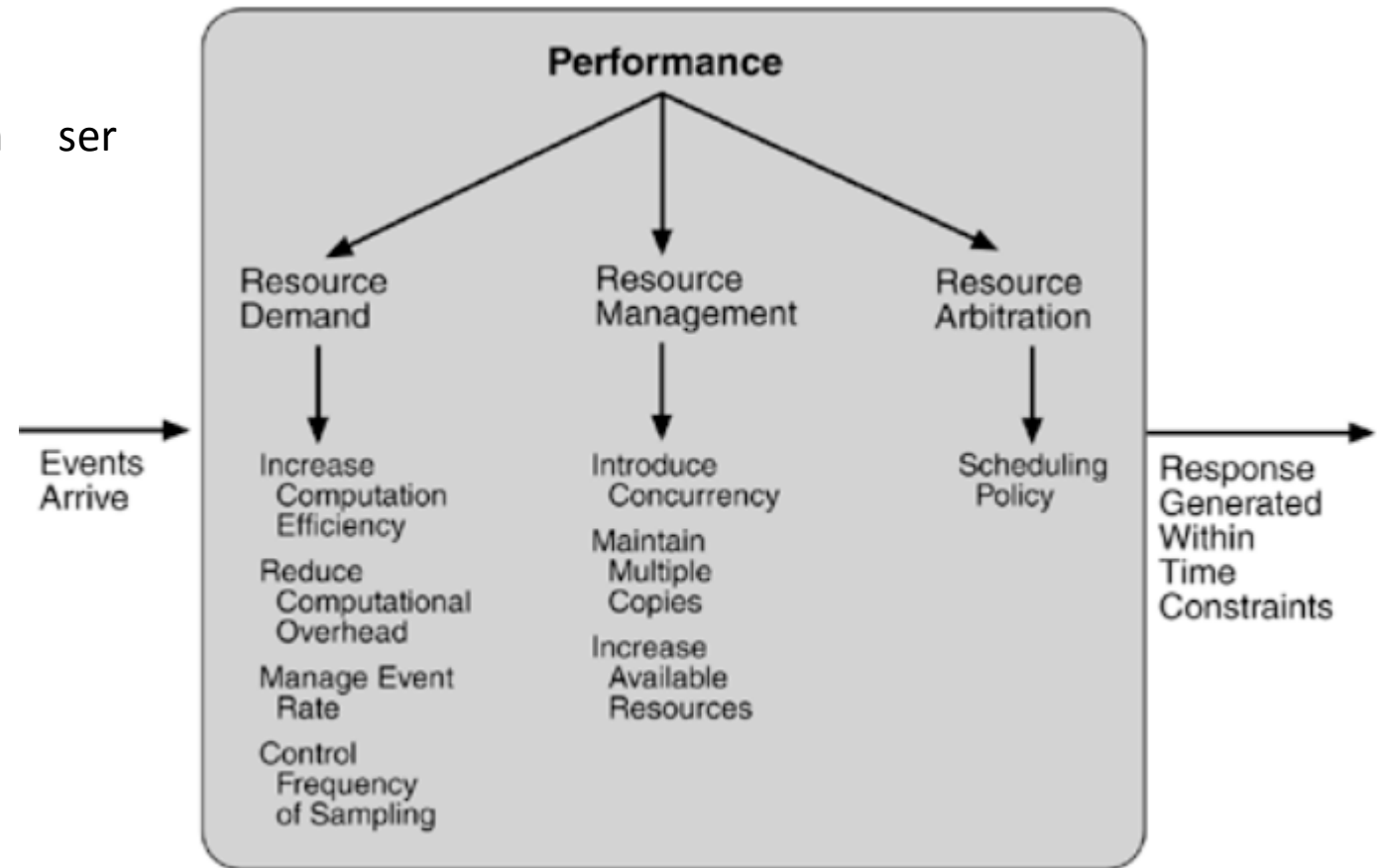
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Performance

Las tácticas de Performance pueden ser clasificadas en tres categorías:

- 1 Tácticas para **Demanda de Recursos**
- 2 Tácticas para **Administración de Recursos**
- 3 Tácticas para **Arbitraje de Recursos**

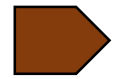


# Arquitectura de Software

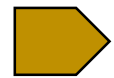
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

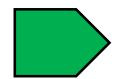
### Performance – Demanda de Recursos



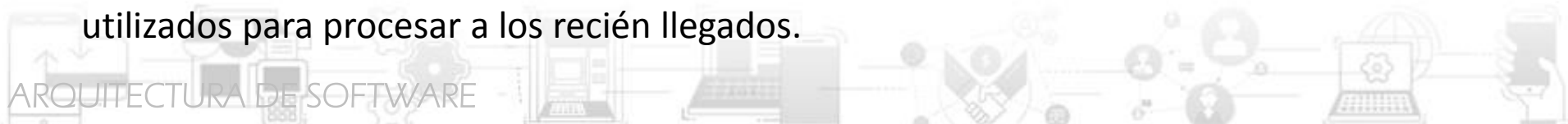
**Reducir la cantidad de recursos involucrados:** Aumentar la eficiencia computacional: Un paso en el procesamiento de un evento o un mensaje es la aplicación de algún algoritmo. La mejora de los algoritmos utilizados en las áreas críticas reducirá la latencia.



**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 ‘sobredimensionado’



**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. *Limitar el tamaño 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.



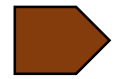


# Arquitectura de Software

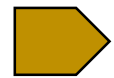
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

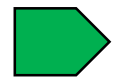
### Performance – Administración de Recursos



**Introducir Concurrency:** 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.



**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. Almacenamiento en caché



**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.



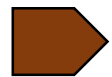
# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

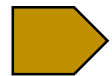
## Tácticas de Calidad

### Performance – Arbitramento de Recursos

- Cada vez que hay disputas para un recurso, el recurso debe planificarse. Los procesadores, búferes, y redes están planificados.
- Su objetivo es entender las características del uso de cada recurso y elegir la estrategia de planificación que sea compatible con él. (FIFO, Round Robin, Importancia semántica, Importancia por deadline).



**Asignación de prioridades**



**Política de despachos**



# Arquitectura de Software

## Atributos de Calidad

### 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 (o legítimos)

#### ¿Qué tipos de ataques pueden ocurrir?

**1**

#### **Acceder a datos/servicios a los que el usuario no está autorizado**

*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.*

**2**

#### **Limitar el servicio a usuarios autorizados (o legítimos)**

*Impedir que un servidor pueda dar servicios a sus usuarios. Se hace inundándolo con requerimientos o consultas.*



# Arquitectura de Software

## Atributos de Calidad

### Seguridad

#### Requerimientos

Requerimiento	Descripción
No repudio	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.
Disponibilidad	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	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.

# Arquitectura de Software

## Atributos de Calidad

### Seguridad

#### Estrategias de la Arquitectura de Software

Sirven para prevenir, detectar y responder a ataques de seguridad:

---

Disponer un servidor de autenticación 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

---

Implican identificar componentes especializados y coordinar su funcionamiento con las demás componentes.



# Arquitectura de Software

## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Seguridad

### Ejemplo de Escenario General

COMPONENTE	DESCRIPCIÓN
FUENTE DE ESTÍMULO	<ul style="list-style-type: none"> <li>Individuo o sistema que está...correctamente identificado, incorrectamente identificado, o es de identidad desconocida</li> <li>que es...interno/externo, autorizado/no autorizado</li> <li>con acceso a...recursos limitados, vastos recursos</li> </ul>
ESTÍMULO	Se realiza un intento no autorizado para mostrar datos, cambiar o eliminar datos, acceder a los servicios del sistema, cambiar el comportamiento del sistema o reducir la disponibilidad.
AMBIENTE	Online u offline; conectado o desconectado; detrás de un firewall o abierto, operativo o no.
ARTEFACTO	Servicios del sistema, datos dentro del sistema, un componente o recursos del sistema, datos producidos o consumidos por el sistema
RESPUESTA	Autentica al usuario; oculta la identidad del usuario; bloquea/permite accesos a datos y/o servicios; garantiza o rechaza permisos para acceder a datos y/o servicios; registra los accesos por usuario; almacena datos en un formato no legible; reconoce un inexplicable aumento de demanda del servicio, informando a un usuario u otro sistema y restringiendo la disponibilidad del servicio.
MEDIDA DE LA RESPUESTA	Uno o más de los siguientes: • Cuánto de un sistema está comprometido cuando un componente en particular o un valor de datos está comprometido • Cuánto tiempo pasó antes de que se detectara un ataque • ¿Cuántos ataques fueron resistidos? • ¿Cuánto tiempo lleva recuperarse de un ataque exitoso? • Cuántos datos son vulnerables a un ataque en particular.

# Arquitectura de Software

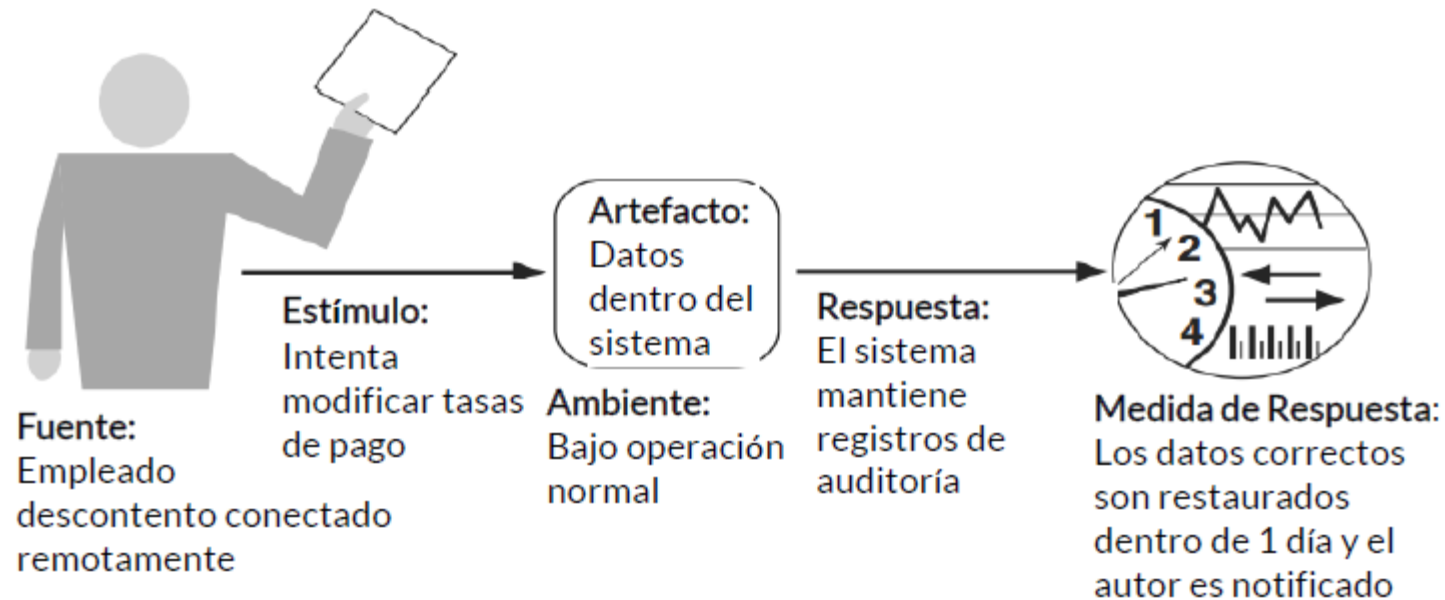
## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Seguridad

### Ejemplo de Escenario Concreto

Un empleado descontento desde una ubicación remota intenta acceder a la tabla de tasas de pago durante la operación normal. El sistema mantiene un registro de auditoría y los datos correctos son restaurados dentro de 1 día.



# Arquitectura de Software

## Tácticas de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Seguridad

- Estos conducen a cuatro categorías de tácticas: detectar, resistir, reaccionar y recuperarse.
- Un método para pensar en cómo lograr la seguridad en un sistema es pensar en la seguridad física. Las instalaciones seguras tienen acceso limitado (por ejemplo, mediante el uso de puntos de control de seguridad), tienen medios para detectar intrusos (por ejemplo, al exigir a los visitantes legítimos que usen insignias), tienen mecanismos de disuasión como guardias armados, tienen mecanismos de reacción como el bloqueo automático de puertas y tener mecanismos de recuperación como copia de seguridad fuera del sitio.





# Arquitectura de Software

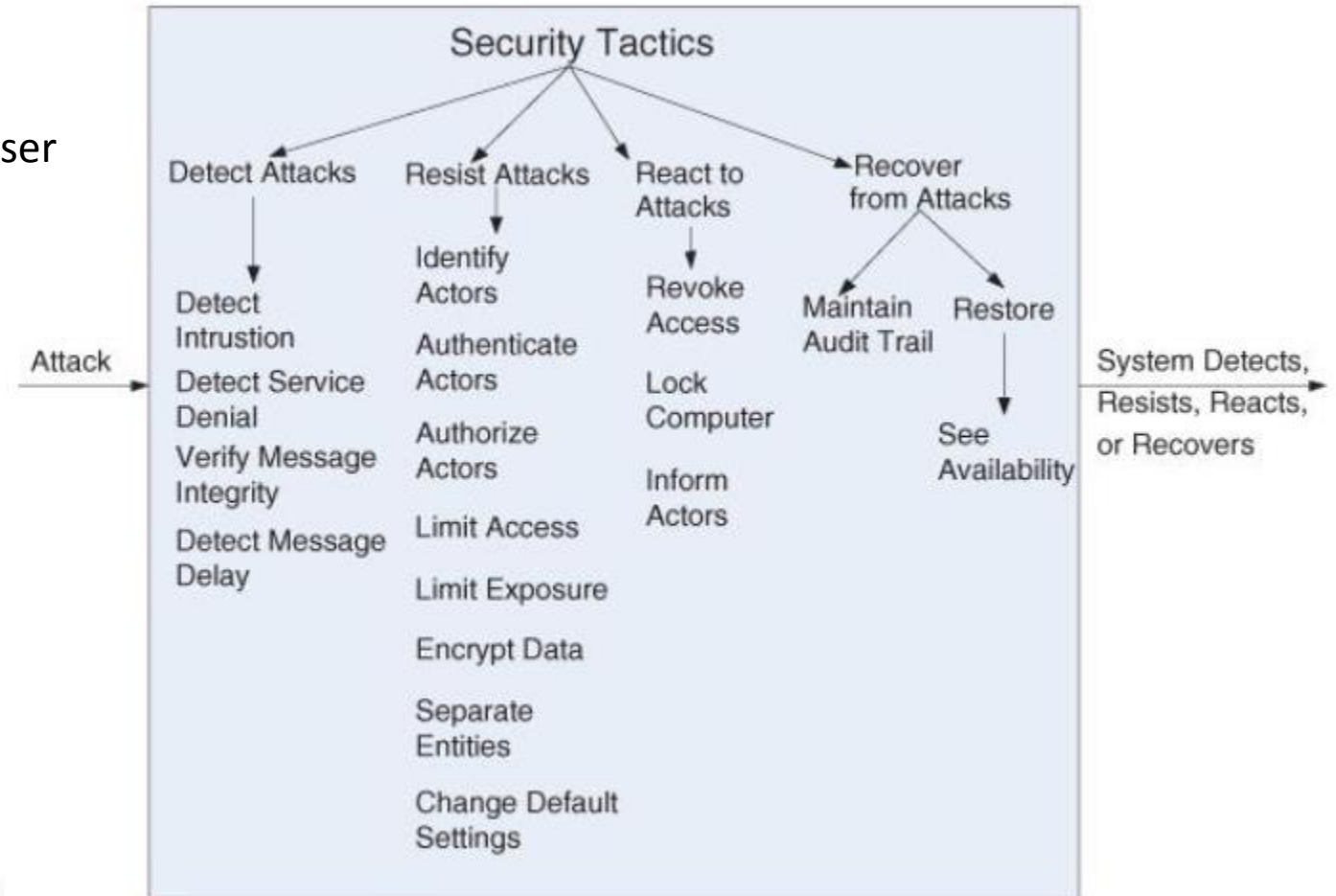
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Seguridad

Las tácticas de Seguridad pueden ser clasificadas en cuatro categorías:

- 1 Tácticas para **Detectar**
- 2 Tácticas para **Resistir**
- 3 Tácticas para **Reaccionar**
- 4 Tácticas para **Recuperar**

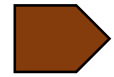


# Arquitectura de Software

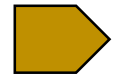
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

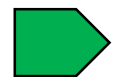
### Seguridad – Detectar



**Detectar Intrusión:** Se basa en la comparación del tráfico de la red o los patrones de solicitud de servicio dentro de un sistema con un conjunto de firmas o patrones conocidos de comportamiento malicioso almacenados en una base de datos. Ejemplo: Listas negras / blancas de direcciones IP que pueden acceder.



**Detectar Negación del Servicio:** Se basa en la comparación del patrón o la firma del tráfico de red que entra en un sistema con los perfiles históricos de ataques conocidos de denegación de servicio.



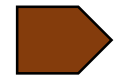
**Verificar Integridad de Mensajes:** Esta táctica emplea técnicas tales como sumas de comprobación o valores de hash para verificar la integridad de los mensajes, archivos de recursos, archivos de implementación y archivos de configuración.



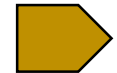
# Arquitectura de Software

## Tácticas de Calidad

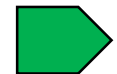
### Seguridad – Resistir



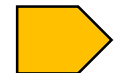
**Identificar Actores:** La identificación de "actores" consiste realmente en identificar la fuente de cualquier aporte externo al sistema. Los usuarios normalmente se identifican a través de ID de usuario. Otros sistemas pueden ser "identificados" a través de códigos de acceso, direcciones IP, protocolos, puertos, etc.



**Autenticar Actores:** La autenticación significa asegurarse que un actor (un usuario o una computadora remota) es realmente quien eso lo que pretende ser. Las contraseñas únicas, los certificados digitales y la identificación biométrica proporcionan un medio para la autenticación



**Autorizar Actores:** La autorización significa garantizar que un actor autenticado tiene los derechos de acceso y de modificación de datos o servicios.



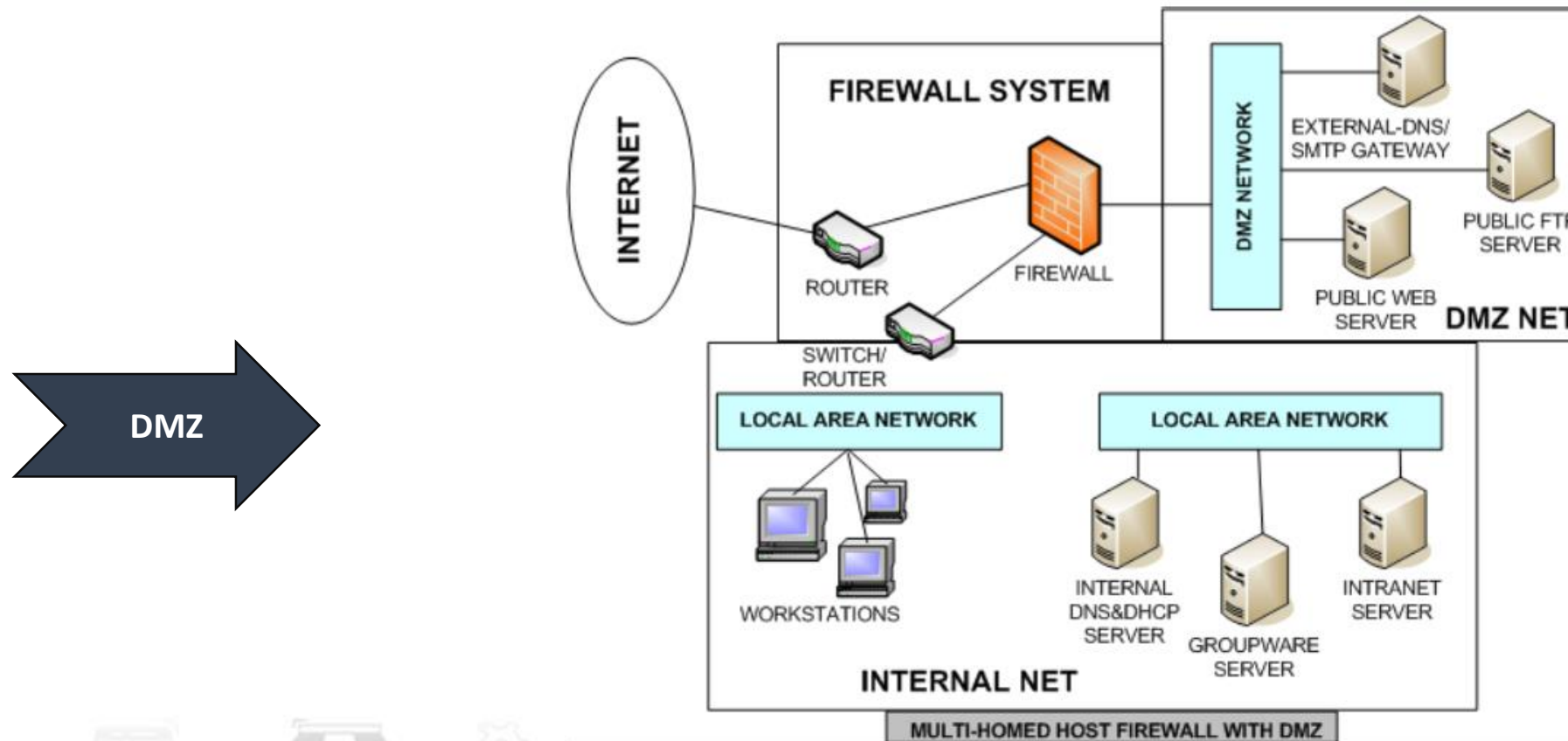
**Acceso Limitado:** Limitar el acceso a los recursos informáticos implica limitar el acceso a recursos como la memoria, conexiones de red o puntos de acceso. Por ejemplo, una zona desmilitarizada (DMZ) se utiliza cuando una organización quiere permitir que los usuarios externos accedan a ciertos servicios y no accedan a otros servicios. Se encuentra entre Internet y un cortafuegos frente a la intranet interna.

# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Seguridad – Resistir

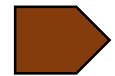


# Arquitectura de Software

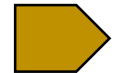
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

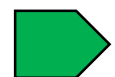
### Seguridad – Reaccionar



**Revocar Acceso:** Si el sistema o el administrador del sistema cree que un ataque está en curso, el acceso puede estar severamente limitado a recursos sensibles, incluso para usuarios y usos normalmente legítimos.



**Bloquear Computador:** Los intentos fallidos de inicio de sesión fallidos pueden indicar un posible ataque. Muchos sistemas limitan el acceso desde una computadora en particular si hay repetidos intentos fallidos de acceder a una cuenta desde esa computadora. Los usuarios legítimos pueden cometer errores al intentar iniciar sesión. Por lo tanto, el acceso limitado sólo puede ser durante un cierto período de tiempo.



**Informar Actores:** Los ataques en curso pueden requerir la acción de operadores, otro personal o sistemas cooperantes. Tal personal o sistemas -el conjunto de actores relevantes-debe ser notificado cuando el sistema ha detectado un ataque.

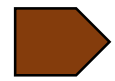


# Arquitectura de Software

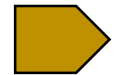
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Seguridad – Recuperación



**Mantener rastros de Auditoría:** Se audita, es decir se mantiene un registro de las acciones del usuario y del sistema y sus efectos para ayudar a rastrear las acciones de un atacante y para identificarlo.



**Recuperación:** Parte de la recuperación es la restauración de los servicios. Por ejemplo, servidores adicionales o conexiones de red pueden mantenerse en reserva para tal propósito.

Puesto que un ataque exitoso puede ser considerado como una especie de fracaso, el conjunto de tácticas de disponibilidad que se ocupan de la recuperación de un fracaso puede ser llevado a cabo para este aspecto de la seguridad también



# Arquitectura de Software

## Atributos de Calidad

### Disponibilidad

Es una medida del tiempo de actividad planificado durante el cual el sistema está realmente disponible para su uso y es totalmente operativo

**MTTF**



Mean Time To Failure (Tiempo Medio Entre Fallas)

**MTTR**



Mean Time To Repair (Tiempo Medio De Reparación)

$$\text{Disponibilidad} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Los requerimientos de disponibilidad se hacen más complejos y más importantes para Web Sites o para aplicaciones globales con usuarios mundiales.



*El Sistema debe estar al menos 99.5% disponible en días laborales, entre las 6:00 am y la medianoche, y al menos 99.95% disponible en días laborales entre las 4:00 pm y las 6:00 pm*



# Arquitectura de Software

## Atributos de Calidad

### Disponibilidad

Otros ejemplos de Requerimientos de Disponibilidad:

Availability	Downtime/90 Days	Downtime/Year
99.0%	21 hours, 36 minutes	3 days, 15.6 hours
99.9%	2 hours, 10 minutes	8 hours, 0 minutes, 46 seconds
99.99%	12 minutes, 58 seconds	52 minutes, 34 seconds
99.999%	1 minute, 18 seconds	5 minutes, 15 seconds
99.9999%	8 seconds	32 seconds





# Arquitectura de Software

## Atributos de Calidad

### Disponibilidad

#### Estrategias de la Arquitectura de Software

---

Las arquitecturas redundantes o con alta recuperabilidad son utilizadas para obtener alta disponibilidad

---

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.

---

Cloud Computing con esquemas de escalamiento vertical y horizontal automático de acuerdo a métricas.

---

# Arquitectura de Software

## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Disponibilidad

### Ejemplo de Escenario General

COMPONENTE	DESCRIPCIÓN
FUENTE DE ESTÍMULO	Interna al Sistema; externa al sistema
ESTÍMULO	Defecto: omisión, crash, timing incorrecto, respuesta incorrecta
AMBIENTE	Operación normal, Inicio del sistema, Modo de reparación Operación degradada (p ej. funcionalidad limitada)
ARTEFACTO	Procesadores del sistema, canales de comunicación, almacenamiento persistente, procesos
RESPUESTA	<p>El sistema detectaría un evento y haría una o varias de las siguientes acciones:</p> <ul style="list-style-type: none"> <li>• Registrarlo</li> <li>• Notificar a quienes corresponda, incluyendo el usuario y otros sistemas</li> <li>• Deshabilitar las fuentes de eventos que causan el defecto o falla de acuerdo a las reglas definidas</li> <li>• Quedar no disponible por un intervalo de tiempo, hasta que se repare</li> <li>• Continuar la operación en modo degradado mientras se repara.</li> </ul>
MEDIDA DE LA RESPUESTA	<ul style="list-style-type: none"> <li>• Intervalo de tiempo en el que el sistema debe estar disponible</li> <li>• Porcentaje de disponibilidad</li> <li>• Tiempo para detectar la falla</li> <li>• Tiempo para reparar la falla</li> <li>• Tiempo que el sistema puede funcionar en modo degradado</li> </ul>

# Arquitectura de Software

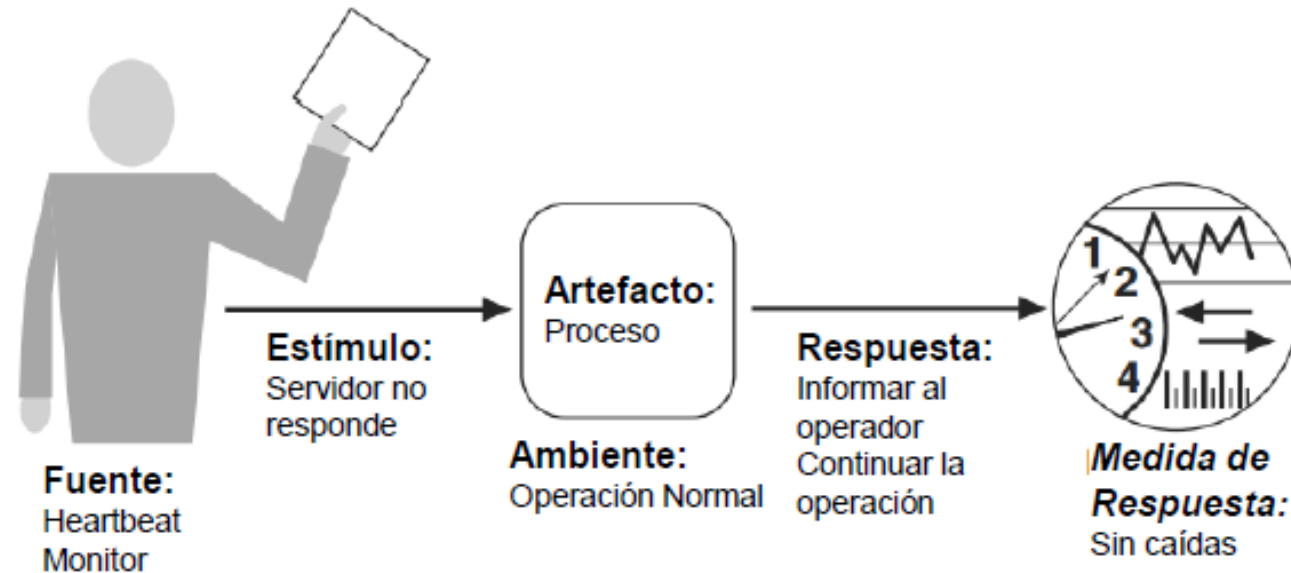
## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Disponibilidad

### Ejemplo de Escenario Concreto

“El monitor cardiaco determina que el servidor no responde ante operaciones normales.  
El sistema le informa al operador y continua operando sin tiempo de caídas.”



# Arquitectura de Software

## Tácticas de Calidad

### Disponibilidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

- Se produce una falla cuando el sistema ya no brinda un servicio que sea consistente con su especificación; Este fallo es observable por los actores del sistema.
- Una falla (o combinación de fallas) tiene el potencial de causar una falla.
- Las tácticas de disponibilidad, por lo tanto, están diseñadas para permitir que un sistema resista fallas del sistema de modo que un servicio entregado por el sistema siga cumpliendo con sus especificaciones.



# Arquitectura de Software

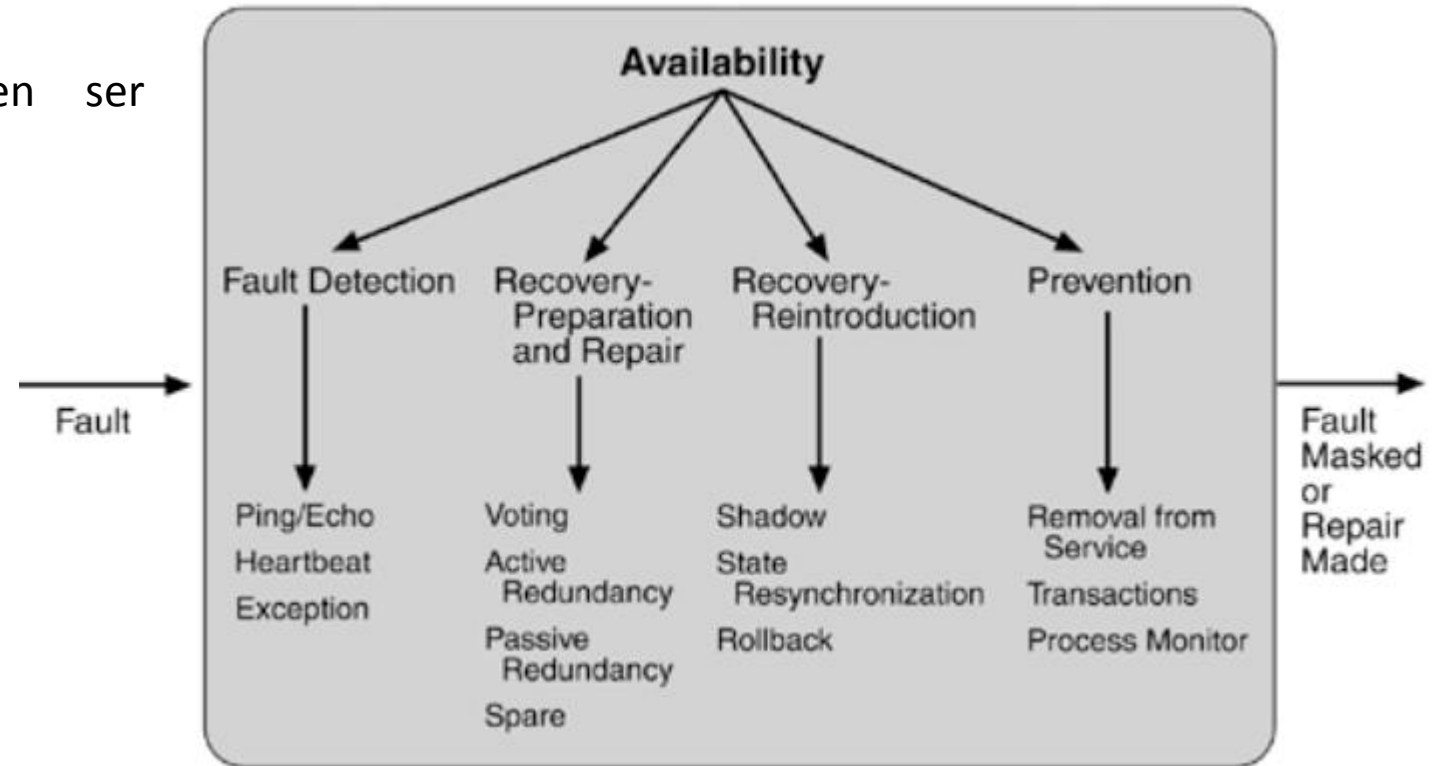
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Disponibilidad

Las tácticas de Disponibilidad pueden ser clasificadas en tres categorías:

- 1 Tácticas para **Detección de fallas**
- 2 Tácticas para **Recuperación de fallas**
- 3 Tácticas para **Prevención de fallas**



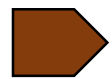
# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

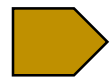
## Tácticas de Calidad

### Disponibilidad – Detección de Fallas

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



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



**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.



# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Disponibilidad – Recuperación de Fallas

Consiste en la preparación para la recuperación y hacer la reparación del sistema

- Votación
- Redundancia activa
- Redundancia pasiva
- Spare (reposición)
- Operación en la sombra
- Re-sincronización de estado
- Checkpoint/rollback.



# Arquitectura de Software


*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad


### Disponibilidad – 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.

 **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.

 **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.





# Arquitectura de Software

## Atributos de Calidad

### Usabilidad

Es la facilidad con la cual las personas pueden utilizar un sistema con el fin de alcanzar un objetivo concreto

#### Usabilidad y Arquitectura

1

**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

2

**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.



## Atributos de Calidad

## ¿De qué depende?



PRINCIPIOS DE ARQUITECTURA DE SOFTWARE

# Arquitectura de Software

## Atributos de Calidad

### Usabilidad

#### Requerimientos

Requerimiento	Descripción
Aprendizaje de Características del Sistema	¿Qué puede hacer el sistema para que a un usuario que no está familiarizado con él pueda aprender a usarlo más fácilmente y más rápidamente?
Uso eficiente del sistema	¿Qué puede hacer el sistema para mejorar la eficiencia (tiempo de respuesta) en la operatoria de un usuario?
Minimizar el impacto de los errores	¿Qué puede hacer el sistema para minimizar el impacto del error de un usuario? ¿Está en la capacidad de anticiparlos y prevenirlos?
Adaptación del sistema a las necesidades del usuario	¿Cómo puede el usuario (o el sistema en si mismo) adaptarse para hacer la tarea más fácil?
Incrementar la confianza y la satisfacción	¿Qué hace el sistema para darle confianza al usuario que está ejecutando la acción correcta?
Recordabilidad	¿Pueden los usuarios recordar cómo usar el sistema entre dos sesiones de uso?

# Arquitectura de Software

## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Usabilidad

### Ejemplo de Escenario General

COMPONENTE	DESCRIPCIÓN
FUENTE DE ESTÍMULO	Usuario Final
ESTÍMULO	Quiere...aprender las características del sistema; usar el sistema eficientemente; minimizar el impacto de errores de usuario; adaptar el sistema a sus necesidades; configurar el sistema
AMBIENTE	<ul style="list-style-type: none"> <li>En tiempo de ejecución</li> <li>En tiempo de configuración</li> </ul>
ARTEFACTO	Sistema
RESPUESTA	<ul style="list-style-type: none"> <li>Para soportar el aprendizaje de las características del sistema</li> </ul> Sistema de ayuda sensible al contexto; UI familiar para el usuario <ul style="list-style-type: none"> <li>Para soportar el uso del sistema eficientemente</li> </ul> Agregación de datos/comandos; reuso de datos y comandos ya ingresados; soporte para una navegación eficiente; distintas vistas con operaciones consistentes; <ul style="list-style-type: none"> <li>Para minimizar el impacto de errores de usuario</li> </ul> Deshacer, rehacer y recuperarse de una falla del sistema; reconocer y corregir un error de usuario; recuperar password olvidada <ul style="list-style-type: none"> <li>Para adaptar el sistema</li> </ul> Personalización; Internacionalización <ul style="list-style-type: none"> <li>Para sentirse cómodo</li> </ul> Mostrar el estado del sistema; trabajar al ritmo del usuario
MEDIDA DE LA RESPUESTA	Tiempo de la tarea; cantidad de errores; cantidad de problemas resueltos; satisfacción del usuario; tasa de operaciones exitosas vs operaciones no exitosas; interacción necesaria para completar una tarea

# Arquitectura de Software

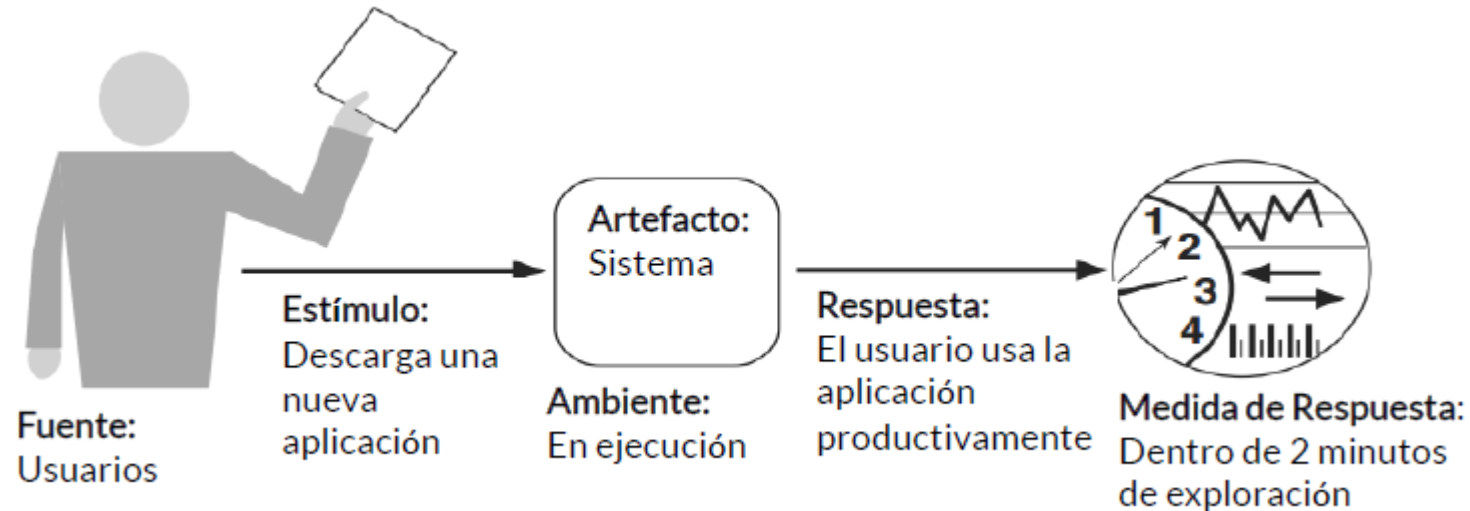
## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Usabilidad

### Ejemplo de Escenario Concreto

Un usuario descarga una nueva aplicación y está usándola productivamente luego de 2 minutos de experimentación.



# Arquitectura de Software

## Atributos de Calidad

### Modificabilidad

- Habilidad para hacer cambios al sistema de una forma *rápida y poco costosa*.
- Es el costo (riesgo) asociado a realizar un cambio en un sistema.

#### ¿Qué puede cambiar?

- |   |                                       |   |                                |
|---|---------------------------------------|---|--------------------------------|
| 1 | Funcionalidad                         | 3 | Ambiente (Entorno)             |
| 2 | Plataforma (HW, SO, Middleware, etc.) | 4 | Capacidad (Número de usuarios) |

#### ¿Cuándo se hace el cambio?

- |   |                          |   |               |
|---|--------------------------|---|---------------|
| 1 | Codificación             | 3 | Configuración |
| 2 | Compilación              | 4 | Ejecución     |
| 5 | Construcción del paquete |   |               |



# Arquitectura de Software

## Atributos de Calidad

### Modificabilidad

**¿En qué etapas de puede hacer?**

- 1 Localizar el (los) lugar(es) dónde debe aplicarse el cambio
- 2 Aplicar el cambio propiamente dicho

**Desde el desarrollo es deseable que un sistema sea mantenible, ya que pasa por múltiples etapas:**

- 1 Control de versiones
- 2 Control de configuración
- 3 Despliegue



# Arquitectura de Software

## Atributos de Calidad

### Modificabilidad

#### ¿Qué determina la necesidad de cambio?

Normalmente viene dada por cambios en el negocio:

- 1 Extender o cambiar la capacidades - la extensibilidad permite seguir siendo competitivo en el mercado
- 2 Quitar capacidades no deseadas - simplificar el producto para hacerlo más liviano o más barato
- 3 Adaptarse a nuevos ambientes de ejecución – portabilidad hace al producto más flexible para más clientes
- 4 Reestructurar - racionalizar servicios, modularizar, optimizar, o crear componentes reusables para futuros desarrollos



# Arquitectura de Software

## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Modificabilidad

### Ejemplo de Escenario General

COMPONENTE	DESCRIPCIÓN
FUENTE DE ESTÍMULO	Usuario final, desarrollador, administrador del sistema
ESTÍMULO	Deseo de agregar/modificar/eliminar funcionalidad, atributos de calidad
AMBIENTE	Ejecución, compilación, build, diseño
ARTEFACTO	UI del sistema, plataforma, ambiente, sistemas con los que interactúa
RESPUESTA	Localizar los lugares a ser modificados <ul style="list-style-type: none"> <li>• Realizar las modificaciones sin afectar otras características del sistema</li> <li>• Testear las modificaciones</li> <li>• Instalar modificaciones</li> </ul>
MEDIDA DE LA RESPUESTA	Costo del cambio en términos de: <ul style="list-style-type: none"> <li>• Elementos afectados</li> <li>• Esfuerzo</li> <li>• Dinero</li> <li>• Grado en que afecta a otras características del sistema</li> </ul>

# Arquitectura de Software

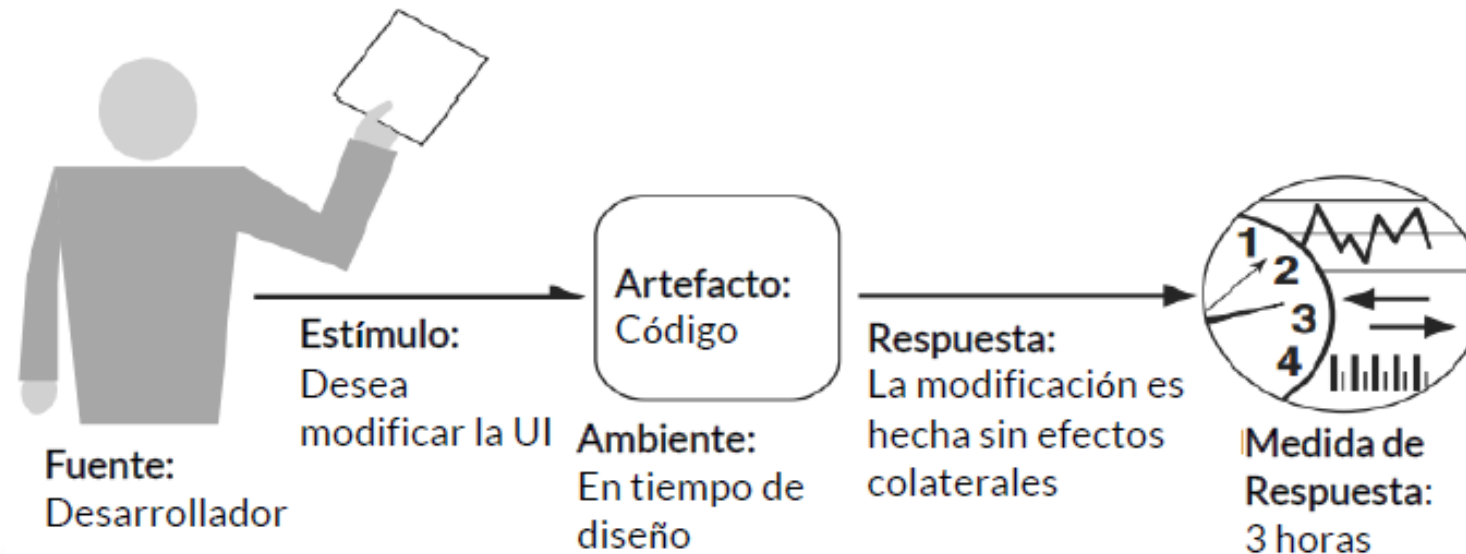
## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Modificabilidad

### Ejemplo de Escenario Concreto

Un desarrollador desea cambiar el color de fondo de la interfaz de usuario. El cambio se realizará sobre el código en tiempo de diseño, debiendo tomar no más de 3 horas en realizarlo y testearlo y sin tener efectos colaterales sobre el comportamiento del sistema



# Arquitectura de Software

## Tácticas de Calidad

### Modificabilidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

- Las tácticas para controlar la modificabilidad tienen como objetivo controlar la complejidad de realizar cambios, así como el tiempo y el costo para realizarlos.



# Arquitectura de Software

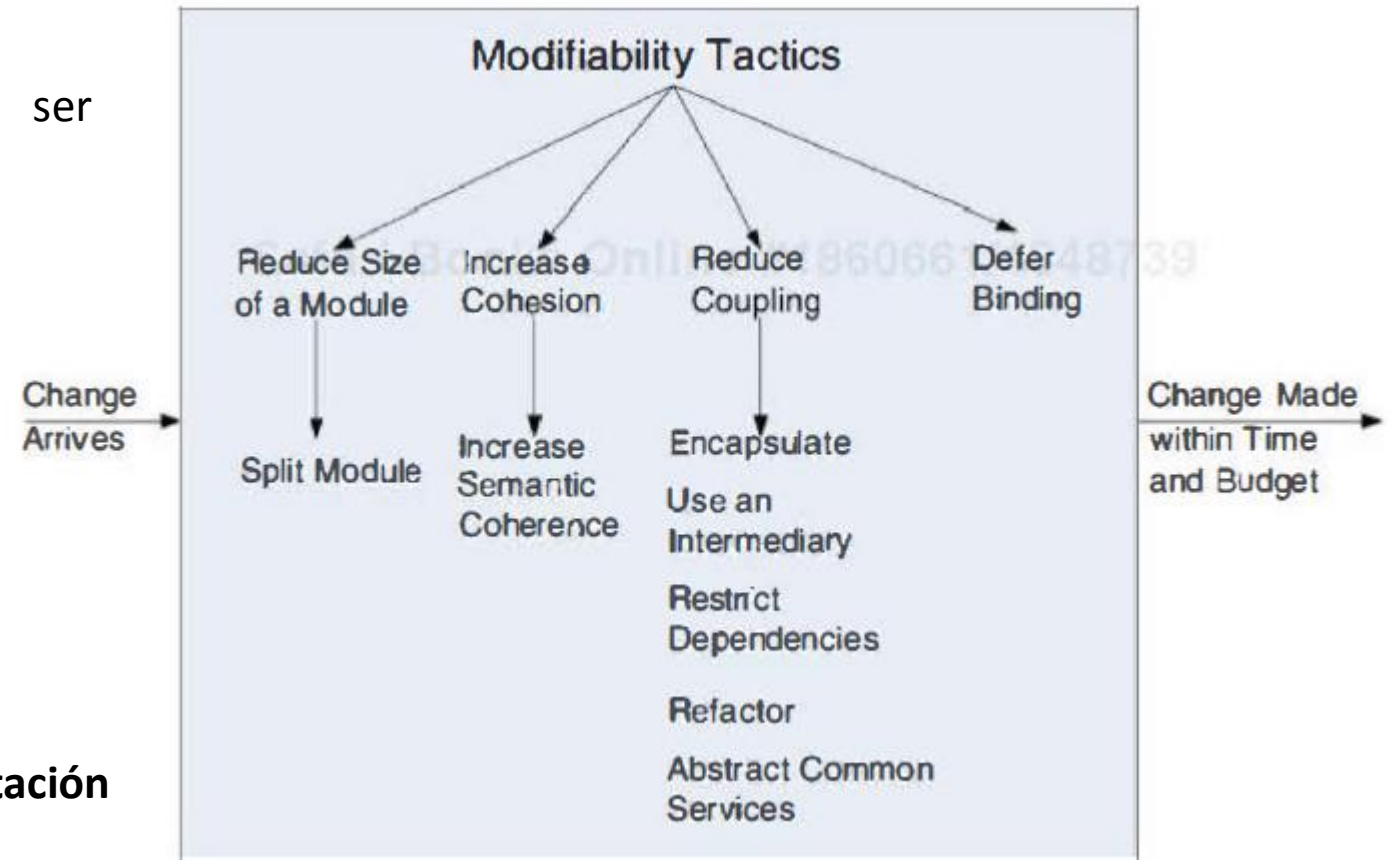
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Modificabilidad

Las tácticas de Modificabilidad pueden ser clasificadas en cuatro categorías:

- 1 Tácticas para **Reducir el tamaño de los módulos**
- 2 Tácticas para **Incrementar la Cohesión**
- 3 Tácticas para **Reducir el Acoplamiento**
- 4 Tácticas para el **Momento de Implementación**

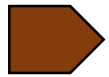


# Arquitectura de Software

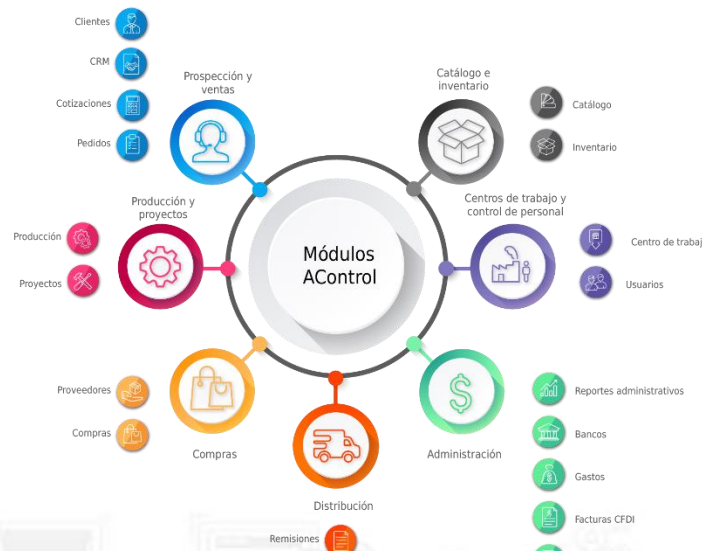
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Modificabilidad – Reducir el tamaño de los módulos



**Dividir el módulo:** Si el módulo que se modifica incluye una gran capacidad, los costos de modificación probablemente serán altos. La refinación del módulo en varios módulos más pequeños debería reducir el costo promedio de los cambios futuros.



# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Modificabilidad – Incrementar la cohesión

Pero ... ¿Qué es cohesión?

- Mide cuán fuertemente están relacionadas las responsabilidades de un módulo.
- La cohesión de un módulo es la probabilidad de que un escenario de cambio que afecte a una responsabilidad también afecte otras responsabilidades (diferentes).
- Cuanto mayor es la cohesión, menor es la probabilidad de que un cambio dado afecte múltiples responsabilidades.
- La alta cohesión es buena; La baja cohesión es mala.

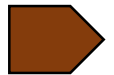


# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Modificabilidad – Incrementar la cohesión



**Incrementar la cohesión semántica:** Si las responsabilidades A y B en un módulo no sirven igual propósito, deben ubicarse en diferentes módulos. Esto puede implicar la creación de un nuevo módulo o puede implicar trasladar una responsabilidad a un módulo existente.

Un método para identificar las responsabilidades que se deben trasladar es hacer hipótesis acerca de los cambios probables que afectan a un módulo. Si algunas responsabilidades no se ven afectadas por estos cambios, entonces esas responsabilidades probablemente deberían eliminarse.

**Si un módulo tiene una baja cohesión, puede ser mejorada eliminando responsabilidad no afectadas por los cambios previstos. Así se reduce la posibilidad de efectos colaterales.**



# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Modificabilidad – Reducir el acoplamiento

Pero ... ¿Qué es acoplamiento?

- Si las responsabilidades de dos módulos se superponen de alguna manera, entonces un solo cambio puede afectarlos a ambos.
- Se puede medir esta superposición midiendo la probabilidad de que una modificación a un módulo se propague al otro.
- Esto se llama **acoplamiento**, y el alto acoplamiento es enemigo de la modificabilidad.



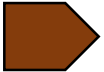
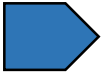
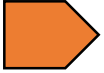


# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Modificabilidad – Reducir el acoplamiento

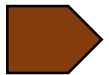
-  **Encapsular:** Táctica de modificabilidad más común. La encapsulación introduce una interfaz explícita para un módulo. Esta interfaz incluye una interfaz de programación de aplicaciones (API) y sus responsabilidades asociadas, como "realizar una transformación sintáctica en un parámetro de entrada a una representación interna".
-  **Utilizar un intermediario:** Busca romper dependencias. Dada una dependencia entre la responsabilidad A y la responsabilidad B (por ejemplo, llevar a cabo A requiere primero llevar a cabo B), la dependencia se puede romper utilizando un intermediario. El tipo de intermediario depende del tipo de dependencia. Por ejemplo, un intermediario de publicación-suscripción eliminará el conocimiento del productor de datos sobre sus consumidores.
-  **Refactorizar:** La refactorización de código fuente es una práctica fundamental de los proyectos de desarrollo ágil, como un paso de limpieza para asegurarse de que los equipos no hayan producido código duplicado o demasiado complejo. Permite alterar la estructura interna sin cambiar el comportamiento externo.

# Arquitectura de Software

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Modificabilidad – Momento de Implementación



#### Diferir la implementación:

En una arquitectura que propicia modificaciones tardías en el ciclo de vida, en promedio, un cambio costará menos que en una arquitectura que fuerza que la misma modificación sea hecha más temprano.



# Arquitectura de Software

## Atributos de Calidad

### Testeabilidad

- Facilidad con la cual el software puede mostrar sus defectos (típicamente a través de pruebas de ejecución).
- Grado en el cual un artefacto de software (sistema, módulo, clase, etc.) soporta *testing* en un contexto de test dado. Si la testeabilidad del artefacto de software es alta, entonces encontrar defectos a través del *testing* es más sencillo.
- Probabilidad de que, suponiendo que el software tiene al menos un defecto, fallará en la siguiente prueba.

### ¿Qué tipos de test se deben hacer?

1 Test automatizados

2 Test unitarios

3 Test de integración

4 Test de interfaces de usuarios



# Arquitectura de Software

## Atributos de Calidad

### Testeabilidad

¿Qué significa que un componente de software sea testeable?

Que sobre el se pueda:

- 1 **Controlabilidad:** Controlar las entradas del componente, probando su estado interno
- 2 **Observabilidad:** Observar las salidas

¿Qué factores inciden en la testeabilidad?

- |   |                                      |
|---|--------------------------------------|
| 1 Nivel de documentación de la arquitectura | 3 Uso de ocultamiento de información |
| 2 Separación de intereses                   | 4 Desarrollo incremental             |



# Arquitectura de Software

## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Testeabilidad

### Ejemplo de Escenario General

COMPONENTE	DESCRIPCIÓN
FUENTE DE ESTÍMULO	Test unitarios; test de integración, test de sistema, test de aceptación, usuarios finales del sistema (ya sea de manera manual o automática)
ESTÍMULO	Un conjunto de test es ejecutado debido a: <ul style="list-style-type: none"><li>• Incremento de código terminado o integración de subsistema terminada</li><li>• Sistema entregado</li></ul>
AMBIENTE	<ul style="list-style-type: none"><li>• En tiempo de desarrollo</li><li>• En tiempo de compilación</li><li>• En tiempo de deployment</li><li>• En tiempo de ejecución</li></ul>
ARTEFACTO	La porción del sistema siendo testeada
RESPUESTA	<ul style="list-style-type: none"><li>• Ejecutar un conjunto de test y capturar el resultado</li><li>• Capturar las actividades que resultaron en una falla</li><li>• Controlar y monitorear el estado del sistema</li></ul>
MEDIDA DE LA RESPUESTA	<ul style="list-style-type: none"><li>• Porcentaje de cobertura del código ejecutable</li><li>• Probabilidad de falla, si la falla existe</li><li>• Tiempo para realizar los test</li><li>• Cantidad de tiempo para preparar un ambiente de test</li></ul>

# Arquitectura de Software

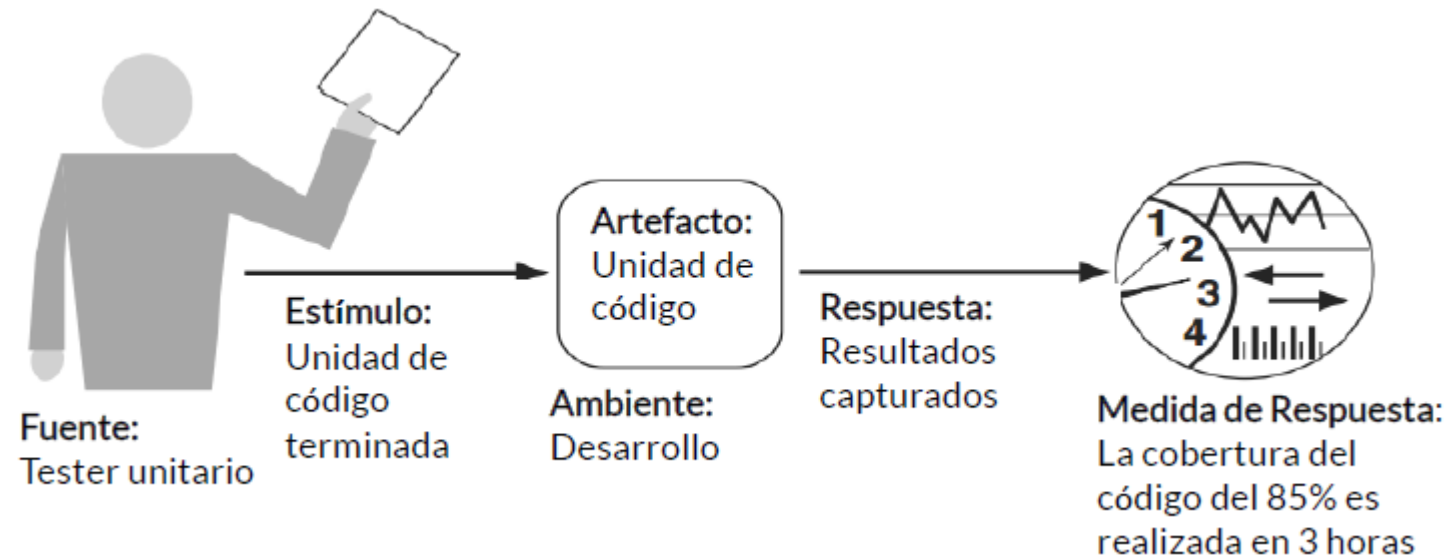
## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Testeabilidad

### Ejemplo de Escenario Concreto

Un tester unitario (desarrollador) terminó una unidad de código durante el desarrollo y realiza una secuencia de test cuyos resultados son capturados logrando una cobertura del 85% en menos de 3 horas.”



# Arquitectura de Software

## Tácticas de Calidad

### Testeabilidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

- El objetivo de las tácticas para la capacidad de prueba es permitir pruebas más fáciles cuando se completa un incremento del desarrollo de software



# Arquitectura de Software

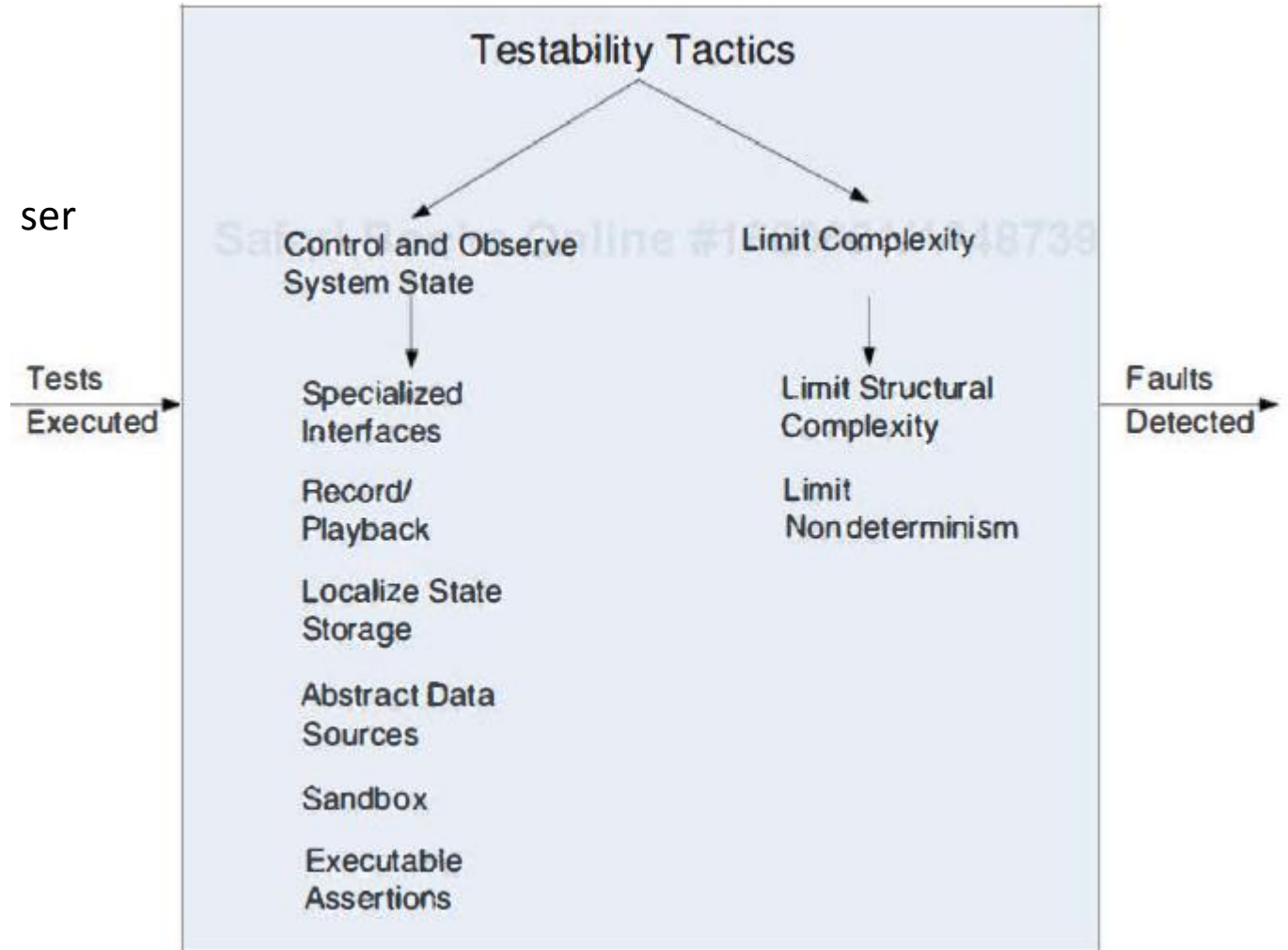
*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

## Tácticas de Calidad

### Testeabilidad

Las tácticas de Testeabilidad pueden ser clasificadas en dos categorías:

- 1 Tácticas para **Controlar y Observar el estado del sistema**
- 2 Tácticas para **Limitar la complejidad**





# Arquitectura de Software

## Atributos de Calidad

### Confiabilidad

- La probabilidad que tiene el software de realizar su función sin fallas durante un período de tiempo determinado.
- Es la capacidad de un sistema de mantenerse operativo en el tiempo.

#### Métricas

**1****Tiempo Medio entre Fallas (MTBF)****2****Porcentaje de operaciones que se ejecutan correctamente en un período**

#### Condiciona otros Atributos de Calidad>

Atributo	Descripción
DISPONIBILIDAD	Es la probabilidad que tiene el Sistema de estar operativo en un determinado punto en el tiempo
TOLERANCIA A FALLOS	La capacidad para mantener un determinado nivel de rendimiento en casos de fallas de software o de violación a su interfaz
RECUPERABILIDAD	La capacidad de restablecer un nivel de rendimiento determinado y recuperar los datos directamente afectados en caso de falla
MADUREZ	La capacidad de evitar fallas como resultado de defectos en el software

# Arquitectura de Software

## Escenarios de Calidad

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

### Confiabilidad

### Ejemplo de Escenario General

COMPONENTE	DESCRIPCIÓN
FUENTE DE ESTÍMULO	Interna al sistema; externa al sistema
ESTÍMULO	Defecto: datos erróneos, falla de hardware, bug en el código, problemas de ambiente
AMBIENTE	Modo Normal, Modo Sobrecargado
ARTEFACTO	Procesadores del sistema, canales de comunicación, almacenamiento persistente, procesos, datos
RESPUESTA	<p>El sistema detectará un evento y haría una o varias de las siguientes acciones:</p> <ul style="list-style-type: none"> <li>• Registrarlo</li> <li>• Notificar a quienes corresponda, incluyendo el usuario y otros sistemas</li> <li>• Tomar las acciones necesarias para dejar los datos consistentes</li> <li>• Reintentar la operación (probablemente con algún delay)</li> </ul>
MEDIDA DE LA RESPUESTA	<ul style="list-style-type: none"> <li>• Tiempo medio entre fallas</li> <li>• Porcentaje de operaciones que se ejecutan correctamente en un periodo</li> <li>• Tiempo que el Sistema puede funcionar en modo sobrecargado</li> </ul>

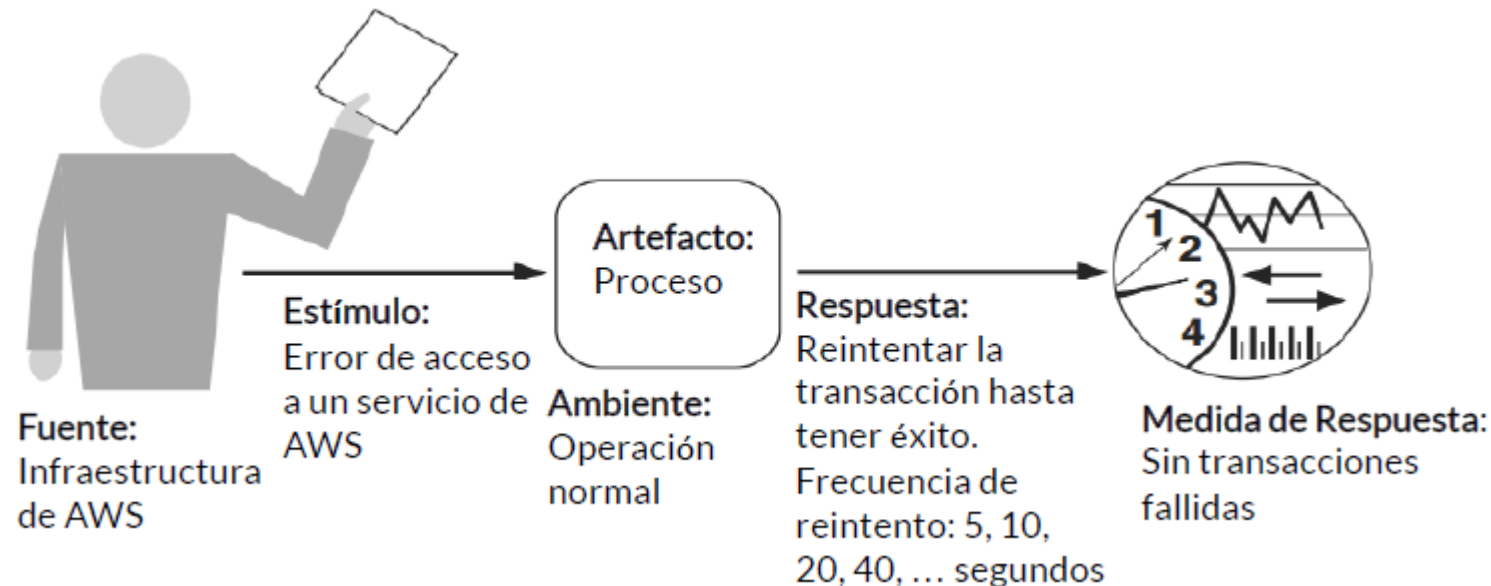
# Arquitectura de Software

## Escenarios de Calidad

### Confiabilidad

#### Ejemplo de Escenario Concreto

En AWS las conexiones a los servicios de AWS pueden fallar eventual y esporádicamente. Ante una falla de la infraestructura de AWS, el sistema debe reintentar la transacción en 5, 10, 20, 40... segundos hasta que logre ejecutarse exitosamente.



# Arquitectura de Software

## Tácticas de Calidad

### Confiabilidad

Las tácticas de Confiabilidad pueden ser clasificadas en tres categorías:

- 1 Tácticas para **Detección de Fallas**
- 2 Tácticas para **Recuperación de Fallas**
- 3 Tácticas para **Prevención de Fallas**

*Software Architecture in Practice (3rd Edition), Bass, Clements, Kazman, AddisonWesley, 2003*

