Resumen Para el Primer Parcial de Arquitectura de Software

**Capítulo 1: El Ciclo de Negocio Arquitectónico**

La arquitectura de software abarca las estructuras de grandes sistemas de software. La vista arquitectónica de un sistema es abstracta, quitando detalles de implementación, algoritmia y representación de datos y concentrándose en el comportamiento e interacción de elementos de caja negra. Una arquitectura de software se desarrolla como el primer paso hacia el diseño de un sistema que tiene un conjunto de propiedades deseadas.

**Definición**: La arquitectura de software de un programa o sistema de computación es la estructura o estructuras del sistema, que comprende elementos de software, las propiedades visibles externamente de esos elementos, y las relaciones entre ellos.

La arquitectura sirve como herramienta de comunicación, razonamiento, análisis y crecimiento de sistemas.

¿Cuál es la relación de una arquitectura de software de un sistema con el entorno en el que ese sistema será construido y existirá? La arquitectura de software es el resultado de influencias técnicas, de negocio y sociales. Su existencia, a su vez, afecta los ambientes técnicos, de negocio y sociales que subsecuentemente influencian futuras arquitecturas. Este ciclo de influencias, desde el entorno a la arquitectura y de vuelta al entorno, se conoce como el ciclo de negocio arquitectónico (Architecture Business Cycle).

**Influencia de los Requerimientos**

Una arquitectura es el resultado de un conjunto de decisiones técnicas y empresariales. Existen varias influencias en su diseño, y la realización de estas influencias cambiará dependiendo del entorno en el cual la arquitectura debe operar. En cualquier esfuerzo de desarrollo, los requerimientos hacen explícitas algunas – pero sólo algunas – de las propiedades deseadas del sistema final. No todos los requerimientos afectan directamente esas propiedades.

**Influencia de los Accionistas**

Muchas personas y organizaciones están interesadas en la construcción de un sistema de software. Los llamamos “stakeholders” o accionistas: el cliente, los usuarios finales, los desarrolladores, el gerente de proyecto, los mantenedores, e incluso aquellos que comercializan el sistema.

Tener un sistema aceptable involucra propiedades tales como performance (eficiencia), confiabilidad, disponibilidad, portabilidad, utilización de memoria, uso de la red, seguridad, modificabilidad, usabilidad, e interoperabilidad con otros sistemas así como también comportamiento. Todos ellos, y otros, afectan cómo es visto el sistema liberado por sus eventuales receptores, y así encuentran una voz en uno o más accionistas. El problema subyacente, por supuesto, es que cada accionista tiene preocupaciones y objetivos diferentes, algunos de los cuales pueden ser contradictorios.

**Influencia de la Organización Desarrolladora**

En adición a los objetivos organizacionales expresados a través de los requerimientos, una arquitectura es influenciada por la estructura o naturaleza de la organización desarrolladora. Las habilidades del personal es una influencia más, pero así también lo son el cronograma de desarrollo y el presupuesto. Existen 3 clases de influencia que provienen de la organización desarrolladora: negocio inmediato, negocio a largo plazo, y estructura organizacional.

**Influencia de los Conocimientos y Experiencias de Arquitectos**

Si los arquitectos de un sistema han tenido buenos resultados usando un enfoque arquitectónico particular, como objetos distribuidos o invocación implícita, existe una gran probabilidad de que intentarán ese mismo enfoque en un nuevo desarrollo. Contrariamente, si su experiencia pasada con este enfoque fue desastrosa, los arquitectos pueden ser reacios a probarlo de nuevo. Las decisiones arquitectónicas pueden provenir de la educación y entrenamiento del arquitecto, la exposición a patrones arquitectónicos exitosos, o la exposición a sistemas que hayan funcionado particularmente pobre o particularmente bien. Los arquitectos pueden también querer experimentar con un patrón arquitectónico o técnica aprendida de un libro o un curso.

**Influencia del Entorno Técnico**

Cuando se diseña una arquitectura, el entorno corriente influencia esa arquitectura. Podría incluir prácticas estándares de la industria o técnicas de ingeniería de software que prevalecen en la comunidad profesional del arquitecto.

**Ramificaciones de las Influencias Sobre la Arquitectura**

Las propiedades requeridas por los objetivos de negocio y organizacionales casi nunca son conscientemente comprendidas, ni siquiera articuladas completamente. Incluso requerimientos del cliente son raramente documentados completamente, lo que quiere decir que no se resuelve el conflicto entre los objetivos de los diferentes accionistas. Sin embargo, los arquitectos necesitan saber y comprender la naturaleza, fuente y prioridad de las restricciones de su proyecto cuanto antes. Por lo tanto, deben identificar y hacer participar a los accionistas activamente para solicitar sus necesidades y expectativas.

La participación temprana de los accionistas permite a los arquitectos comprender las restricciones de la tarea, manejar expectativas, negociar prioridades, y hacer concesiones. Las revisiones de la arquitectura y la construcción iterativa de prototipos son dos maneras de conseguirlo.

Para un arquitecto efectivo, las habilidades de diplomacia, negociación y comunicación son esenciales.

**La Arquitectura Afecta los Factores que la Influencian**

La relación entre los objetivos de negocio, requerimientos del producto, la experiencia del arquitecto, las arquitecturas, y sistemas desplegados forman un ciclo con bucles de retroalimentación que la empresa puede administrar. Una empresa administra este ciclo para manejar crecimiento, para expandir su área empresarial, y para tomar ventaja de inversiones previas en arquitectura y construcción de sistemas. Parte de la retroalimentación proviene de la arquitectura misma y otra parte del sistema construido a partir de ella:

1. La arquitectura afecta la estructura de la organización desarrolladora. La arquitectura prescribe una estructura para un sistema; particularmente prescribe las unidades de software que deben ser implementadas (u obtenidas) e integradas para formar el sistema. Estas unidades son la base para la estructura del desarrollo del proyecto. Se forman equipos para cada unidad de software individual. Los equipos se incrustan en la estructura de la organización. Retroalimentación de la arquitectura a la organización desarrolladora.
2. La arquitectura puede afectar los objetivos de la organización desarrolladora. Un sistema exitoso construido a partir de ella puede permitir a la empresa establecer un punto de apoyo en un área de mercado específico. La arquitectura puede proveer oportunidades para la producción y despliegue efectivos de sistemas similares, y la organización puede ajustar sus objetivos para tomar ventajas de su pericia nueva para sondear el mercado. Retroalimentación del sistema a la organización desarrolladora.
3. La arquitectura puede afectar los requerimientos del cliente para el siguiente sistema dándole al cliente la oportunidad de recibir un sistema (basado en la misma arquitectura) más confiable, económico y a tiempo que si el sistema subsecuente hubiera sido construido de cero.
4. El proceso de la construcción de un sistema afectará la experiencia del arquitecto para con los sistemas posteriores mediante la adición a la base de experiencia corporativa.
5. Pocos sistemas influenciarán y efectivamente cambiarán la cultura de la ingeniería de software, esto es, el entorno técnico en el cual los constructores de sistemas operan y aprenden.

**Actividades de Arquitectura**

1. Crear el modelo de negocio (business case) para el sistema.
2. Entender los requerimientos.
3. Crear o seleccionar la arquitectura.
4. Documentar y comunicar la arquitectura.
5. Analizar o evaluar la arquitectura.
6. Implementar el sistema en base a la arquitectura.
7. Asegurarse de que la implementación conforma la arquitectura.

**Características de una Buena Arquitectura**

No existe tal cosa como una arquitectura inherentemente buena o mala. Las arquitecturas son más o menos aptas para ciertos propósitos. Las arquitecturas pueden ser evaluadas, pero sólo en el contexto de sus objetivos específicos. No obstante, hay ciertas reglas de oro que deberían seguirse cuando se diseña una arquitectura.

Recomendaciones de Procesos:

1. La arquitectura debe ser el producto de un solo arquitecto o de un grupo pequeño de arquitectos con un líder identificado.
2. El arquitecto (o equipo de arquitectura) debe tener los requerimientos funcionales para el sistema y una lista de atributos de calidad articulada y priorizada que la arquitectura debe cumplir.
3. La arquitectura debe ser bien documentada, con al menos una vista estática y una vista dinámica, usando una notación acordada que todos los accionistas puedan entender con un esfuerzo mínimo.
4. La arquitectura debe ser circulada a los accionistas, quienes deben estar activamente involucrados en su revisión.
5. La arquitectura debe ser analizada en función de medidas cuantitativas aplicables, y evaluada formalmente por los atributos de calidad antes de que sea demasiado tarde para hacer cambios a la misma.
6. La arquitectura debe prestarse para una implementación incremental mediante la creación de un sistema “esqueleto” en el que los caminos de comunicación son ejercitados pero tiene en principio funcionalidad mínima. Este esqueleto puede entonces ser usado para “hacer crecer” el sistema incrementalmente, haciendo de la integración y el testeo un esfuerzo más fácil.
7. La arquitectura debe resultar en un conjunto específico y pequeño de áreas de contención de recursos, cuya resolución está claramente especificada, circulada y mantenida.

Reglas Estructurales:

1. La arquitectura debe ofrecer módulos bien definidos cuyas responsabilidades funcionales se distribuyen en los principios de ocultación de información y la separación de preocupaciones.
2. Cada módulo debe tener una interfaz bien definida que encapsula u “oculta” aspectos cambiables de otro software que usa sus facilidades. Estas interfaces deben permitir que sus respectivos equipos de desarrollo trabajen independientemente.
3. Los atributos de calidad deben ser logrados usando tácticas arquitectónicas bien conocidas específicas para cada atributo (capítulo 5).
4. La arquitectura nunca debe depender de una versión comercial particular de una herramienta o producto. En el caso de que dependa, debe ser estructurada de manera tal que cuando se quiera cambiar de producto se pueda hacerlo fácilmente y económicamente.
5. Módulos que producen datos deben estar separados de módulos que consumen datos. Esto aumenta la modificabilidad.
6. Para los sistemas de procesamiento en paralelo, la arquitectura debería ofrecer procesos o tareas bien definidas que no necesariamente reflejen la estructura de descomposición del módulo.
7. Cada tarea o proceso debe estar escrito de tal manera que su asignación a un procesador específico pueda ser cambiado fácilmente, incluso en tiempo de ejecución.
8. La arquitectura debe ofrecer un número pequeño de patrones de interacciones simples. Esto es, el sistema debería hacer las mismas cosas de la misma manera.

**Capítulo 2: ¿Qué es la Arquitectura de Software?**

**Otras Definiciones de Arquitectura**

1. “Es el concepto de mayor alto nivel de un sistema en su ambiente.” (IEEE)
2. Arquitectura de Software = {Elementos (qué), Forma (cómo), Restricciones (por qué)} (Perry, 92).
3. Conjunto de las principales decisiones de diseño realizadas sobre el sistema. Comprenden decisiones de: estructura, comportamiento funcional, interacción entre elementos, requerimientos no funcionales, implementación. (Saftp)

**Definición de Kazman: Profundización**

La arquitectura define elementos de software. Encarna información sobre como los elementos se relacionan. Esto quiere decir que omite específicamente cierta información sobre los elementos que no tiene que ver con cómo interaccionan. Por lo tanto, la arquitectura es principalmente una abstracción de un sistema que suprime detalles de elementos que no afectan como ellos usan, son usados por, se relacionan o interactúan con otros elementos. La arquitectura se refiere a la parte pública de la división; detalles privados que tienen que ver con la implementación interna NO son arquitectónicos.

Los sistemas pueden comprender y comprenden más de una estructura y no hay una única estructura que pueda proclamarse irrefutablemente como la arquitectura.

Cada sistema de cómputo con software tiene una arquitectura de software, ya que se puede demostrar que cada sistema está compuesto por elementos y las relaciones entre ellos. Desafortunadamente, una arquitectura puede existir independientemente de su descripción o especificación, lo que aumenta la importancia de la documentación de arquitectura y de la reconstrucción de arquitectura.

El comportamiento de cada elemento es parte de la arquitectura siempre que ese comportamiento pueda ser observado o discernido desde el punto de vista de otro elemento.

La definición es indiferente en cuanto a si la arquitectura para un sistema es buena o mala; es decir, si la arquitectura permitirá o prevendrá que se cumplan los requisitos de comportamiento, eficiencia y ciclo de vida. Esto aumenta la importancia de la evaluación de arquitectura y del diseño de arquitectura.

**Otros Puntos de Vista**

1. Arquitectura es diseño de alto nivel. Esto es cierto, pero no son intercambiables; es decir, no todo diseño de alto nivel es arquitectura.
2. Arquitectura es la estructura general del sistema. Esto implica incorrectamente que la arquitectura tiene una sóla estructura.
3. Arquitectura es la estructura de los componentes de un programa o sistema, sus interrelaciones, y los principios y guías que gobiernan sus diseños y evolución a través del tiempo. Todo sistema tiene una arquitectura que puede ser descubierta y analizada independientemente de cualquier conocimiento del proceso por el cual la arquitectura fue diseñada o evolucionada.
4. Arquitectura es componentes y conectores. Esta definición se concentra en las estructuras arquitectónicas en tiempo de ejecución. Cuando hablamos de relaciones entre elementos, intentamos capturar tanto relaciones en tiempo de ejecución como aquellas relaciones que no se dan en tiempo de ejecución.

**Patrones Arquitectónicos, Modelos de Referencia & Arquitecturas de Referencia**

Un **patrón arquitectónico** es una descripción de tipos de elementos y relaciones junto con un conjunto de restricciones sobre cómo pueden ser usados. Un patrón puede ser pensado como un conjunto de restricciones sobre una arquitectura y estas restricciones definen un conjunto o una familia de arquitecturas que las satisface. Un patrón arquitectónico NO es una arquitectura, pero transmite una imagen útil del sistema pues impone restricciones útiles sobre la arquitectura y por consiguiente sobre el sistema. Uno de los aspectos más útiles de los patrones es que exhiben atributos de calidad conocidos. La elección de un patrón arquitectónico es muchas veces la primera decisión de diseño de un arquitecto. También son conocidos como estilos arquitectónicos.

Un **modelo de referencia** es una división de funcionalidad junto con flujo de datos entre las partes. Un modelo de referencia es una descomposición estándar de un problema conocido en partes que cooperativamente resuelven el problema.

Una **arquitectura de referencia** es un modelo de referencia mapeado a elementos de software (que cooperativamente implementan la funcionalidad definida en el modelo de referencia) y el flujo de datos entre ellos. Una arquitectura de referencia es el mapeo de la funcionalidad a una descomposición del sistema. El mapeo puede ser uno a uno, pero no necesariamente lo sea. Un elemento de software puede implementar parte de una función o varias funciones.

Los modelos de referencia, patrones arquitectónicos y arquitecturas de referencia NO son arquitecturas; son conceptos útiles que capturan elementos de una arquitectura.

**¿Por Qué es Importante la Arquitectura de Software?**

Existen 3 razones por las cuales la arquitectura de software es importante desde una perspectiva técnica:

1. Comunicación entre los accionistas.

A cada accionista de un sistema de software le interesan diferentes características del sistema que son afectadas por la arquitectura. La arquitectura provee un lenguaje común en el cual diferentes intereses pueden ser expresados, negociados y resueltos en un nivel intelectual manejable incluso en sistemas grandes y complejos.

1. Decisiones tempranas de diseño.

La arquitectura de software representa las decisiones de diseño más tempranas: Define restricciones sobre la implementación; dicta la estructura organizacional; inhibe o habilita atributos de calidad de un sistema; permite realizar predicciones de calidad sobre un sistema; facilita razonar sobre el cambio y manejarlo; ayuda en la construcción evolucionaria de prototipos; permite crear estimaciones de tiempo y costos más exactos.

1. Abstracción transferible (re-uso) de un sistema.

Se da en estos casos: líneas de productos de software comparten una arquitectura común; sistemas pueden ser construidos usando elementos grandes desarrollados externamente; menos es más: vale la pena restringir el vocabulario de alternativas de diseño; la arquitectura permite desarrollo en base a plantillas; la arquitectura puede ser la base para entrenamiento.

**Estructuras Arquitectónicas & Vistas**

Una vista es una representación de un conjunto coherente de elementos arquitectónicos, escritos por y leídos por accionistas. Consiste de una representación de un conjunto de elementos y las relaciones entre ellos. Una estructura es un conjunto de elementos en sí mismo, tal como existen en software o hardware.

Las estructuras arquitectónicas pueden ser divididas en 3 grupos, dependiendo de la naturaleza de los elementos que ilustran:

1. **Estructuras de Módulos**: Los elementos son módulos, unidades de implementación. Los módulos representan una forma basada en código para considerar el sistema. Son áreas asignadas de responsabilidad funcional.
2. **Estructuras de Componentes & Conectores**: Aquí los elementos son componentes en tiempo de ejecución (unidades principales de computación) y conectores (vehículos de comunicación entre componentes).
3. **Estructuras de Asignación**: Muestran la relación entre elementos de software y los elementos en uno o más entornos externos en los cuales el software es creado y ejecutado.

Estas estructuras responden a 3 preguntas que el diseño arquitectónico involucra:

* ¿Cómo debe ser estructurado el sistema como un conjunto de de unidades de código (módulos)?
* ¿Cómo debe ser estructurado el sistema como un conjunto de elementos que tienen comportamiento en tiempo de ejecución (componentes) e interacciones (conectores)?
* ¿Cómo debe el sistema relacionarse con estructuras que no son software en su entorno?

**Estructuras de Software**

1. Estructuras Basadas en Módulos:
2. **Descomposición**: Las unidades son módulos relacionados entre sí por la relación “es un submódulo de”, mostrando como módulos más grandes son descompuestos en módulos más pequeños recursivamente hasta que son lo suficientemente pequeños como para ser entendidos fácilmente. Esta estructura provee una gran parte de la modificabilidad del sistema, asegurándose de que los posibles cambios recaigan como mucho en un grupo pequeño de módulos. A menudo, es utilizado como la base para la organización del desarrollo del proyecto, incluyendo la estructura de la documentación, su integración y planes de testeo. Las unidades en esta estructura a menudo tienen nombres específicos de la organización.
3. **Usos**: Las unidades de esta estructura también son módulos, o procesos o recursos sobre las interfaces de módulos. Las unidades se relacionan mediante la relación “usa”. Una unidad usa a la otra si la correctitud de la primera requiere la presencia de una versión correcta de la segunda. Esta estructura se usa para ingeniar sistemas que pueden ser fácilmente extendidos para agregar funcionalidad o de los cuales se puede extraer fácilmente subconjuntos funcionales útiles. Permite el desarrollo incremental.
4. **En Capas**: Cuando las relaciones de uso en la estructura son controlados cuidadosamente de forma particular, emerge un sistema de capas, donde una capa es un conjunto coherente de funcionalidad relacionada. Las capas son a menudo diseñadas como abstracciones (máquinas virtuales) que ocultan especificaciones a la implementación de las capas de abajo a las capas de arriba, generando portabilidad.
5. **De Clases o Generalización**: Las unidades de módulos en esta estructura se llaman clases. La relación es “hereda de” o “es una instancia de”. Esta vista apoya el razonamiento sobre colecciones de comportamiento o capacidad similares y diferencias parametrizadas que son capturadas con la creación de subclases. La estructura de clases permite razonar sobre el re-uso y la adición incremental de funcionalidad.
6. **Modelo de Datos**. (No está en el libro).
7. Estructuras de Componentes & Conectores:
8. **Proceso o Procesos de Comunicación**: Se ocupa de los aspectos dinámicos de un sistema en ejecución. Las unidades aquí son procesos o hilos que son conectados entre sí mediante comunicación, sincronización, y/o operaciones de exclusión. La relación aquí es adjunción. La estructura de proceso es importante para ayudar a ingeniar una ejecución eficiente del sistema y disponibilidad.
9. **Concurrencia**: Permite al arquitecto determinar oportunidades para el paralelismo y las locaciones donde una contención de recursos puede ocurrir. Las unidades son componentes y los conectores son “hilos lógicos”. Un hilo lógico es una secuencia de computación que más tarde en el proceso de diseño puede ser asignado a un hilo físico separado. Es usado tempranamente en diseño para identificar requerimientos para el manejo de temas asociados con la ejecución concurrente.
10. **Datos Compartidos o Repositorio**: Esta estructura comprende componentes y conectores que crean, almacenan y acceden a datos persistentes. Muestra cómo los datos son producidos y consumidos por elementos de software en tiempo de ejecución, y puede ser usada para asegurar una buena performance e integridad de datos.
11. **Cliente-Servidor**: Los componentes son los clientes y servidores, y los conectores son los protocolos y mensajes que comparten para llevar a cabo la labor del sistema. Esto es útil para la separación de intereses (apoyando a la modificabilidad), para la distribución física, y el balanceo de carga (apoyando a la eficiencia en tiempo de ejecución).
12. **Otras vistas**: Pipes & Filters, Publish/Suscribe, Peer to Peer, Service Oriented Architecture, Multi-tier. (No están en el libro)
13. Estructuras de Asignación:
14. **Despliegue**: Esta estructura muestra como el software es asignado a elementos hardware de proceso y comunicación. Los elementos son software (usualmente un proceso de una vista de componentes y conectores), entidades hardware (procesadores), y vías de comunicación. Las relaciones son “asignado a”, mostrando sobre qué unidades físicas residen los elementos de software, y “migra a”, si la asignación es dinámica. Esta vista permite al ingeniero razonar acerca de la eficiencia, integridad de datos, disponibilidad y seguridad. Es de particular interés en sistemas paralelos o distribuidos.
15. **Implementación**: Muestra cómo los elementos de software (usualmente módulos) son mapeados a las estructuras de archivos en los entornos de desarrollo, integración o control de configuración del sistema.
16. **Asignación de Trabajo**: Esta estructura asigna responsabilidad para implementar e integrar los módulos a los equipos de desarrollo apropiados.

**Capítulo 4: Atributos de Calidad**

**Funcionalidad & Arquitectura**

¿Qué es funcionalidad? Es la capacidad de un sistema de hacer el trabajo para el cual fue destinado. Una tarea requiere que muchos o la mayoría de los elementos del sistema trabajen de manera coordinada para completar la labor. Por lo tanto, si los elementos no fueron asignados con su responsabilidad correcta o no fueron dotados con las facilidades correctas para coordinar con otros elementos, el sistema no va a poder ofrecer la funcionalidad requerida. La funcionalidad es en gran medida independiente de la estructura. La arquitectura de software limita su asignación estructural cuando otros atributos de calidad son importantes. El interés de la funcionalidad es cómo interactúa con, y limita a, esos otros atributos de calidad.

**Escenarios de Atributos de Calidad**

Un escenario de atributo de calidad es un requisito de calidad de un atributo específico. Consiste de 6 partes:

1. Fuente del Estímulo: Alguna entidad que generó el estímulo.
2. Estímulo: Una condición que necesita ser considerada cuando llega al sistema.
3. Entorno: El estímulo se produce dentro de ciertas condiciones.
4. Artefacto: Algún artefacto es estimulado; puede ser todo el sistema o partes de él.
5. Respuesta: La actividad realizada luego de la llegada del estímulo.
6. Medida de respuesta: Cuando la respuesta ocurre, ésta debe ser medible de alguna manera para porder testear el requerimiento.

**Escenarios de Atributos de Calidad en Práctica**

**Availability (Confiabilidad o Disponibilidad)**

La disponibilidad trata de los fallos del sistema y sus consecuencias. Un fallo en el sistema ocurre cuando el sistema deja de ofrecer un servicio consistente con su especificación. Tal fallo es observable por los usuarios del sistema.

Dentro de las áreas de interés se encuentran: cómo se detecta el fallo del sistema, cuán frecuentemente ocurren los fallos, qué sucede cuando ocurre un fallo, cuánto tiempo tiene permitido el sistema quedar fuera de operación, cuándo las fallas pueden ocurrir de forma segura, cómo se pueden prevenir los fallos, y qué tipo de notificaciones son requeridas cuando los fallos ocurren.

Falta: puede volverse un fallo si no se enmascara o corrige. Por ello, un fallo es observable por el usuario y una falta no. Cuando la falta se vuelve observable, se convierte en fallo. Si se ejecuta un código con faltas pero el sistema logra recuperarse de las faltas sin que éstas sean observables, no existe fallo.

Una vez que el sistema falla, aparece un concepto relacionado importante: el tiempo que lleva reparar el fallo. El tiempo que lleva reparar es el tiempo que pasa hasta que la falla ya no es observable. La disponibilidad de un sistema es la probabilidad de que esté operativo cuando se lo necesita. Típicamente se define como: el tiempo medio entre fallas/ (tiempo medio entre fallas + tiempo medio de reparación). Las bajas programadas usualmente no son consideradas cuando se calcula la disponibilidad, porque el sistema no es necesitado por definición.

**Escenario:**

Fuente de Estímulo: Interna o Externa.

Estímulo: Omisión, Choque, Cronometraje, Respuesta

Artefacto: Procesador, Canal de Comunicación, Proceso, Almacenamiento.

Entorno: El estado del sistema cuando la falta o falla ocurre (Normal o Degradado).

Respuesta: Logear la falla, Notificar usuarios selectos u otros sistemas, Cambiar a modo degradado con menos capacidad o menos funciones, Apagado de sistemas externos, Dejar de estar disponible durante la reparación.

Medida de Respuesta: Porcentaje de disponibilidad, Tiempo para reparar, Tiempos durante el cual el sistema debe estar disponible, o la duración durante la cual el sistema debe estar disponible.

**Modifiability (Modificabilidad o Mantenibilidad)**

La modificabilidad trata sobre el costo de hacer un cambio y trae consigo dos preocupaciones:

1. ¿Qué puede cambiar (artefacto)? Un cambio puede ocurrirle a cualquier aspecto de un sistema, más comunmente las funciones que ese sistema computa, la plataforma sobre el cual el sistema existe, el entorno dentro del cual el sistema opera, las cualidades que el sistema exhibe y su capacidad.
2. ¿Cuándo se genera el cambio y quién lo genera (entorno)? Cambios pueden hacerse en la implementación, durante la compilación, durante la construcción, o durante la ejecución. El cambio también puede ser generado por un desarrollador, un usuario final o un administrador del sistema.

**Escenario:**

Fuente de Estímulo: Usuario final, desarrollador, administrador del sistema.

Estímulo: Agregar/borrar/modificar/variar funcionalidad, atributos de calidad, capacidad.

Artefacto: Interfaz de usuario, plataforma, entorno; sistema que interopera con el sistema objetivo.

Entorno: En tiempo de ejecución, tiempo de compilación, tiempo de construcción, tiempo de diseño.

Respuesta: Localiza lugares en la arquitectura a ser modificados; realiza modificacion sin afectar otra funcionalidad; prueba la modificación; despliega modificación.

Medida de Respuesta**:** Costo en términos de números de elementos afectados, esfuerzo, dinero; en qué medida fueron afectadas otras funcionalidades u otros atributos de calidad.

**Performance (Eficiencia)**

La eficiencia trata sobre tiempos. Ocurren eventos y el sistema debe responder a ellos. Una de las cosas que complica la eficiencia es el número de fuentes de eventos y patrones de llegada.

**Escenario:**

Fuente de Estímulo: Interna o Externa (posiblemente múltiple).

Estímulo: Llegadas de eventos (periódicamente, estocásticamente, esporádicamente).

Artefacto: Los servicios del sistema.

Entorno: Varios modos operacionales (normal, emergencia, sobrecarga).

Respuesta: Procesa los estímulos; cambia el nivel de servicio.

Medida de Respuesta**:** Latencia (tiempo de respuesta luego de un evento); deadline (tiempo límite), throughput (capacidad de procesar transacciones por unidad de tiempo), jitter, razón de pérdida, pérdida de datos.

**Security (Seguridad)**

La seguridad es la medida de la capacidad del sistema de resistir el uso desautorizado mientras sigue ofreciendo sus servicios a usuarios legítimos. El intento de violar la seguridad se llama ataque y puede tomar varias formas.

La seguridad puede caracterizarse como el sistema proveyendo no repudio, confidencialidad, integridad, garantía (assurance), disponibilidad y auditoría (seguimiento de actividades para poder reconstruirlas).

**Escenario:**

Fuente de Estímulo: Individuo o sistema que puede ser identificado correctamente, identificado incorrectamente, o sin identidad; que a su vez es interno o externo, autorizado o no, y que tiene acceso a muchos recursos o a recursos limitados.

Estímulo: Trata de visualizar datos, cambiar/borrar datos, acceder a servicios del sistema, reducir disponibilidad de los servicios del sistema.

Artefacto: Servicios del sistema; datos dentro del sistema.

Entorno: online/offline, conectado/desconectado, con cortafuegos o abierto.

Respuesta: Autorizar usuarios legítimos y garantizarles acceso a lo datos y servicios; denegar usuarios desautorizados, negándoles el acceso, y reportando acceso desautorizado.

Medida de Respuesta: Tiempo/esfuerzo/recursos requeridos para evitar medidas de seguridad con probabilidad de éxito; probabilidad de detectar un ataque; probabilidad de identificar al individuo responsable del ataque; porcentaje de servicios que aún son disponibles bajo un ataque de negación de servicios; en qué medida fueron dañados datos/servicios; en qué medida fue denegado acceso legítimo.

**Testeability (Testeabilidad)**

La testeabilidad de software se refiere a la facilidad con la cual se pueden demostrar las fallas de un sistema de software a partir de pruebas (“testing”). Al menos 40% del costo de desarrollar sistemas bien diseñados es gastado en testeo. Si el arquitecto puede reducir este costo, el beneficio es grande.

En particular, la testeabilidad se refiere a la probabilidad, asumiendo que el software tiene al menos una falla, de que falle en su proxima ejecución de prueba.

Para que un sistema sea sometido a pruebas, debe ser posible controlar el estado interno de cada componente y entradas y luego observar sus salidas. Frecuentemente esto se realiza mediante el uso de un arnés de prueba, que es un software especializado en ejercitar el software bajo pruebas.

El testeo se realiza por varios desarrolladores, testers, verificadores, o usuarios y es el último paso de varias partes del ciclo de vida del software. Las medidas de respuesta para la testeabilidad tienen que ver con cuán efectivas son las pruebas em descubrir fallos y cuánto tiempo lleva realizar las pruebas para un nivel de cobertura deseado.

**Escenario:**

Fuente de Estímulo: Desarrollador de unidad, integrador incremental, verificador del sistema, tester de aceptación del cliente, usuario del sistema.

Estímulo: Análisis, arquitectura, diseño, clase, integración de subsistema completa, sistema liberado.

Artefacto: Parte de un diseño, parte de un código, aplicación completa.

Entorno: En tiempo de diseño, en tiempo de desarrollo, en tiempo de compilación, en tiempo de despliegue.

Respuesta: Provee acceso a valores de estado; provee valores computados; prepara entorno de pruebas.

Medida de Respuesta: Porcentaje de sentencias ejecutables ejecutadas; probabilidad de fallo si existe falta; tiempo que lleva realizar las pruebas; largo de la cadena de dependencia más larga en una prueba; largo del tiempo para preparar el entorno de pruebas.

**Usability (Usabilidad)**

La usabilidad tiene que ver con cuán fácil es para el usuario completar una determinada tarea y el tipo de soporte al usuario que el sistema provee. Puede ser dividido en 5 áres:

1. Mecanismos de aprendizaje: ¿qué puede hacer el sistema para hacer el aprendizaje más fácil?
2. Uso eficiente del sistema: ¿qué puede hacer el sistema para que el usuario sea más eficiente?
3. Minimización del impacto de errores: ¿qué puede hacer el sistema para que los errores de usuario tengan un impacto mínimo?
4. Adaptación del sistema a las necesidades del usuario: ¿cómo puede el usuario (o el propio sistema) adaptarse para hacer la tarea del usuario más fácil?
5. Incremento de la confianza y la satisfacción: ¿qué hace el sistema para que el usuario confíe en que se esté tomando la acción correcta?

**Escenario:**

Fuente de Estímulo: Usuario final.

Estímulo: aprender a usar el sistema; usar el sistema eficientemente, minimalizar el impacto de errores, adaptar el sistema, sentirse cómodo.

Artefacto: Sistema.

Entorno: Las acciones de los usuarios con los cuales tiene que ver la usabilidad ocurren siempre en tiempo de ejecución.

Respuesta: Depende de cada área (ver libro – pág. 93)

Medida de Respuesta: Tiempo de tarea, número de errores, número de problemas resueltos, satisfacción del usuario, ganancia de conocimiento del usuario, razón de operaciones éxitosas sobre operaciones totales, cantidad de tiempo/información perdida.

**Otros Atributos de Calidad del Sistema**

Existen otros atributos que son importantes: Escalabilidad (modificación de la capacidad del sistema), Portabilidad (modificación de la plataforma), Interoperabilidad.

**Atributos de Calidad de Negocio**

1. Tiempo para la comercialización
2. Costo & beneficio
3. Tiempo de vida proyectado para el producto: si se va a usar poco tiempo, menos modificable.
4. Mercado objetivo: características pueden variar según usuarios.
5. Cronograma de despliegue: si se introduce el producto con funcionalidad básica, tiene que ser flexible y customizable para poder ser expandido o contraído en el futuro.
6. Integración con sistemas legados

**Atributos de Calidad de Arquitectura**

1. Integridad Conceptual: el tema o visión subyacente que unifica el diseño del sistema en todos los niveles. La arquitectura debería hacer cosas similares de maneras similares. La integridad conceptural de un sistema es de importancia primordial; los sistemas que no la tienen, fallan.
2. Correctitud y completitud: Son esenciales para que la arquitectura pueda llevar a cabo todos los requerimientos del sistema y límites de recursos en tiempo de ejecución. Una evaluación formal de la arquitectura es la mejor manera de verificar que se tenga correctitud y completitud.
3. Edificabilidad (buildability): permite al sistema ser completado por el equipo disponible en tiempo y forma y abrirse a ciertos cambios a medida que el desarrollo progresa.

**Capítulo 5: Logrando Calidades (Tácticas)**

**Capítulo 9: Documentando Arquitectura**

**Parte Práctica del Curso**