Requerimientos

Tipos:

- **Requerimientos Funcionales**
- **Restricciones**
- **Atributos de Calidad**

Atributos de Calidad

"propiedad medible o testeable de un sistema que se utiliza para indicar qué tan bien el mismo satisface las necesidades de los interesados"

Mide qué tan bueno es el producto respecto a alguna dimensión (AC) Definen cualidades del producto en general o de determinadas funcionalidades (ortogonalidad) Son fundamentales porque determinan la arquitectura y en base a ellos se definen Tácticas y estilos de arquitectura.

Disponibilidad

Capacidad que tiene el sistema de enmascarar defectos o reparar o recuperarse de fallas .

Escenario

- **Fuente del estímulo:** Diferenciamos entre orígenes internos y externos de defectos o fallas porque la respuesta del sistema podría ser distinta.
- **Estímulo:** Un defecto de una de las siguientes clases ocurre:
  - **Omisión**: Un componente falla a responder a un input.
  - **Crash**: El componente sufre repetidamente defectos de omisión.
  - **Timing**(Cronometraje): El componente responde, pero la respuesta es muy rápida o muy lenta
  - **Respuesta**: Un componente responde con un valor incorrecto.
- **Artefacto:** Especifica el recurso que es requerido que este altamente disponible. Procesadores, canales de comunicación, Almacenamiento persistente, procesos.
- **Ambiente:** El estado del sistema cuando el defecto o la falla ocurre puede afectar la respuesta buscada. Operación normal, iniciando, apagando, modo de reparación, operación degradada, operación en sobrecarga.
- **Respuesta:** Primero el defecto debe detectarse y separarse antes que otra respuesta sea posible. Luego que es detectado el defecto, debe recuperarse. Acciones posibles: Loggear la falla, notificar a los usuarios o a otros sistemas, acciones para limitar el daño, cambiar a un modo degradado, apagar sistemas externos, o volverse no disponible.

Tácticas

- *Detección de Defectos*
  - **Ping/Echo:** Ping para verificar si está vivo
  - **Monitor:** Es un componente que monitorea el estado de distintas partes del sistema.
  - **HeartBeat**: Parecido al Ping/echo pero a la inversa, el monitoreado da señales de vida.

    - **Timestamp**:
    - **Sanity checking:** Valída que el resultado de operaciones sean coherentes.
    - **Monitoreo de la condición:** Chequea de forma simple condiciones de procesos o dispositivos o supuestos realizados en el diseño. (ej: checksum)
    - **Voting**: 3 componentes que realizan la misma tarea, se los ejecuta bajo las mismas condiciones y se evalúan los resultados. (igual entrada => igual salida)
    - **Replicación**: Parecido al Voting pero los componentes son clones.
    - **Redundancia funcional**: Es un voting de igual responsabilidad, entradas y condiciones pero distinta implementación.
    - **Redundancia analítica**: Es voting que permite componentes distintos, entradas distintas, salidas distintas pero que tienen la misma responsabilidad.
    - **Detección de excepciones**: Detección de interrupciones del flujo normal del sistema.
    - **Self-test**: Componentes o sistemas que pueden ejecutar sus propios test.
- *Recuperación de Defectos* *(en azul los de reintroducción)*
    - **Redundancia Activa (hot spare)**: Todos los nodos reciben los inputs al momento. Esta redundancia puede servir como sustituto al instante.
    - **Redundancia Pasiva (warm spare)**: Los nodos activos reciben los inputs periódicamente.
    - **Redundancia Fría (cold spare)**: Los reemplazos están offline hasta que ocurre una falla.
    - **Manejo de Excepciones**
    - **Rollback**: Vuelve a un estado de correcto funcionamiento.
    - **Software Upgrade**:
    - **Retry**: Se asume que fue un error casual.
    - **Ignorar comportamiento erróneo**
    - **Degradación**: Mantiene en funcionamiento sólo los sistemas más críticos, pero el resto desactivados.
    - **Reconfiguración**
    - **Shadow:** Un componente que falló, luego de arreglado se lo reintroduce en “modo sombra“ para ser monitoreado en producción antes de activarlo por completo.
    - **Resincronización de Estados**: La copia activa y la copia “arreglada” sincronizan sus estados para verificar el correcto funcionamiento.
    - **Escalating Restart:** Granularidad de los componentes a la hora de reiniciarlos.
    - **Non-Stop Forwarding:** funcionalidad y datos separados en dos planos, si falla una parte, se cambian las rutas a una parte redundante.
- *Prevención de Defectos*
    - **Remover Servicio**
    - **Transacciones**
    - **Modelo Predictivo**: Monitores que evalúan posibles causas de defectos.
    - **Prevención de Excepciones**
    - **Incrementar el conjunto de competencia**: Aumentar estados de competencia. Poder responder a errores desconocidos.

Modificabilidad

Capacidad que tiene un sistema de aceptar una modificación al menor costo posible.

Escenario-

- **Fuente del estímulo**: Quien hace el cambio, usuario final, desarrollador, administrador del sistema
- **Estimulo**: Especifica el cambio a ser realizado, pedido para agregar, borrar o modificar funcionalidad, o cambio en un atributo de calidad, capacidad o tecnología.
- **Artefactos**: Que es lo que va a cambiar: código, datos, interfaces, componentes, recursos, configuraciones, etc.
- **Ambiente**: Cuándo será realizado el cambio, en ejecución, compilación, tiempo de construcción, tiempo de iniciación, tiempo de diseño.
- **Respuesta**: Hacer la modificación, testearla y desplegarla.
- **Medida de la respuesta**:
    - Costo y Tiempo:
        - Número, tamaño y complejidad de los artefactos afectados.
        - Esfuerzo.
        - Tiempo.
        - Dinero.
        - Nuevos defectos introducidos.

Tácticas

- *Reducción del Tamaño de un Módulo*
    - **Dividir módulo**: Refinar el módulo en submódulos.
- *Aumentar Cohesión*
    - **Aumentar coherencia semántica:** Propósitos distintos, en módulos distintos.
- *Reducir el Acoplamiento*
    - **Encapsulamiento**: Con una interfaz explícita, evita la propagación de errores.
    - **Usar intermediario:** Rompe la dependencia
    - **Restricción de Dependencias**
    - **Refactorización**: Cuando a dos módulos le afectan el mismo cambio, los mismos pueden ser refactoreados.
    - **Servicios abstractos en común**
- *Aplazar la Unión*
    - *Unir Valores en tiempo de compilación o construcción*
        - **Reemplazo de componentes**
        - **Parametrización**
    - *Unir valores en tiempo de despliegue*
    - *Unir valores en tiempo de inicialización*
        - Archivos de configuración y recursos
    - *Unir valores en tiempo de ejecución*

        - Interpretación de parámetros
        - Name Servers
        - Búsqueda dinámica
        - Plug-ins
        - Publicador-Suscriptor
        - Repositorios Compartidos
        - Polimorfismo

Eficiencia

Capacidad del sistema de responder a eventos en un tiempo dado.

Escenario

- **Fuente del estímulo**: Interna o externa al sistema
- **Estímulo**: Arribo de eventos periódicos, esporádicos o estocásticos (distribución probabilística).
- **Artefacto**: Sistema o uno o más componentes de este.
- **Ambiente**: normal, emergencia, pico de carga, sobrecargado.
- **Respuesta**: Procesamiento de eventos, cambios en el nivel de servicio.
- **Medida de la respuesta**: Latencia, tiempo límite, rendimiento, variación, eventos no procesados.

Tácticas

- *Control de demanda de recursos*
    - **Gestión de la frecuencia de muestreo**
    - **Limitar la respuesta del evento**: Manteniendo una cola de capacidad del procesador.
    - **Priorización de eventos**
    - **Reducción de overhead**: Sacrificar modificabilidad, por ej sacando un intermediario.
    - **Limitar tiempo de ejecución**
    - **Incrementar la eficiencia de los recursos**: Mejora de algoritmos
- *Gestión de Recursos*
    - **Incrementar recursos**: Recursos físicos a servidores y red.
    - **Introducir concurrencia**: Paralelismo
    - **Mantener múltiples copias de cómputos**: Balanceador de carga
    - **Mantener múltiples copias de datos**: Uso de caché
    - **Límite del tamaño de las colas**
    - **Programar recursos** Políticas de procesamiento, ejemplo Round Robin.

Seguridad

Capacidad del sistema de proteger sus datos de accesos no autorizados y manteniendo disponible el acceso a los si autorizados. (Confiabilidad, Integridad, Disponibilidad)

Escenario

- **Fuente del estímulo**: Humano u otro sistema. Puede estar previamente identificado (correcta o incorrectamente), o puede ser desconocido. Un atacante humano puede ser externo o interno.
- **Estimulo**: Es un ataque. Es un intento no autorizado de ver, cambiar o borrar datos, acceder a sistemas de servicios, cambiar el comportamiento del sistema o reducir la disponibilidad.
- **Artefacto**: El objetivo del ataque puede ser contra los servicios de un sistema, los datos dentro de este, o los datos producidos y consumidos por el sistema. Algunos ataques se realizan sobre componentes que se sabe son vulnerables.
- **Ambiente**: El ataque puede suceder cuando el sistema esta online u offline, conectado o desconectado de una red, detrás de un firewall o no, completamente operacional, parcialmente operacional o no operacional.
- **Respuesta**: El sistema debe asegurar que las transacciones son llevadas a cabo de forma que los datos y servicios son protegidos de accesos no autorizados, los datos o servicios no son manipulados sin autorización, los participantes de una transacción están identificados, los participantes no pueden repudiar, el sistema está disponible.
- **Medir la respuesta**: Cuanto de un sistema es comprometido, cuanto tiempo paso hasta la detección, cuantos ataques resistió, cuanto le tomó recuperarse, cuantos datos fueron vulnerables a un ataque particular.

Tácticas

- *Detección de ataques*
    - **Detección de intrusos**
    - **Detección de negación del servicio**
    - **Verificación de integridad del mensaje**: ej checksums
    - **Detección de retraso de mensaje**: Poder detectar intermediarios fraudulentos.
- *Resistencia a ataques*
    - **Identificación de actores**
    - **Autenticar actores**
    - **Autorizar actores**
    - **Limitar acceso**
    - **Limitar la exposición**
    - **Encriptar los datos**
    - **Separación de entidades**: Separar datos sensibles de los no sensibles (incluso una separación física)
    - **Cambiar las configuraciones por default**
- *Reacción a ataques*
    - **Revocación de accesos**
    - **Bloqueo de computadora**
    - **Informar a los actores**
- *Recuperación de ataques*
- *Mantener rastro de auditoría*
- *Restaurar*

Testabilidad

Capacidad que tiene el sistema de descubrir defectos durante el testing.

Escenario

- **Fuente del estímulo**: Testers unitarios, testers de integración, testers del sistema, testers de aceptación, usuarios finales, ya sea ejecutando tests manuales o automáticos.
- **Estimulo**: Conjunto de tests ejecutados debido a la finalización de un incremento de código, una integración completa de un subsistema, la implementación completa de un sistema o la entrega de un sistema a un cliente.
- **Ambiente**: Tiempo de diseño, tiempo de desarrollo, tiempo de compilación, tiempo de integración, tiempo de despliegue o tiempo de ejecución.
- **Artefactos**: La porción del sistema a ser testeada.
- **Respuesta**: Ejecución y captura de los tests, captura de la actividad que resulta en un defecto, control y monitoreo del estado del sistema.
- **Medida de la respuesta**: Esfuerzo para encontrar el defecto, esfuerzo para lograr el porcentaje de cobertura, probabilidad de falla revelada, tiempo para realizar los tests, esfuerzo para detectar las fallas, etc.

Tácticas

- *Controlar y observar el estado del sistema*
    - **Interfaces especializadas**: Permite capturar valores variables de un componente
    - **Record playback**: Poder capturar datos de fallas para poder recrearlas.
    - **Localizar almacenamiento de datos**:
    - **Fuente de datos abstractos**: Abstraer interfaces
    - **Sandbox**: Mocks, stubs
    - **Asserts ejecutables**
- *Limitar la complejidad*
    - **Limitar la complejidad estructural**: Limitar dependencias.
    - **Limitar el no determinismo**: Limitar complejidad de comportamiento.

Interoperabilidad

Capacidad del sistema de comunicarse e intercambiar información relevante a través de interfaces en cierto contexto.

Usabilidad

Capacidad del sistema de para que el usuario pueda realizar la tarea deseada de forma fácil.

Vistas

Representación de un set de elementos arquitectónicos y sus relaciones.

- **Vistas de Casos de Uso**:
    - Elementos: actores, casos de uso
    - Relaciones: asignaciones, dependencias, generalizaciones
- **Vista Lógica**:
    - Elementos: clases, objetos, interfaces, paquetes, colaboraciones
    - Relaciones: asociaciones, dependencias, generalización, realización
- **Vista de Componentes**:
    - Elementos: Componentes, interfaces, puertos
    - Relaciones: Dependencia, generalización, realización,
- **Vista de Procesos**:
    - Elementos: Clases, Hilos, objetos, estados, tareas
    - Relaciones: asociación, dependencia, generalización
- **Vista de Despliegue**:
    - Elementos: Nodos
    - Relaciones: Asociación, dependencia, generalización

<u>Categorías:</u>

- **Módulos:** Muestra y engloba las decisiones de cómo el sistema se particiona en unidades estáticas de implementación.
- **Componentes y Conectores**: Muestra como otras estructuras dinámicas interactúan entre sí en tiempo de ejecución.
- **Asignación**: Muestra y engloba decisiones de cómo se mapearan las estructuras de software a equipos de desarrollo, o en su instalación y ejecución (servidores, etc).
- **Calidad:**
    - **Vista de Seguridad:**
    - **Vista de Comunicaciones**
    - **Vista de Gestión de Errores**
    - **Vista de Confiabilidad**
    - **Vista de Performance**

Patrones y Estilos de Arquitectura

Son un paquete de decisiones de diseños que se repiten comúnmente en la práctica y permiten ser reutilizadas.

Patrón Arquitectónico: Un Contexto, Un problema, una solución. Se debe aclarar que atributos de calidad favorece la solución.

Estructuras de Módulos

- **Estructura de descomposición**: Los módulos se dividen en módulos más pequeños hasta que sean más entendibles y modificables.
- **Estructura de Usos:** La estructura de usos es usada para planear sistemas a los cuales se los puede extender para agregar más funcionalidad. Se utiliza para diseñar sistemas que pueden extenderse para agregar funcionalidad, o de los cuales se pueden extraer subconjuntos funcionales útiles.
- **Estructura de Capas:** Provee portabilidad. Una capa es como una máquina virtual abstracta con un conjunto cohesivo de servicios que a través de interfaces y pudiendo hacer uso sólo de la capa inmediata abajo.
- **Estructura de Clases:** Permite razonar sobre el reuso y la adición incremental de funcionalidad.
- **Modelo de Datos:** Describe la información estática de la estructura en términos de entidades y relaciones.

Estructuras de Componentes y Conectores

- **Estructura de Servicio:** Servicios que operan entre ellos por medio de servicios de interoperación.
- **Estructura de Concurrencia:** Paralelismo y concurrencia.

Estructuras de Asignación

- **Estructura de despliegue:** Como el software se asigna a elementos de hardware.
- **Estructura de implementación:** Como el software se mapea en archivos.
- **Estructura de asignación de trabajo:** Mapeo de software a equipos de trabajo.

Táctica: Decisión de diseño que favorece un atributo de calidad.

Patrones de Módulos

- **Patrón de Capas**

Los módulos se dividen formando capas que se relacionan unidireccionalmente y cada una con su capa adyacente.

Esta estructura brinda modificabilidad y portabilidad. Pero penaliza la performance y agrega complejidad.

Patrones de Componentes y Conectores

- **Patrón Publicador-Suscriptor**

Elementos que tienen que comunicar a otros sin estar conscientes de su identidad.

El Publicador anuncia sus eventos al event-bus y el mismo los pone a disposición a los suscriptores quienes se suscribirán a los que necesiten recibir de ellos notificaciones.

Esto agrega latencia en la comunicación.
Menos control sobre el orden de mensajes y no se garantiza que los mensajes lleguen.

Tipos:

- Basado en Lista: Se tiene una lista de suscriptores a notificar
- Basado en Broadcast: Se disponibiliza los eventos y los suscriptores los chequean.
- Basado en Contenido: Se suscriben a eventos bajo tópicos de interés.

- **Patrón MVC**

Los módulos se reorganizan en un **Modelo** conteniendo los datos de la aplicación, una **Vista** que muestra parte de la información e interactúa con los usuarios, y un **Controlador** que es el intermediario entre los dos y maneja las notificaciones.

Para interfaces simples su complejidad puede no valer la pena.4

- **Patrón Orientado a Servicios**

Apunta a la interoperabilidad, dónde sistemas de distinta índoles puede comunicarse.
Ejemplos: SOAP, Rest, etc.

- **Patrón Multi-tier**

Puede ser un patrón de C&C o de Asignación.

Se agrupan en niveles lógicos bajo distintos criterios, restringiendo así el tipo de comunicación permitiendo interactuar sólo con el mismo nivel o con los adyacentes.

Favorece la seguridad, performance, modificabilidad, disponibilidad, pero es costoso y complejo de mantener.

Two-Tier (2 capas físicas) Client Tier contiene la lógica de presentación (interfaz de usuario) y la lógica de negocio o dominio. Database Tier representa otra máquina que hospeda la base de datos de la aplicación.

Three-Tier (3 capas físicas) Client Tier puede contener la lógica de presentación (interfaz de usuario) o puede ser un cliente fino (thin client): Browser. Web / App Tier contiene la lógica de presentación (web) y/o la lógica de dominio de la aplicación en un servidor independiente. Database Tier representa otra máquina que hospeda la base de datos de la aplicación.

N-Tier (4 o más capas físicas) Client Tier representa el Browser (thin client). Web Tier contiene la lógica de presentación (web) en un servidor independiente. Bussiness Logic Tier contiene la lógica de negocio de la aplicación en otro servidor. Database Tier representa otra máquina que hospeda la base de datos de la aplicación.

Pipe and Filter

Contexto: Algunos sistemas son requeridos para transformar cadenas de ítems de datos discretos de entradas a salidas. Algunos tipos de transformaciones ocurren repetidamente y entonces es deseable crearlas como partes reusables independientes.

Problema: sistemas que necesitan ser divididos en componentes de acoplamiento flexible reusable con mecanismo de interacción genérica, esto permite que sean combinados unos con otros. Los componentes que son de acoplamiento flexible son fáciles de reusar. Los componentes que son independientes se pueden ejecutar en paralelo.

Solución:

Este patrón es caracterizado por sucesivas transformaciones de cadenas de datos. Los datos llegan a los puertos de entrada, son transformados por los filtros, y son pasados por los puertos de salida por medio de los pipes. Un filtro simple consume datos de o desde uno o más puertos.

Repository

Contexto:

En muchas aplicaciones, la lógica de negocio tiene acceso a datos de los almacenes de datos como bases de datos o servicios Web. El acceso directo a los datos puede resultar en lo siguiente:

- Código duplicado
- Un mayor potencial de errores de programación
- Tipificación débil de los datos de negocio
- Dificultad para la centralización de las políticas relacionadas con los datos, tales como el almacenamiento en caché
- La incapacidad de probar fácilmente la lógica de negocio en forma aislada de las dependencias externas.

Problema:

- Usted desea maximizar la cantidad de código que se puede probar con la automatización y aislar la capa de datos para apoyar las pruebas unitarias.
- Acceder a la fuente de datos desde muchos lugares y gestionar los datos de forma centralizada, las reglas de acceso coherentes y lógicas.
- Usted desea implementar y centralizar una estrategia de almacenamiento en caché de la fuente de datos.
- Desea mejorar la mantenibilidad del código y la legibilidad separando la lógica de negocio de datos o lógica de acceso a servicios.
- Usted desea utilizar las entidades empresariales, establecimiento inflexible de tipos para que pueda identificar los problemas en tiempo de compilación en lugar de en tiempo de ejecución.
- Desea asociar un comportamiento con los datos relacionados. Por ejemplo, se desea calcular campos o hacer cumplir las complejas relaciones o reglas de negocio entre los elementos de datos dentro de una entidad.
- Usted desea aplicar un modelo de dominio para simplificar la lógica de negocio compleja.

Solución:

Utilice un repositorio para separar la lógica que recupera los datos y le asigna el modelo de entidad de la lógica de negocio que actúa sobre el modelo. La lógica de negocio debe ser transparente para el tipo de datos que comprende la capa de fuente de datos. Por ejemplo, la capa de origen de datos puede ser una base de datos, o un servicio web.

El repositorio media entre la capa de origen de datos y las capas de negocio de la aplicación. Se consulta el origen de datos para los datos, asigna los datos del origen de datos a una entidad comercial, y se persisten cambios en la entidad a la fuente de datos. Un repositorio separa la lógica empresarial de las interacciones con el origen de datos subyacente o servicio Web.