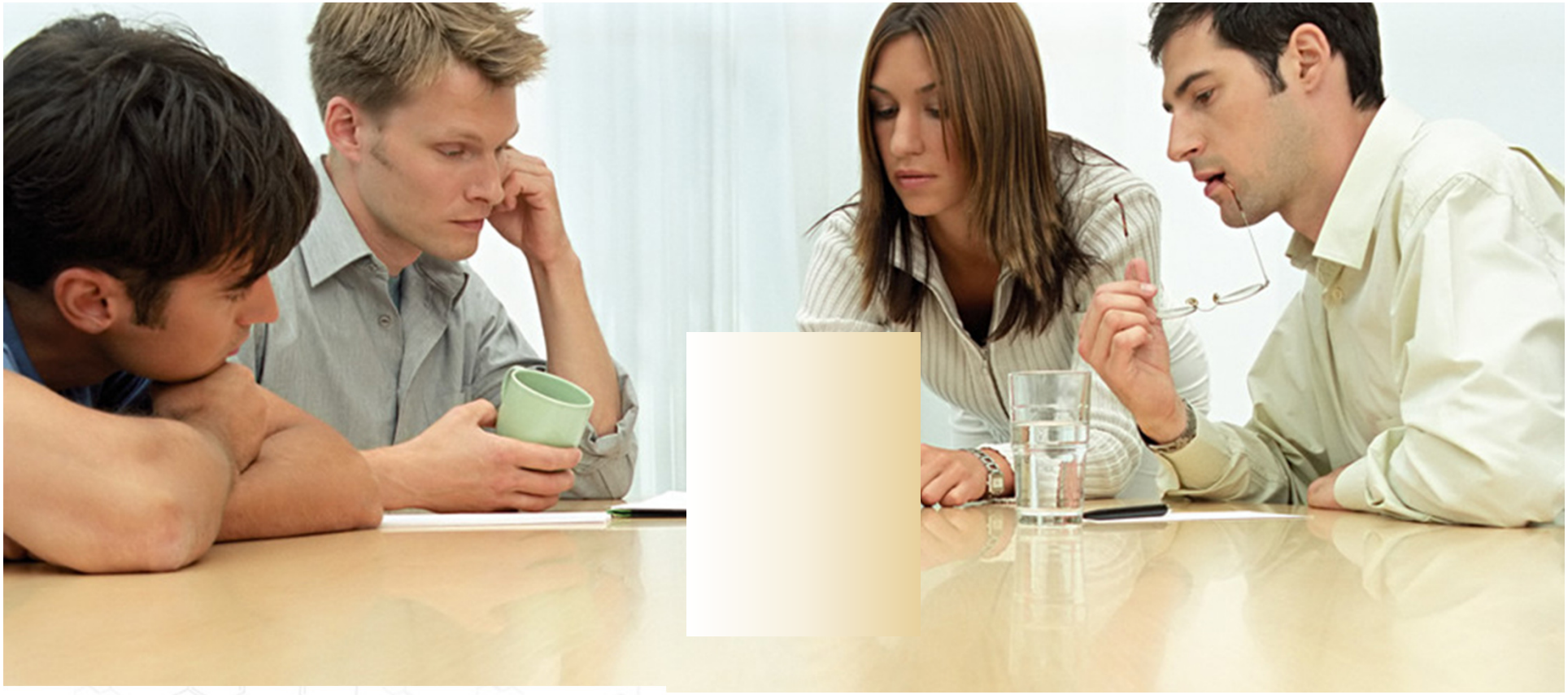


UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Arquitectura de Software

Administración de Recursos

Ing.Mariano Gecik

Arquitectura de Software

¿Que es la arquitectura de software?

Definición:

"La arquitectura de software de un sistema, define de manera abstracta, el conjunto de estructuras que la componen. En sí, son elementos de tecnología, relaciones y propiedades entre ellas."

Objetivos:

"Los sistemas de software son construidos para poder satisfacer los objetivos del negocio."

En que consiste:

"La arquitectura consiste en estructuras, y las estructuras contienen elementos y sus relaciones. Existen interacciones entre estos elementos de software. Es por esta razón, que la arquitectura trata de omitir ciertos detalles internos de cada uno de estos elementos, para abstraerse de su dificultad, y se ocupa más bien de lo exterior".

Interfaces → Dividen lo privado de lo público, se centra en la complejidad de la interacción de los elementos.

“NO TODAS LAS ARQUITECURAS SON BUENAS…”

¿Porqué la arquitectura de Software es importante?

1. El usuario, está pendiente de la rapidez, de la disponibilidad y de que el sistema sea confiable.

2. El cliente, está preocupado porque la arquitectura seleccionada sea implementada conforme al calendario y al presupuesto acordados.

3. El Project Manager, se preocupa, de que la arquitectura permita que los equipos trabajen en forma independiente interactuando a su vez con disciplina.

4. El arquitecto, está preocupado porque todos los puntos anteriores funcionen correctamente y en forma sincronizada.

¿Quiénes son los interesados en una Arquitectura de Software?

1. Clientes.
2. Usuarios.
3. Project Managers.
4. Arquitectos.
5. Desarrolladores.
6. Testers.
7. Subsistemas…
8. Y más…

Decisiones de diseño de la arquitectura que se deben tener en cuenta:

1. ¿El sistema deberá correr en un procesdor o debe ser distribuido en más de uno?.

2. ¿El software deberá ser dividido en capas? ¿Si es así, en cuántas capas? ¿Que debería hacer cada capa?

3. ¿Los componentes deberían comunicarse sincrónica o asincrónicamente?

4. ¿El sistema depende de algunas características del sistema operativo y/o del hardware?

CONTEXTO

- Técnico: ¿Qué roles técnicos ocupa la arquitectura de software en el sistema o sistemas?.
- Ciclo de vida del proyecto: ¿Cómo se relaciona la arquitectura de software con otras fases del ciclo de vida de desarrollo? (“Waterfall”, “Iterativo”, “Agile”, “Model-Driven Development”).
- Negocio: ¿Cómo afecta la arquitectura de software al entorno de negocio de la organización?.
- Profesional: ¿Cuál es el rol de la arquitectura de software en la organización en el desarrollo del proyecto?.

“A pesar de que estos contextos no cambian, las especificaciones del sistema cambian constantemente. Uno de los desafíos del arquitecto es prever que estos cambios no repercutan en gran medida sobre los sistemas”.

Actividades importantes en la creación de la Arquitectura de Software:

1. Crear casos de negocio para el sistema.
2. Entender, los requerimientos importantes para la arquitectura.
3. Crear o seleccionar una arquitectura.
4. Documentar y comunicar la arquitectura.
5. Analizar y evaluar la arquitectura.
6. Implementar y testear el sistema, basándose en la arquitectura.
7. Asegurarse que la implementación conforme a la arquitectura elegida.

Arquitectura de Software

Atributos de Calidad

Atributos de Calidad

"Un atributo de calidad es una propiedad de medida o de testeo, que permite indicar que tan bien funciona un sistema y cómo satisface éste, las necesidades de los interesados".

Requerimientos

Existen distintos tipos de requerimientos:

Requerimientos Funcionales: Estos definen, qué tiene que hacer el sistema, y cómo debe reaccionar o actuar en tiempo de ejecución.

Requerimientos de calidad del sistema: Estos definen, las características de los requerimientos funcionales, a lo largo de todo el producto. Un ejemplo de este, podría ser, que tan rápido debería realizarse un requerimiento funcional.

Restricciones: La restricción es una decisión de diseño que debe tomarse. Un ejemplo podría ser, un lenguaje de programación elegido, la reutilización de un módulo, etc.

Orientar las decisiones de diseño de calidad

La arquitectura puede verse como el resultado de aplicar un conjunto de decisiones de diseño. La forma de categorizar estas decisiones, que pueden llegar a generar algunos inconvenientes a la hora de la construcción de la misma, pueden ser divididas en 7 secciones:

1. Asignar responsabilidades.
2. Modelo de coordinación.
3. Modelo de datos.
4. Gestión de recursos.
5. Mapeo entre los elementos de la arquitectura.
6. Decisiones sobre los tiempos.
7. Elección de la tecnología.

DISPONIBILIDAD

"Fundamentalmente, la disponibilidad tiene que ver con minimizar las interrupciones de servicio y mitigar las posibles fallas que puedan ocurrir".

"Las pérdidas significan desviaciones de las especificaciones del sistema, dónde esta desviación es externamente visible. La idea de todo esto es entender, cuáles fueron las causas de los problemas rápidamente, para tener una estrategia de mitigación diseñada dentro del software".

Tácticas de Disponibilidad

Las tácticas de disponibilidad se pueden dividir en 3 secciones:

1. Detectar fallas
2. Recuperación de fallas
3. Prevención de fallas

INTEROPERABILIDAD

"La interoperabilidad significa que 2 o más sistemas pueden intercambiar información importante vía interfaces en un contexto particular. No solo esto, sino también poder comprender la información que se está intercambiando".

"Si nosotros conocemos las interfaces de los sistemas externos, donde nuestros sistemas operan, entonces podemos diseñar este conocimiento, dentro de los propios sistemas".

Y también existen 2 tipos de tácticas para emplear:

Locate**:** Existe una sola táctica en esta categoría, "discover service". Se utiliza cuando los sistemas que operan deben ser descubiertos en tiempo de ejecución.

Manage Interfaces: Existen 2 tácticas. "Orchestrate" y "tailor interface". La primera radica en utilizar un mecanismo de control que coordina, gestiona y secuencia la invocación de servicios particulares. Con respecto a "Tailor interface", es una táctica, que agrega o elimina capacidades en una interface.

ADAPTABILIDAD

El concepto de adaptabilidad tiene que ver con el cambio, y nuestro interés se debe centrar en el costo y en el riesgo de realizar estos cambios.

Para poder planificar estos cambios, el arquitecto debe considerar los siguientes puntos:

1. ¿Qué puede cambiar?.
2. ¿Cuál es la probabilidad del cambio?.
3. ¿Cuándo el cambio se debe realizar y quien lo hará?.
4. ¿Cuál es el costo del cambio?.

PERFORMANCE

La Performance se refiere a **tiempo**, y a la **habilidad** que tienen los sistemas para contestar los requerimientos en **tiempo y forma**. Cuando un evento ocurre (por ejemplo, interrupciones, mensajes, peticiones de usuarios y de otros sistemas, o eventos de reloj) el sistema o los elementos de un sistema, deben responder a ellos en tiempo.

Podemos entonces dividir, nuestras tácticas de performance, en 2 categorías:

Demanda del control de recursos. Esta táctica opera en el lado de la demanda, para producir menos demanda de los recursos que deben servir a los eventos.

Gestión de recursos. Esta táctica opera en el lado de la respuesta para que los recursos trabajen más eficientemente en gestionar las demandas que se suscriben sobre ellos.

SEGURIDAD

“La seguridad es una medida de habilidad que tienen los sistemas, para proteger datos e información de un acceso **no autorizado**, mientras se provee de acceso a gente y sistemas que **sí** están autorizados”.

Las tácticas de seguridad son las siguientes:

Detectar ataques: Las categorías para detectar ataques consisten en 4 tácticas: la detección de intrusos, la detección de la denegación de un servicio, la verificación de integridad de mensajes, y la detección del atraso de mensajes.

Resistir ataques: Existen un número de medios bien conocidos para resistir a un ataque. Identificar actores: Por ejemplo identificación de usuarios mediante ID, direcciones IP, etc.

Autenticación de actores.
Límite de acceso.
Encriptación de datos.

SEGURIDAD

Reaccionar ante un ataque: Algunas tácticas tienen intención de responder a potenciales ataques, por ejemplo:

Revocar un acceso: Si un sistema o administrador cree que puede originarse un ataque, entonces se limita el acceso notablemente, a pesar de que sea sobre usuarios legítimos.

Lockeo de una computadora: Si existen varios logueos fallidos puede que se esté indicando un posible ataque. Muchos sistemas prohíben ciertas PC si este tipo de acciones ocurre. Por ejemplo, un acceso a una cuenta bancaria.

Recuperación frente a ataques: Cuando un sistema detecta y atenta a resistir un ataque, éste tiene que recuperarse. Parte de esta recuperación tiene que ver con restaurar los servicios dañados. Por ejemplo, servicios adicionales o conexiones de redes son reservados para este propósito.

CAPACIDAD DE PRUEBA Y TESTEO

”La industria indica en una estimación, que entre el 30 y el 50 por ciento (o quizá más) del costo de una buena ingeniería en el desarrollo de los sistemas es absorbida por las pruebas. Si la arquitectura de software puede reducir este costo, el pago será mayor”.

“La capacidad de prueba se refiere, a la facilidad con la que el software puede hacer para demostrar sus fallos a través de pruebas. Específicamente, esta capacidad se refiere a la probabilidad, asumiendo que el software va a fallar al menos una vez, y que fallará también en la próxima ejecución de testeo”.

El objetivo de la táctica de la capacidad de prueba, es permitir un testeo sencillo cuando se completa un incremento del desarrollo de software.

Existen 2 tipos de categorías sobre las tácticas de testeo:

1. Las primeras tienen que ver con controlar y observar al sistema.
2. Las segundas, con limitar la complejidad del diseño del sistema.

USABILIDAD

La usabilidad está interesada en cuán fácil es para el usuario, ejecutar una tarea deseada y cuál es el tipo de usuario que soporta el sistema.

Por varios años, un enfoque sobre la usabilidad ha demostrado ser una de las formas más baratas y más fáciles de mejorar la calidad de un sistema.

La usabilidad comprende las distintas áreas:

Aprendizaje de las características de un sistema.

Utilización eficiente del sistema.

Minimizar el impacto de errores.

Adaptar el sistema a lo que necesita el usuario.

Incrementar la confianza y la satisfacción.

Iniciativa de Apoyo a los usuarios

Cuando el sistema es ejecutado, se mejora la facilidad de uso brindando feedback de los usuarios de que es lo que el sistema está haciendo y habilitando al usuario a tener respuestas apropiadas.

Por ejemplo, las tácticas de "**cancel**", "**undo**", "**pause**", "**resume**" y "**aggregate**" ayudan al usuario a corregir los errores y ser más eficiente.

El arquitecto, diseña las respuestas por iniciativa del usuario, mediante la enumeración y la asignación de las responsabilidades del sistema para responder a la orden del usuario.

Iniciativa de Apoyo de sistema

Cuando el sistema toma la iniciativa, debe basarse en un modelo del usuario, donde la tarea que se lleva a cabo, es por el usuario o el propio estado del sistema. Cada modelo requiere de distintos tipos de input, para lograr este tipo de iniciativas. Las tácticas de iniciativa de apoyo del sistema, son las que identifican los modelos que utilizan los sistemas para predecir su propio comportamiento, o la intención del usuario.

Arquitectura de Software

Otros Atributos de Calidad

OTROS ATRIBUTOS DE CALIDAD

"En las slide anteriores, hemos observado distintos atributos de calidad que merecen un detalle mayor, por la implicancia que estos tienen en el armado de una arquitectura. Sin embargo, además de estos, existen otros atributos que merecen al menos, una breve explicación".

Variabilidad = Adaptación al contexto.
Portabilidad = Que pueda realizar cambio de plataforma.
Desarrollo Distribuido = Diseño de soft que soporta el desarrollo de SD.
Escalabilidad = Agregar más recursos como un server o más memoria.
Capacidad de Ejecución = Como los ejecutables llegan a la plataforma de host y como son invocados.
Movilidad = Tamaño, tipo de visualización, tipo de dispositivo de entrada.
Monitoreo = Monitorear el sistema mientras esta trabajando.
Seguridad = Evitar entrar en estados que causan o provocan daños.
Integridad Conceptual = Consistencia en el diseño de la arquitectura.
Comerciabilidad = No siempre se adaptan a lo que necesitamos.

Arquitectura de Software

Tácticas de Arquitecturas y Patrones

Tácticas de Arquitectura y Patrones

"Existen varias maneras de realizar un mal diseño, y muy pocas de hacerlo bien. Tener éxito en el diseño de la arquitectura es complejo y cambiante, es por ello que los diseñadores han estado buscando maneras de capturar una reutilización del conocimiento arquitectónico".

"Los patrones y tácticas de arquitectura son los caminos para realizar un buen diseño de estructuras que pueden ser reutilizadas".

Un patrón de arquitectura:

1. Es un paquete de decisiones de diseño, que se puede encontrar repetidamente en la práctica.

2. Conoce propiedades que permiten reutilización.

3. Describe una clase de arquitectura.

Patrones de Arquitectura

Los patrones de arquitectura establecen relaciones entre:

El contexto: Una situación común que se repite en el mundo que da lugar a un problema.

El problema: El problema, apropiadamente generalizado, que surge en el contexto dado. La descripción del patrón describe el problema y sus variantes, y describe las fuerzas complementarias u opuestas. La descripción del problema a menudo incluye atributos de calidad que deben cumplirse.

La solución: Una resolución de arquitectura exitosa al problema, resumido adecuadamente. La solución se describe por las estructuras arquitectónicas que resuelven el problema, incluyendo la forma de equilibrar las fuerzas en el trabajo. En la solución se describen las responsabilidades.

Patrones

“La aplicación de un patrón no es una decisión de todo o nada. Las definiciones de patrón que se da en los catálogos, son estrictas, pero en la práctica, los arquitectos pueden optar por modificarlos en pequeños elementos cuando hay una buena solución de compromiso de diseño que se tenía”.

“Por ejemplo, el patrón de capas prohíbe expresamente software en capas más bajas de uso de software en capas superiores, pero puede haber casos (por ejemplo, para ganar algo de rendimiento) cuando una arquitectura podría permitir algunas excepciones específicas”.

“Los patrones pueden ser categorizados por el dominio de los tipos de elementos que éstos muestran: los patrones de módulos muestran módulos, los patrones de conectores y componentes muestran eso mismo, y los patrones de asignación, muestra la combinación de elementos de software (módulos, componentes, conectores) y elementos que no son del software”.

“Patrones de módulos” – “Layered Pattern”

Contexto: Todos los sistemas complejos experimentan la necesidad de desarrollar y evolucionar partes del sistema de forma independiente. Por esta razón, los desarrolladores del sistema necesitan una separación clara y bien documentada de las preocupaciones, por lo que los módulos del sistema pueden ser desarrollados y mantenidos de forma independiente.

Problema: El software debe ser segmentado de tal manera que los módulos pueden ser desarrollados y evolucionado por separado con poca interacción entre las partes, el apoyo a la portabilidad, adaptabilidad, y la reutilización.

Solución: Para lograr esta separación de las preocupaciones, el patrón de capas divide el software en unidades llamadas capas. Cada capa es una agrupación de módulos que ofrece un conjunto cohesivo de servicios. Hay limitaciones en la relación-permitida de usar entre las capas: las relaciones deben ser unidireccional. Las capas particionan completamente, un conjunto de software, y cada una de esas particiones, se expone a través de una interfaz pública.

“Patrones Componente - Conector” – “Broker Pattern”

Contexto: Algunos sistemas son construidos bajo una colección de servicios distribuidos sobre múltiples servers. La implementación de estos sistemas es complejo, porque uno debe preocuparse de cómo los sistemas deben interoperar (es decir, cómo deben conectarse unos con otros, y como deben intercambiar información), teniendo en cuenta la disponibilidad de los servicios de los distintos componentes.

Problema: Depende de cómo distribuimos el software los servicios de usuario no necesitan saber la locación y naturaleza de los proveedores de servicio, para que se haga dinámico el intercambio de información entre usuarios y proveedores.

Solución: El “Broker Pattern” separa los servicios de usuarios (clientes) de los servicios de los proveedores (servidores) insertando un intermediario que es denominado “Broker”. Cuando un cliente solicita un servicio, este consulta al Broker vía un servicio de interfaz. Luego el Broker reenvía el servicio enviado por el cliente al servidor del proveedor, donde es procesada la solicitud. El resultado del servicio es comunicado nuevamente del servidor al Broker, que luego retornará el resultado (y si hay excepciones) al cliente que había solicitado el servicio.

“Patrones Componente - Conector” – “Model-View-Controller (MVC) Pattern”

Contexto: El software de interfaz de usuario, es la porción de las aplicaciones interactivas que más frecuentemente son modificadas. Es por esta razón, que las modificaciones sobre dicha interfaz, deben quedar separadas del resto del sistema. Los usuarios comúnmente desean ver la información desde distintas perspectivas, como puede ser un gráfico de barras, o un gráfico de tortas. Ambas representaciones reflejan el estado actual de la información.

Problema: ¿Cómo la funcionalidad de la interfaz de usuario puede mantenerse separada de la funcionalidad de la aplicación, y seguir siendo responsable de los inputs del usuario o de modificar los datos que maneja dicha aplicación? ¿Y cómo múltiples vistas de la interfaz de usuario pueden ser creadas, mantenidas y coordinadas cuando la información de la aplicación cambia?.

Solución: El patrón modelo-vista-controlador (MVC) separa la funcionalidad de la aplicación en 3 componentes distintos:
El modelo, que contiene la información de la aplicación.

“Patrones Componente - Conector” – “Model-View-Controller (MVC) Pattern”

El MVC no es apropiado para cualquier situación. El diseño y la implementación de estos 3 tipos de componentes, a través de varias formas de interacción, suelen tener un costo, y ese costo no tiene mucho sentido, en interfaces simples de usuario.

“Patrones Componente - Conector” – “Client-Server Pattern”

Contexto: Existen recursos y servicios que deben ser compartidos y distribuídos a un número grande de clientes que quieren acceder, y es por esto que queremos manejar el control de acceso y el manejo de la calidad del servicio.

Problema: Mediante la gestión de un conjunto de recursos y servicios compartidos, podemos promover la adaptabilidad y reutilizar, los servicios comunes y tener que solo modificarlos en un solo lugar, o un pequeño número de lugares. Queremos mejorar la escalabilidad y adaptabilidad centralizando el control de estos recursos y servicios, distribuyendo los recursos mediante múltiples servidores físicos.

Solución: Los clientes interactúan solicitando servicios a los servers, donde se proveen un conjunto de estos servicios. Algunos componentes pueden interactuar tanto como clientes como también servidores. Puede haber, un servidor central, o un conjunto múltiple distribuido.
Los tipos de componentes son clientes y servidores. El tipo de conector principal en este patrón es, un conector de información que utiliza un protocolo de petición / respuesta utilizado para invocar servicios.
Algunas de las desventajas de este patrón, radica en que los servidores pueden tener un cuello de botella y dificultar su performance

“Patrones Componente - Conector” – “Service-Oriented Architeture Pattern”

Contexto: Un número de servicios son ofrecidos por proveedores de servicios y consumidos por servicios de consumidores. Estos últimos, deben entender y utilizar estos servicios, sin tener conocimiento y detalle de como son implementados.

Problema: ¿Cómo soportamos la interoperabilidad de componentes distribuidos en tiempo de ejecución, en diferentes plataformas, escritos en diferentes lenguajes de implementación, proveídos por diferentes organizaciones y distribuidos por Internet? ¿Cómo alojamos servicios y los combinamos en distintas uniones ejecutando una buena performance, una buena seguridad y adaptabilidad?.

Solución: El patrón, servicio de arquitectura orientado (SOA) describe una colección de componentes distribuidos que proveen y/o consumen servicios. En SOA, los componentes de proveedor de servicios y los componentes de consumidor de servicios utilizan distintos lenguajes de implementación y plataformas. Los servicios son en gran parte independientes: los servicios del proveedor y los servicios del consumidor usualmente se ejecutan separados y provienen en general de distintas organizaciones y sistemas.

“Patrones de alocación” – “Map-Reduce Pattern”

Contexto: Los negocios necesitan poder analizar grandes volúmenes de información que ellos generan o acceden. Por ejemplo, accediendo a un sitio de una red social, documentos masivos, o repositorios de información, y a unos cuántos links dentro de un motor de búsqueda. Los programas que analizan estos datos deberían ser sencillos de escribir, deberían ejecutarse eficientemente, y ser fuertes frente al fallo del hardware.

Problema: Para muchas aplicaciones que manejan un gran volumen de información, es suficiente con ordenar, agrupar y analizar dicha información. El problema que éste patrón resuelve, es llevar a cabo, de manera eficiente, el proceso de un gran conjunto de datos distribuidos y paralelos y proporcionar un medio simple para que el programador pueda especificar el análisis que se debe hacer.

Solución: Este patrón requiere 3 partes: Primero, una infraestructura especializada, que tenga cuidado para alocar el software en los nodos de hardware en un entorno paralelo y masivo de procesamiento de cálculo, y que gestione la información que se requiera en forma ordenada. Segundo y tercero, son 2 funciones de código de programador, que son denominados “mapeo“ y “reducción”.

“Atributos de Calidad de Modelado y Análisis”

“El objetivo de experimentos, simulaciones, y prototipos, es de proveer alternativas de análisis sobre la arquitectura. Estas técnicas son de gran valor en la resolución de compensaciones, ayudando a convertir parámetros arquitectónicos desconocidos en constantes o rangos”.

Por ejemplo, considerar sólo algunas de las preguntas, que pueden surgir al crear una infraestructura de cliente-servidor distribuido, con restricciones de tiempo real de un sistema de conferencias web:

¿Convendría mudar a una base de datos distribuida los archivos locales o impactaría negativamente el tiempo de realimentación (latencia) para los usuarios?.

¿Cuántos participantes podrían ser hosteados por un solo servidor de conferencias?.

¿Cuál es el ratio correcto entre servidores de base de datos y servidores de conferencias?.

“Análisis en diferentes etapas del ciclo de vida”

“Cuanto más se avanza en el análisis, los resultados de rendimientos del ciclo de vida merecen una confianza mayor. Sin embargo, esta confianza tiene un precio. En primer lugar, el costo de realizar el análisis también tiende a ser mayor, por lo tanto descubrir un problema en etapas posteriores en el ciclo de vida, radica en un aumento del costo”.

Es importante entonces, elegir la mejor forma de realizar un análisis, considerando varios factores (experiencias pasadas, checklists, modelos analíticos, simulaciones, prototipos, experimentos, etc).

¿En qué etapa del ciclo de vida estamos?.
¿Qué tan importante son los atributos de calidad en la etapa?.
¿Qué preocupado uno está en la ejecución de esos atributos?.
¿Y por último…Cuanto más presupuesto y planificación necesitamos para ejecutar esta forma de mitigación del riesgo?.

Debemos lidiar entonces, con todas estas consideraciones, y elegir las mejores técnicas de análisis ,para el ciclo de vida.

Arquitectura de Software

ADS en el Ciclo de Vida

“La Arquitectura en Proyectos ágiles”

En los últimos años el desarrollo de software ha cambiado notablemente. Los métodos y los procesos se han agilizado, y por lo tanto, los proyectos han tenido que cambiar.

1. Responsabilidad sobre a las personas que están involucradas.
2. Realizar un desarrollo más rápido de su funcionalidad.
3. Mostrar un progreso más rápido en el ciclo de vida.
4. Realizar una documentación justa y necesaria.

Puntos Interesantes de este tipo de Proyectos:

- Poner una prioridad alta en la satisfacción del cliente cuando se entrega una versión al cliente.
- Si cambian los requerimientos, aunque sea tarde en el desarrollo, bienvenidos sean. Los procesos ágiles, aprovechan los cambios, para obtener ventajas competitivas del cliente.
- Las entregas de software trabaja con una frecuencia que puede ir desde un par de semanas a un par de meses, con una preferencia a la escala de tiempo más corto.
- La gente de negocio, y los desarrollos, deben trabajar conjuntamente a lo largo del proyecto.

- La construcción de proyectos debe motivar a los individuos. Les tienen que dar un entorno y el soporte necesario, y confiar en que ellos puedan hacer un buen trabajo.
- El método más efectivo y eficiente para que mejore el intercambio de información, son conversaciones, cara a cara.
- Trabajar en el software, es la primera medida de progreso.
- Los procesos ágiles promueven desarrollo sustentable.
- Atención continúa en la excelencia técnica y buenas decisiones en el diseño, mejoran la agilidad.
- Simplicidad. El arte de maximizar la cantidad de trabajo no es buena. Esto es esencial.
- Las buenas arquitecturas, requerimientos, y diseños, emergen, de los propios equipos técnicos de las organizaciones.
- En intervalos regulares, el equipo debe mostrar cómo se puede ser más efectivo, y ajustar los comportamientos acordemente a ello.

“El movimiento de software ágil está basado en el concepto de “ágil” y un conjunto de principios, que asignan un gran valor a los equipos de trabajo. Los procesos ágiles se emplearon inicialmente en pequeños y medianos proyectos de cortos periodos de tiempo y disfrutaron de un éxito considerable. Ellos no eran de uso frecuente para proyectos más grandes, particularmente aquellos con desarrollo distribuido”.

“Aunque podría parecer ser que existen grandes diferencias entre las prácticas ágiles y las “comunes”, las esencias de ambas no son tan distintas, y muchas veces los proyectos de éxito necesitan una mezcla de los dos enfoques. Demasiada planificación y compromiso por adelantado, puede ser hasta “sofocante” y no responde a las necesidades del cliente, mientras que demasiada agilidad simplemente puede resultar en caos. Los Arquitectos de metodologías ágiles tiende a tener un término medio, proponiendo una arquitectura inicial hasta que se vuelva el proyecto demasiado grande, entonces en ese momento, modificarla o combinarla con una arquitectura que no sea del estilo “ágil”.

"Los Requerimientos y la Arquitectura"

"Los requerimientos de arquitectura significativos (ASR) son aquellos que tienen un efecto profundo sobre la arquitectura, es decir, la arquitectura bien podría ser diferente en ausencia de tal requisito".

"No podemos esperar, diseñar una arquitectura satisfactoriamente sino conocemos los ASR".

"Los ASR generalmente (aunque no siempre) dan forma a los requerimientos de atributos de calidad como ser (la performance, la seguridad, la adaptabilidad, la disponibilidad, la usabilidad, etc.), que la arquitectura provee a un sistema".

"Los arquitectos deben identificar a los ASR, usualmente lo hacen luego de trabajar y descubrir cuáles pueden ser los candidatos apropiados. Para ellos, lo primero que hacen es dialogar con los interesados más importantes de la arquitectura".

- Reunir ASRs de los documentos de requerimientos
- Reunir ASRs entrevistando a los interesados
- Reunir ASRs entendiendo los objetivos de Negocio

Evaluación de una Arquitectura

"El análisis se encuentra en el corazón de una evaluación en la arquitectura, y es el proceso de determinar si una arquitectura es apta para el propósito para el cual está destinada".

"La arquitectura es parte fundamental para el éxito de un proyecto de ingeniería de sistemas y software, que se utiliza para tratar de asegurar que la arquitectura que se ha diseñado, será capaz de proporcionar todo lo que se espera de ella".

En general, la evaluación utiliza una de estas formas:

1. Evaluación por el diseñador, que diseña los procesos.

2. Evaluación por pares dentro del proceso de diseño.

3. Análisis por personas independientes una vez que la arquitectura fue diseñada.

Arquitectura de Software

Gestión y Gobierno

Evaluación de una Arquitectura

"Los aspectos de la gestión de proyectos y la gobernabilidad son importantes de saber para un arquitecto".

"El Project manager, es la persona que junto al arquitecto, deben trabajar en conjunto, por la perspectiva de la organización".

"La arquitectura es más útil en proyectos de mediana y gran escala, que típicamente tienen varios equipos, demasiada complejidad para cualquier individuo, y duración de varios años",

Planificación

"La planificación en un proyecto sucede todo el tiempo. Existe un plan inicial, de arriba hacia abajo, que es necesario implementar para convencer a la alta dirección, de construir un sistema y dar una idea de su costo y agenda".

"El Project manager debe educar a los otros managers para poder corregir los desvíos en el desarrollo de software".

Organización

"Algunos de los elementos de la organización de proyectos, son la organización de equipos, división de responsabilidades entre el Project manager, y el arquitecto de software, y la planificación global o distribuida del desarrollo".

"Una vez que el diseño de la arquitectura, esta ok, es utilizada para definir el proyecto de organización".

"La responsabilidad de dar contenido a e implementar el diseño se repartió a los que tuvieron un papel en su definición".

Los roles típico, del equipo de desarrollo del software son los siguientes:

Team Leader – Gestionan las tareas en el equipo.
Developer – Diseñan e implementan los subsistemas de código.
Configuration Manager – Ejecutan y construyen test de integración.
System test manager – Testeo de sistema, y testing de aceptación.
Product Manager – Representan el marketing. Definen características y como el Sistema debe ser integrado, con otros sistemas en una "suite" de productos.

Métodos de coordinación

- Contactos informales. Como reuniones en un sala de café, o en el pasillo, solo serán posibles si todo el equipo trabaja en el mismo lugar.
- Documentación. Si está bien escrita, bien organizada, y bien diseminada, se utilizará para coordinar equipos que están alojados en el mismo lugar, o bien en forma remota.
- Reuniones. Los quipos pueden realizar reuniones, planificadas o a propósito, y también cara a cara, o remota, para ayudar a unir al equipo y crear conciencia sobre cuestiones puntuales.
- Comunicación electrónica. Varias formas de este tipo de comunicaciones pueden utilizarse, como un mecanismo de coordinación, tal como el email, los grupos, los blogs y las wikis.

Implementación

"Durante la fase de implementación de un proyecto, el Project manager, y el arquitecto, tienen que tomar decisiones importantes. A continuación hablaremos de las compensaciones, el desarrollo incremental y la gestión del riesgo".

"Estos son los aspectos del proyecto que son importantes para los grupos de interés externos, y los grupos de interés externos son elegidos por el jefe de proyecto. ¿Cuál de estos aspectos es más importante? Depende del contexto del proyecto, y una de las principales responsabilidades del gerente de proyecto, es hacer elección".

"Finalmente, el Project manager debe priorizar los riesgos. Frecuentemente éste realiza esa tarea con el arquitecto, y para la mayoría de los riesgos altos, se debe desarrollar una estrategia de mitigación. Esta estrategia, produce un costo, e implementar una estrategia para reducir los riesgos depende de una prioridad, y de una probabilidad de ocurrencia, y del costo si este realmente ocurre".

El análisis económico de las Arquitecturas

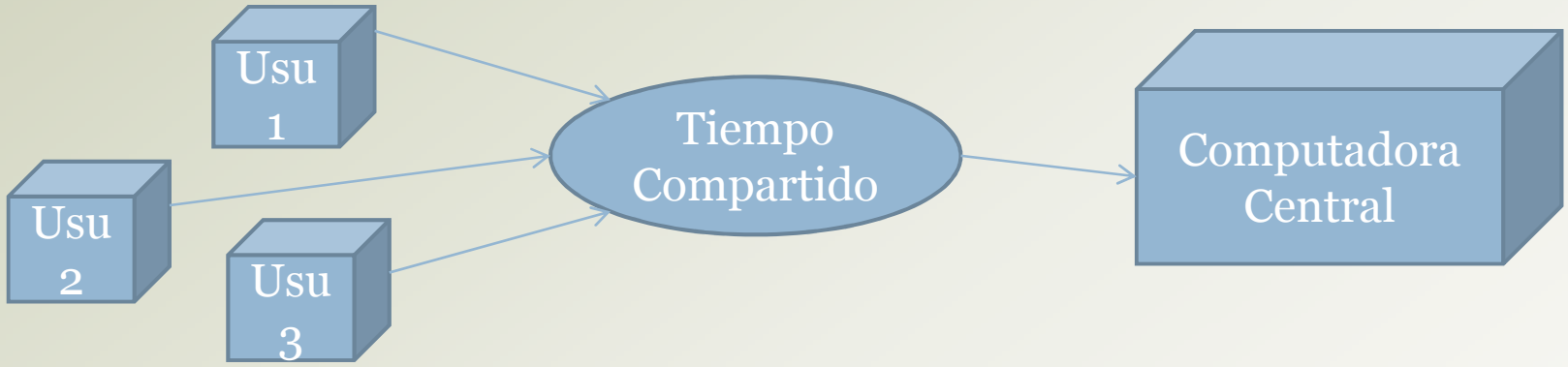
“El Análisis económico basado en la arquitectura, tiene en cuenta la comprensión de la curva de utilidad de respuesta, de los distintos escenarios y realiza una comparación entre ellos. Una vez que están en esta forma común - sobre la base, de la moneda común de utilidad - la VFC para cada mejora la arquitectura, analiza el escenario relevante y puede de esta forma calcularlo y compararlo”.

“La aplicación de la teoría en la práctica, tiene una serie de dificultades, pero a pesar de esas dificultades, creemos que la aplicación de las técnicas económicas es inherentemente mejor que los enfoques de toma de decisiones ad hoc que los proyectos emplean hoy”.

Arquitectura de Software

La arquitectura en la Nube

La arquitectura en la nube



"En realidad, muchas de las técnicas, como la virtualización, que es utilizada en la nube hoy en día, también eran en aquellos momentos. Cada usuario de la aplicación de un CLOUD, no tiene que saber dónde se encuentra esa aplicación y la información, y de hecho, pueden encontrarse en distintas bandas horarias y donde miles de usuarios comparten lo mismo".

"Por supuesto que con la llegada de Internet, la disponibilidad de una o muchas computadoras potentes hoy en día, el requisito de uso compartido controlado, y el diseño de la arquitectura de una aplicación basada en la nube, es muy diferente de diseñar, a la arquitectura para una aplicación basada en "tiempo compartido". Sin embargo, las fuerzas impulsoras, siguen siendo prácticamente la misma".

Definiciones básicas del Cloud

1. Servicio a demanda: Consumidor utiliza servicio, según necesite de forma automática.

2. Acceso único de red: Los servicios y recursos cloud se utilizan heterogéneamente entre todos los clientes.

3. Pool de recursos: Los recursos se agrupan, para que el proveedor pueda brindar a los consumidores acorde a la demanda.

4. Independencia de ubicación: Locación independiente del acceso a red, a veces puede producir latencia.

5. Elasticidad rápida: Puesta en común de recursos, brinda capacidades rápidas de escalar rápidamente hacia afuera o hacia dentro del cloud(parece ilimitada).

6. Servicios medidos: Los sistemas cloud automáticamente controlan y optimizan los recursos. La utilización de los mismos puede ser monitoreada, controlada y reporteada, para que los consumidores de los servicios, solo sean facturados según su uso.

Modelos de Servicios y Opciones de Desarrollo

1. El software como un servicio (**SaaS**).

- El consumidor en este caso es el usuario final.
- Los consumidores utilizan las aplicaciones que se están en el cloud.
- Las aplicaciones pueden ser, email, calendarios, streaming de video, etc.
- El consumidor no gestiona o controla la infraestructura del cloud. (incluyendo la red, los servidores, los sistemas operativos, el almac). en base a sus propio seteo de configuración.

2. La plataforma como un servicio (**PaaS**).

- El consumidor en este caso es desarrollador o administrador del sistema.
- La plataforma provee una variedad de servicios que el consumidor puede elegir usar.
- Estos servicios pueden incluir, opciones de base de datos, balanceo de carga, disponibilidad, y entornos de desarrollo.
- El consumidor desplegará aplicaciones en la infraestructura del cloud, utilizando lenguajes de programación y herramientas soportadas por el proveedor.
- El consumidor no controlará ni gestionará la infraestructura del cloud, pero sí tendrá bajo control sus aplicaciones y otras aplicaciones que se configuren en el entorno.

Modelos de Servicios y Opciones de Desarrollo

3. La infraestructura como servicio (**IaaS**).

- El consumidor en este caso es desarrollador o administrador del sistema.
- La capacidad que tiene el consumidor es de proveer, procesamiento, almacenamiento, networking, y cualquier otro recurso fundamental donde el consumidor, es capaz de desplegar y ejecutar software arbitrario que puede incluir tanto a los sistemas operativos como a las aplicaciones.
- El consumidor por ejemplo puede elegir, crear una instancia de una computadora virtual y proveer versiones de Linux específicas.
- El consumidor no gestiona ni maneja la infraestructura del cloud, pero tiene el control de los sistemas operativos, el almacenamiento, las aplicaciones desplegadas, y posiblemente pueda limitar el control de los componentes de redes (como ser un firewall).

Modelos de Desarrollos

"Los modelos de desarrollo del cloud se diferencian por quienes son dueños y quienes lo operan".

"Es posible que el cloud pertenezca a una parte, y sea operado por otra. Pero nosotros asumiremos que el que tiene el cloud también lo opera".

Existen 2 modelos básicos que tienen 2 variantes.

Cloud privado. La infraestructura de cloud es comprado por una organización y operado solamente por aplicaciones compradas por esa organización. El principal objetivo de la organización no es vender servicios de cloud.

Cloud público. La infraestructura del cloud está disponible para el público, o para un gran grupo de una industria, y es comprada por la organización para vender servicios de cloud.

Justificación Económica

A continuación, citaremos brevemente, 3 distinciones económicas, entre los data centers “cloud”, basándose en sus tamaños, y la tecnología, ellos utilizan:

Economía de escala

Los grandes centros de datos, son inherentemente menos caros de operar por unidad de medida, (como ser el costo por gigabyte) que los centros de datos más pequeños. Los centros de datos de gran tamaño pueden tener cientos de miles de servidores (costo de energía, infraestructura, hardware, seguridad).

Utilización de equipamiento

La práctica común en centros de datos no virtualizados, es ejecutar una aplicación por servidor, esto es causado por la dependencia de muchas aplicaciones empresariales en sistemas operativos particulares o incluso versiones particulares de estos sistemas operativos. Un resultado de la restricción de una aplicación por servidor es, una gran baja de utilización del servidor.

Multi-tenencia

Las múltiples aplicaciones de clientes, tienen una arquitectura explícita, que soportan solo aplicaciones con distintos conjuntos de usuarios. El beneficio económico de múltiples clientes se basa en la reducción de los costos de actualización y gestión de aplicaciones.

Bases de Datos

- RDBMS
- Hbase
- MongoDB
- NoSQL

Arquitectura en un entorno de “Cloud”

“El Cloud es una plataforma, y construir el sistema para ejecutar en el Cloud, especialmente utilizando el IaaS no es muy diferente que construir una plataforma distribuida. Esto significa, que el arquitecto necesita prestar atención en la adaptabilidad, la usabilidad, la interoperabilidad, y el testeo, como haría en cualquier otra plataforma.

“Los atributos de calidad que tienen alguna diferencias, son la seguridad, la performance y la disponibilidad”.

Seguridad

1. La seguridad, como siempre, es un tema tanto técnico como no técnico.
2. Los aspectos no técnicos de la seguridad, como qué confianza se coloca en el proveedor de la nube, son los que hay que tener en cuenta.
3. Las aplicaciones en la nube se acceden utilizando Internet a través de protocolos estándar.
4. Las cuestiones de seguridad y privacidad que se derivan de la utilización de internet son sustanciales.

Arquitectura en un entorno de “Cloud”

Performance

1. La capacidad computacional instantánea de cualquier máquina virtual, variará dependiendo de lo que esté ejecutando en esas máquinas.
2. Toda solicitud deberá controlar en sí, para determinar qué recursos está recibiendo en comparación con lo que va a necesitar.
3. Una virtud de la nube es que proporciona un host elástico. Elasticidad significa que los recursos adicionales pueden ser adquiridos según sea necesario.
4. Una máquina virtual adicional, por ejemplo, proporcionará la capacidad computacional adicional.
5. Algunos proveedores de la nube asignarán automáticamente recursos adicionales, según sea necesario, mientras que otros proveedores solicitan recursos adicionales como responsabilidad del cliente.

Arquitectura en un entorno de “Cloud”

Disponibilidad

1. Se supone que siempre el Cloud está disponible. Pero todo puede fallar.
2. Una máquina virtual, por ejemplo, está alojada en un equipo físico que puede fallar.
3. La red virtual es menos probable que falle, pero también es falible.
4. Corresponde al arquitecto del sistema planificar para evitar que las cosas fracasen.

Bibliografía

Software Architecture in Practice

Bass, Clements y Kazman

Third Edition:

- Capítulos: 1,2,3,4,5,
- 6,7,8,9,10,
- 11,12,13,14,15,
- 16,21,22,23,26