Fundamentos de Arquitectura de Software

> EL PROCESO DE DESARROLLO DE SOFTWARE

INTRODUCCIÓN AL CURSO DE FUNDAMENTOS DE ARQUITECTURA DE SOFTWARE

En la arquitectura del software se habla de:

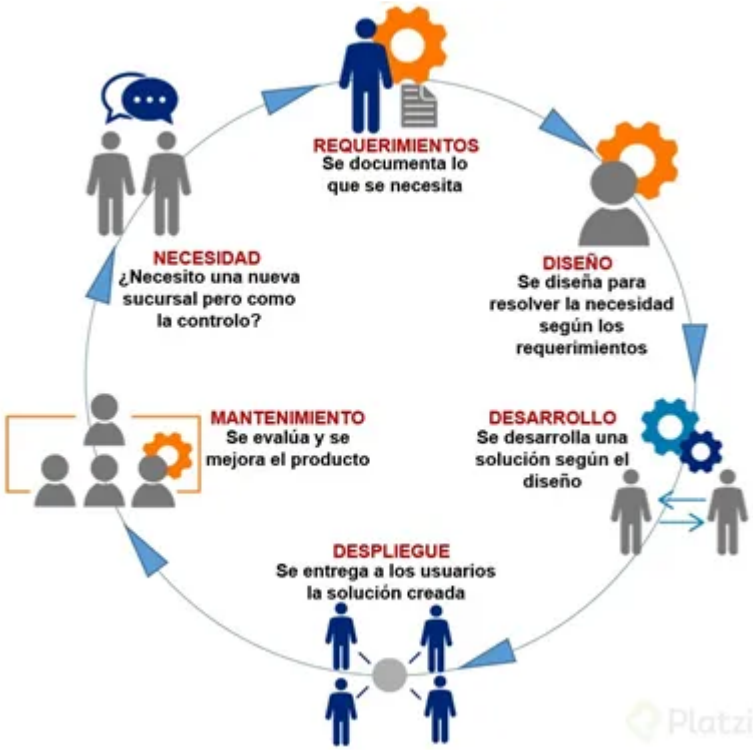
- Estructuras
- Modelos con diagramas
- Se suelen hablar de la comunicación entre diferentes sistemas o incluso entre diferentes módulos del sistema

1. En este curso va a atravesar todo el camino para atender que es el proceso de desarrollo y como la arquitectura está involucrada en cada uno de los pasos del proceso de desarrollo del software.
2. Entenderemos cuál es el rol del arquitecto y como el arquitecto puede ayudar al éxito o fracaso de un sistema.
3. Este curso va a hacer evidentes decisiones que a veces son implícitas y nos va a ayudar a ser consiente de cuando estamos tomando una decisión arquitectónica en un sistema y cómo hacer para tomar la mejor decisión posible en ese momento.

ETAPAS DEL PROCESO DE DESARROLLO DE SOFTWARE

El proceso de desarrollo tradicional tiene etapas muy marcadas, que tienen entradas, procesos y salidas que funcionan como entradas de la siguiente etapa.

- **Análisis de requerimientos**: Todo nace de un disparador que nos crea la necesidad de crear un artefacto o un sistema. Necesitamos entender cuál es el problema que queremos resolver. Hay requerimientos de negocio, requerimientos funcionales, requerimientos no funcionales.
- **Diseño de la solución**: Análisis profundo de los problemas para trabajar en conjunto y plantear posibles soluciones. El resultado de esto debe ser el detalle de la solución, a través de requerimientos, modelado, etc.
- **Desarrollo y evolución**: Implementación de la solución, para garantizar que lo que se está construyendo es lo que se espera. Al finalizar esta etapa tendremos un artefacto de software.
- **Despliegue**: Aquí vamos a necesitar de infraestructura y de roles de operación para poder poner el artefacto a disponibilidad.
- **Mantenimiento y evolución**: Desarrollo + despliegue + mantenimiento, en esta etapa estamos atentos a posibles mejoras que se hacen al sistema. En esta etapa el software se mantiene hasta que el software ya deja de ser necesario.



ETAPAS TRADICIONALES DEL PROCESO DE DESARROLLO DE SOFTWARE

1. Análisis de requerimientos.
2. Diseño de la solución.
3. Desarrollo y evaluación.
4. List
5. Mantenimiento y evolución.

ANÁLISIS DE REQUERIMIENTOS

El análisis de requerimientos nace a partir de un disparador sea un negocio, una idea que resuelva algún problema. En esta etapa tenemos que entender de qué se trata el problema, es donde descubrimos cuáles son los requerimientos, sea de negocio, funcionales, no funcionales, restricciones y/o atributos de calidad, etc. En el análisis de requerimientos debemos tener total compresión del problema.

DISEÑO DE LA SOLUCIÓN

En esta etapa debemos hacer un análisis más profundo del problema (solución detallada) para plantear posibles soluciones sea a través de los requerimientos o documentación, es donde definimos qué tecnologías se van a usar, el modelado, etc. Esta parte es muy importante porque es donde también definimos restricciones del desarrollo, así cuando empezamos a desarrollar la solución ya tenemos bien delimitado lo que vamos a usar para el mismo.

DESARROLLO Y EVALUACIÓN

El desarrollo y evaluación es la etapa donde implementamos el diseño de la solución, realizamos testes automatizados para garantizar no solo eficiencia y eficacia del desarrollo, como también garantizar que lo que estamos desarrollando es lo que se espera, al final de esta parte del proceso tendríamos que tener el artefacto de software.

DESPLIEGUE

En esta etapa ponemos a disponibilidad el artefacto de software, es donde necesitamos de infraestructura y roles de operación para la disponibilidad, por ejemplo, si estamos desarrollando un software web, vamos de un servidor para alojar el código.

MANTENIMIENTO Y EVOLUCIÓN

El mantenimiento y evolución es compuesto por desarrollo y despliegue, es donde debemos quedar atentos a posibles errores y nuevos requerimientos de funcionalidades hasta que el software cumpla su ciclo de utilidad, es decir, cuando el software no sea más necesario y cuando esto pasa decimos que la solución esta deprecada.

DIFICULTADES EN EL DESARROLLO DE SOFTWARE

Existen dos tipos de problemas: los problemas esenciales y los problemas accidentales.

ESENCIALES

Se dividen en 4 tipos de problemas

- **Complejidad**: cuando un dominio de un problema es complejo en sí mismo. En el caso de adiciones y todas las acciones que conlleven al sistema a ser más complejo. Manejo del problema de complejidad: No desarrollar: Comprar – OSS
- **Conformidad**: en qué contexto se usa el software y cómo debe adecuarse al mismo. Se incluyen todo lo que le compete. Ej.: Ambiente, conectividad, impuestos, etc. *Manejo del problema:* **Prototipado rápido,** feedback y ciclos rápidos para solucionar pequeñas.
- **Tolerancia al cambio:** posibilidad del cambio en el mismo y que sea responsivo a diferentes contextos. *Manejo del problema:* **Desarrollo Evolutivo**, desarrollos pequeños. Paso a paso, pero de manera firme e ir haciendo crecer el software.
- **Invisibilidad:** problemas de tangibilidad nula. *Manejo del problema:* **Grandes diseñadores,** Arquitectos que saben abstraer el problema y que realiza soluciones elegantes, de manera simple, con la mejor calidad posible en los componentes que lo

ACCIDENTALES

Está relacionado con la plataforma que vamos a implementar, tecnología, lenguajes, frameworks, integraciones, entre otros, que tienen 3 entornos:

- **Lenguajes de alto nivel**
- **Multiprocesamiento**
- **Entornos de programación**

“Considero a la especificación, diseño y comprobación del **concepto** la parte difícil de hacer *software*. (...) si esto es cierto, hacer software siempre será difícil. no existe la bala de plata.” – Del libro “*No Silver Bullet”*

ROLES

Es importante que diferenciemos el ROL del puesto de trabajo, hay roles que pueden ser desarrollados por la misma persona.

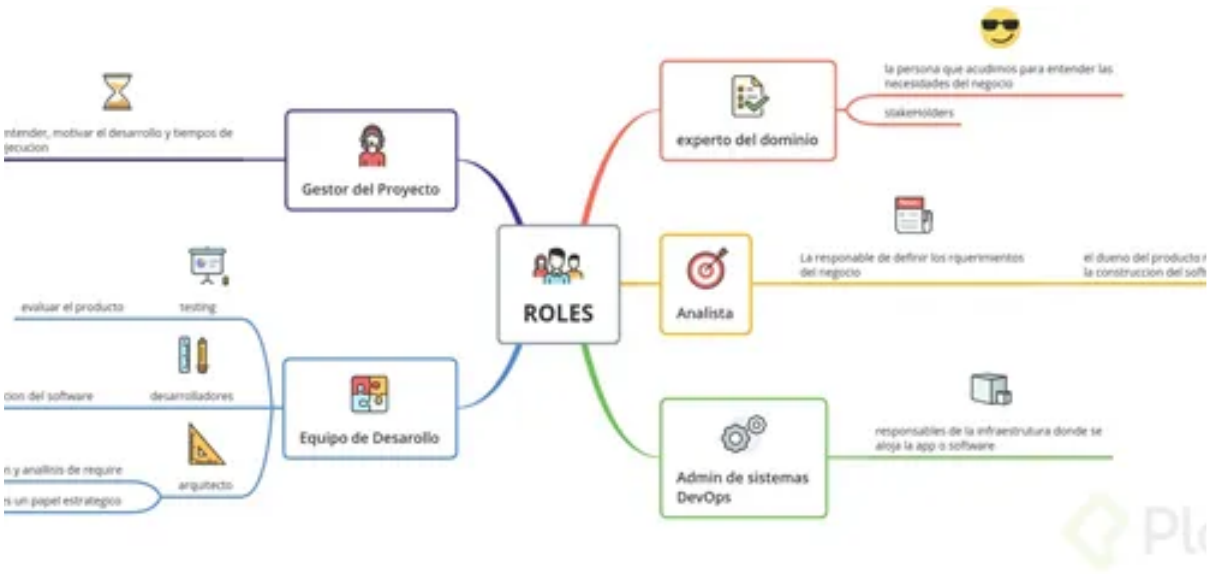
- **Experto del dominio:** en una metodología tradicional, es la persona a la que acudimos para entender las necesidades del negocio. En metodologías Ágiles à
- **Analista:** funcional/de negocio, la persona responsable de definir los requerimientos que van a llevar a software a un buen puerto. En el caso de Ágiles el dueño del producto es quien arma las historias y que nos acompaña en el proceso de construcción del software.

- **Administrador de sistemas / DevOps:** es el rol de operaciones y desarrollo, son las personas responsables de la infraestructura que alojara nuestra aplicación.
- **Equipo de desarrollo:** QA / Testing se encargan de la evaluación de nuestro software, comprobar lo que se está haciendo es lo que se espera que se haga. Desarrolladores involucrados en la construcción del software. Arquitecto, diseña la solución y análisis de los requerimientos, es un papel más estratégico. La arquitectura emerja del trabajo de un equipo bien gestionado.
- **Gestor del proyecto / facilitador:** llevan al equipo a través del proceso iterativo e incremental, entender lo que pasa con el equipo y motivar el avance en el desarrollo del producto.

PRINCIPIOS DEL MANIFIESTO ÁGIL

- Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempos más corto posible.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan entorno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- El método más eficiente y efecto de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto organizados.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar sus comportamientos en consecuencia.

Metodología tradicional	Metodología ágil
rto del dominio: persona para resolver as necesidades del requerimiento.	Partes interesadas (stakeholders) partes interesadas de nuestro software
sta: el encargado en indagar en que es que hay que resolver, responsable en lefinir los problemas y consigo los requerimientos.	Cliente/ Dueño del producto: Tiene el rol poder armar historias de usuario, para pod construyendo el software, dependiendo de necesidades.
nistrador de sistemas: Administradores rgados de la operación del sistema, se rgaban de los sistemas operativos, los rvidores, feedback a los equipos de desarrollo.	DevOps/SRE: Es operaciones y desarrollo u Viene a cumplir el rol de la persona respon de entender la infraestructura, requerimien nuestro proyecto. Además el Rol de SR ingeniero de la confianza del sitio, que est poco más cerca del administrador de siste conecta la infraestructura con el desarrol
uipo de desarrollo : compuestos por diferentes roles, equipo de tester, desarrolladores, arquitectos.	No hay puestos de trabajo, el equipo se enc de todo, y que la arquitectura sumerja del ec
stor del Proyecto: se encarga de las as, gestión de ciclo de vida del sistema.	Facilitador: se encarga de entregas, llevar equipo a través de un ciclo de desarroll



> INTRODUCCIÓN A LA ARQUITECTURA DE SOFTWARE

¿QUÉ ES ARQUITECTURA DE SOFTWARE?

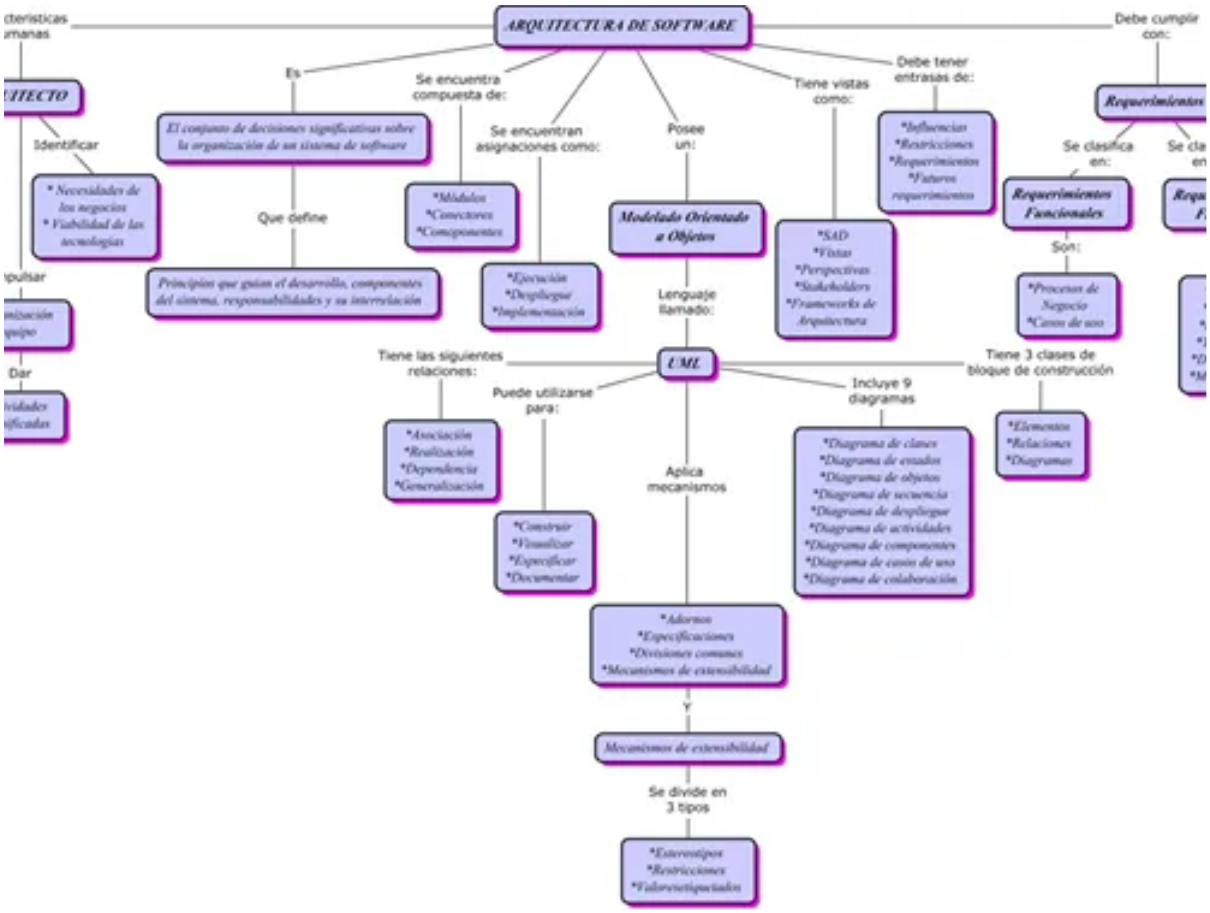
La arquitectura, más que un modelo es algo estructural. El concepto de arquitectura de software se refiere a la estructuración del sistema que, idealmente, se crea en etapas tempranos del desarrollo.

Esta estructuración **representa un diseño de alto nivel del sistema** que tiene dos propósitos

- Satisfacer los atributos de calidad (desempeño, seguridad, modificabilidad)

- Servir como guía en el desarrollo.

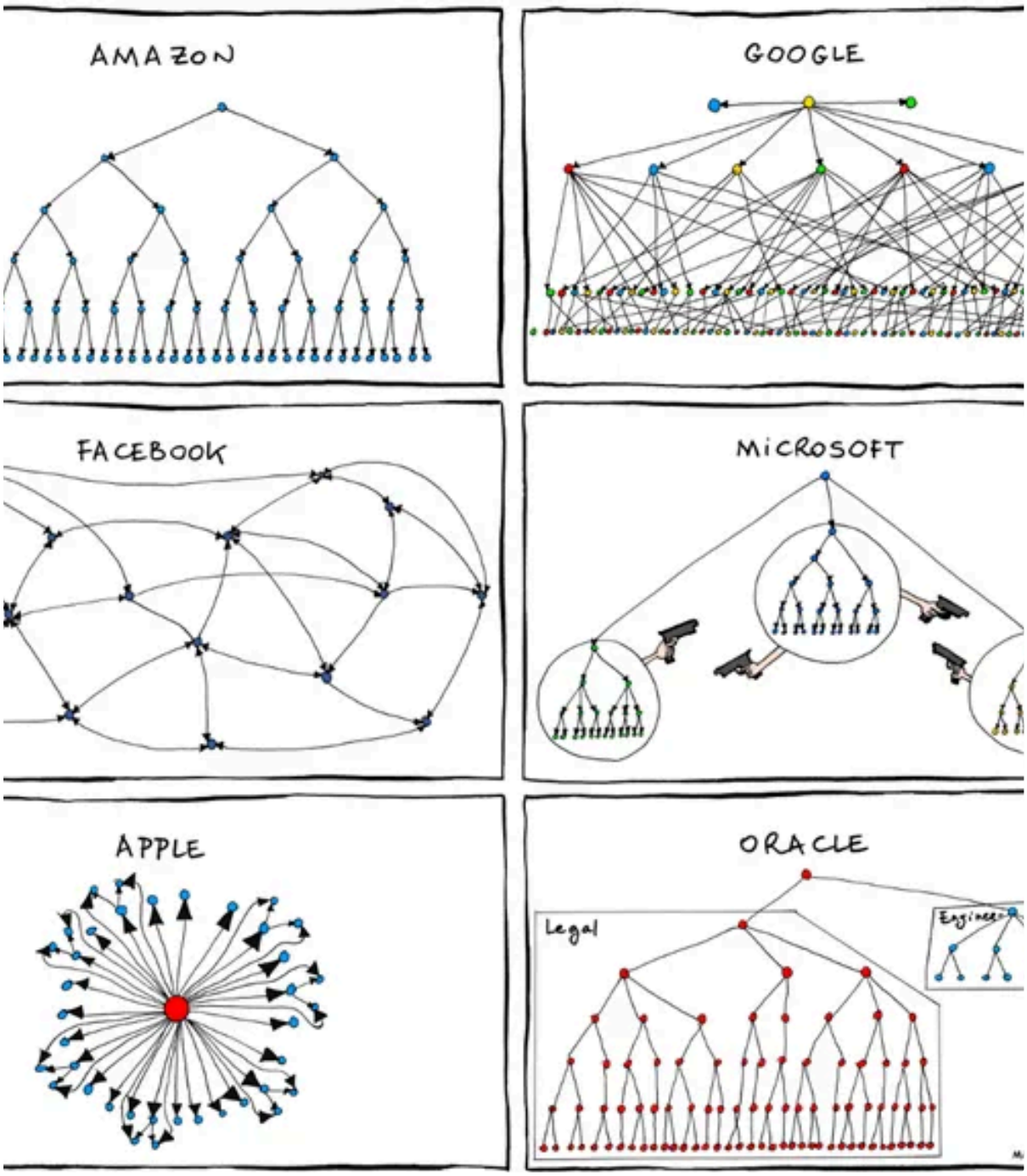
El no crear este diseño desde etapas tempranas del desarrollo puede limitar severamente el que el producto final satisfaga las necesidades de los clientes. Además, el costo de las correcciones relacionadas con problemas en la arquitectura es muy elevado. Es así que la arquitectura de software juega un papel fundamental dentro del desarrollo.



LA IMPORTANCIA DE LA COMUNICACIÓN - LEY DE CONWAY

“Cualquier organización que diseñe un sistema producirá un diseño que copia la estructura de comunicación de dicha organización”.

Conway no dijo esta afirmación como una broma, sino con una justificación real por detrás. Este hecho es causado porque dos componentes software (por ejemplo, A y B), no pueden conectarse correctamente a menos que quien diseña y quien implemente el módulo A se comunique con quien diseñe e implemente el módulo B. Así, este problema en la forma de comunicación de la empresa se refleja en el software, ya que el desarrollo es una actividad intelectual que depende mucho de las propias personas que lo desarrollan.



OBJETIVOS DEL ARQUITECTO

El arquitecto conecta los stakeholder con el sistema a construir. Cada uno de los roles que tienen los SH afectan de diferente forma el sistema.

Cada uno de los stakeholder tiene que ser conectado por el Arquitecto con sus requerimientos.

Stakeholder à Arquitecto à Requerimientos = Implementaciones en el sistema.

Los Requerimientos de cada stakeholder afectan de forma única el sistema.

- **Cliente:** entrega a tiempo y dentro del presupuesto.
- **Manager:** permite equipos independientes y comunicación clara.
- **Dev:** que sea fácil de implementar y de mantener.
- **Usuario:** es confiable y estará disponible cuando lo necesite.
- **QA:** es fácil de comprobar.

La unión de todos estos requerimientos (funcionales o no funcionales) van a llevar al arquitecto a tomar decisiones que impacten sobre el sistema.

OBJETIVOS DEL ARQUITECTO

El arquitecto tiene varias partes interesadas “stakeholders” el cual tiene que conectar esto requerimientos de cada stakeholders con la implementación del sistema.

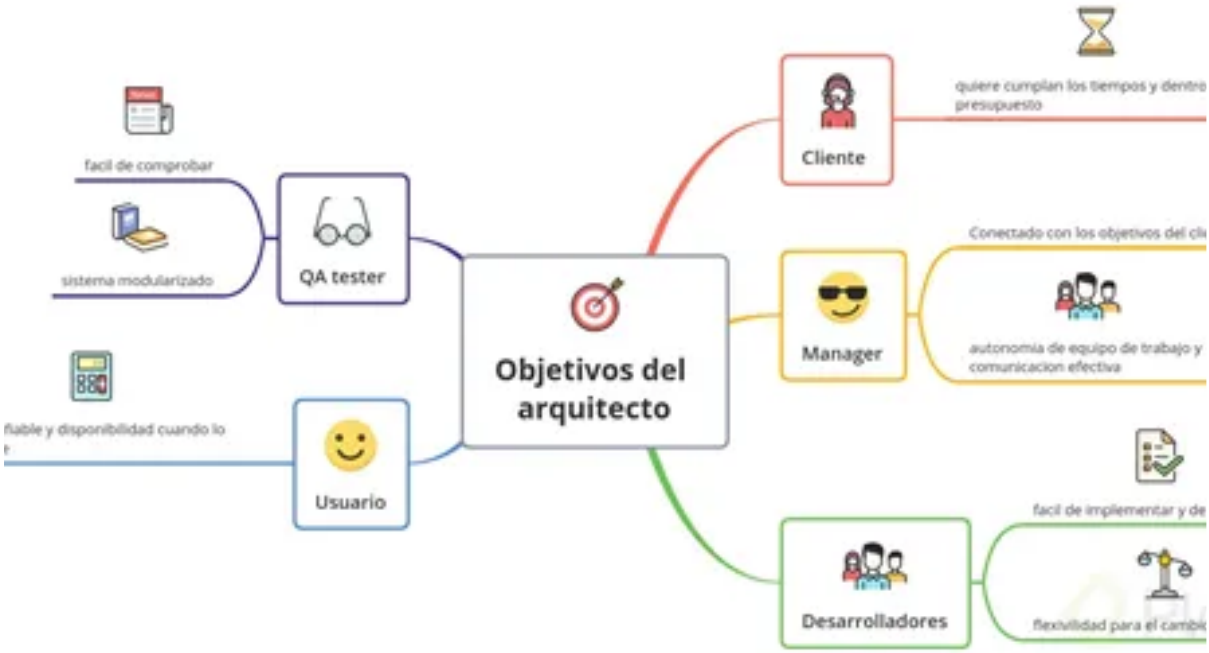
Stakeholders involucrados con diferentes requerimientos:

- **Cliente:** entrega a tiempo y que no rebase el presupuesto.
- **Manager:** comunicación clara entre los equipos que participan en el desarrollo del sistema
- **Dev:** que el desarrollo llevado a cabo sea fácil de implementar y mantener
- **Usuario:** disponibilidad del producto.
- **QA:** fácil de aprobar.

El arquitecto de software debe gestionar los siguientes puntos para cada Stakeholder:

- Encontrar los riesgos más altos que afecten en el desarrollo del sistema (**Cliente**).
- Modularización y flexibilidad del sistema que se está desarrollando (**Manager**).
- Modularidad, mantenibilidad y capacidad de cambio de software (**Dev**)
- Decidir estrategias para la disponibilidad del sistema (**Usuario**)
- Que el sistema pueda ser modularizado y cada una de estas partes pueda ser probado de forma fácil (**QA**).

La unión de estos requerimientos (funcionales / no funcionales) va a llevar al arquitecto a tomar decisiones que impactan directamente en el desarrollo del software.



ARQUITECTURA Y METODOLOGÍAS

En el contexto de método tradicional... Las decisiones de arquitectura es en la **ETAPA DE DISEÑO** En el contexto de método ágil... Las decisiones de arquitectura son **EN CADA ITERACIÓN**

Una metodología de desarrollo de software es un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software.

https://is1blog.wordpress.com/2016/04/07/metodologia-agil-o-metodologia-tradicinal/.

La arquitectura nace en metodologías tradicionales en donde su objetivo era principalmente encontrar los problemas y diseñar una solución a gran escala que ataque a esos problemas esenciales, como también a los de alto riesgo del desarrollo a realizar.

Las metodologías Ágiles plantean que la arquitectura emerge de un equipo auto gestionable y por ende ven al diseño de una solución como algo evolutivo y que se va dando de sprint a sprint.

DIFERENCIAS

- **Tradicional:** en la etapa de diseño es donde se tiene la definición del problema, requerimientos, restricciones y riesgos todos estos son los agentes que van a ayudar al arquitecto a tomar decisiones, el arquitecto contara con herramientas de diseño para poder tomar esto como entrada, para luego tener un modelo de la arquitectura y la documentación para implementar ese modelo, la etapa de diseño no implementa software, sino que le da a la etapa de desarrollo las herramientas para implementar lo que se analizó, al modelo tradicional lo que le falta es el feedback ya que el arquitecto no tiene nociones de lo que el sistema ya hace y como el usuario interactúa con ese sistema, hasta que el arquitecto no hace falta toda esa solución y la solución no se implementa y se despliega no se tiene feedback de como esas decisiones son tomadas o si son buenas decisiones.
- **Agile:** en la metodología ágil todo se trata de momentos en donde podemos evaluar o reevaluar nuestras decisiones, esto se realiza en el planeamiento del spring que es donde planeamos los momentos arquitectónicos importantes. Una vez planeado el spring y se define lo que se tiene que definir arquitectónicamente se ejecuta en base a las prioridades que se tienen y luego se le lleva al usuario haciendo el despliegue y gracias a eso se obtiene feedback, cabe destacar que una vez que se obtiene feedback se pueden reevaluar las decisiones de la arquitectura.

> ANÁLISIS DE REQUERIMIENTOS

ENTENDER EL PROBLEMA

La parte más importante de entender el problema es: **separar la compresión del problema de la propuesta de solución,** si no se entiende la diferencia entre estos dos puntos se tiende a solucionar problemas inexistentes y a hacer sobre ingeniería.

PROBLEMA

Detalla ¿qué es lo que se va a resolver? (y que no se va a resolver) sin entrar en detalles del “como”à (análisis del problema).

El espacio del problema nos ayuda a entender que es lo que vamos a resolver y exactamente como imaginamos como esto va a agregar un valor a nuestros usuarios sin entrar en detalle de cómo lo va a resolver el sistema.

- **Idea:** ¿qué queremos **resolver**?
- **Criterios de éxito:** ¿Cómo identificamos si **estamos resolviendo el problema**?

- **Historias de usuario:** supuestos de historias de lo que **va a ganar** el usuario al utilizar la solución usando las **características del problema** a resolver.

SOLUCIÓN

Brinda el detalle del **¿" cómo" se va a resolver?**, reflejando los detalles del problema detectado y evitando resolver problemas que no se quiere o necesita resolver. à (**detalles técnicos**).

Se refleja en el espacio del problema y trata de resolverlo teniendo en cuenta todos los detalles técnicos necesarios.

Consta de:

- **Diseño:** todo lo referente a la planificación del software, desde diseño UI, UX hasta diseño de sistemas.
- **Desarrollo:** escribir el código, configuraciones y contrataciones de servicios.
- **Evaluación:** medir la eficiencia y eficacia del software frente al problema.
- **Criterios de aceptación:** medir el **impacto** del software, no importa lo bueno que sea la solución si los usuarios no lo usan o no le ven uso.
- **Despliegue (deploy):** lanzar el software en ambientes productivos (mercado) y empezar a mejorar las características con un feedback loop (crear, medir, aprender).

REQUERIMIENTOS

Una vez que entendemos el espacio del problema y el espacio de la solución, vamos a entrar a analizar los requerimientos de nuestro sistema.

Requerimientos de producto: Los podemos dividir en 3.

- Capa de requerimientos de negocio, son reglas del negocio que alimentan los requerimientos del negocio.
- Capa de usuario, tienen que ver en cómo el usuario se desenvuelve usando el sistema, qué atributos del sistema se deben poner por encima de otros.
- Capa Funcional, se ven alimentados por requerimientos del sistema, ¿qué cosas tienen que pasar operativamente? Esta capa se ve afectada por las restricciones que pueden afectar operativamente a lo funcional.

Requerimientos de proyecto: Tienen que ver más con el rol de gestor de proyectos, se usan para dar prioridad a los requerimientos del producto.

Estos dos mundos de requerimientos hablan de las prioridades del equipo de trabajo del proyecto.

Requerimientos de producto:

- **Requerimientos funcionales:** Tienen que ver con las historias de usuarios, que hablan sobre específicamente lo que hace el sistema, por ejemplo, que usuario ingrese al sistema.

- **Requerimientos no funcionales:** son aquellos que agregan cualidades al sistema, por ejemplo, que el ingreso de ese usuario sea de manera segura.

RIESGOS

Es necesario identificar los riesgos para poder priorizarlos y atacarlos en orden y asegurar que las soluciones arquitectónicas que propongamos resuelvan los problemas más importantes.

IDENTIFICACIÓN DE LOS RIESGOS

- **Toma de requerimientos (Requerimientos funcionales):** se calificará su riesgo de acuerdo a su dificultad o complejidad.
- **Atributos de calidad (Requerimientos NO funcionales):** se calificará su riesgo de acuerdo a la incertidumbre que genere, cuanta más incertidumbre hay, más alto es el riesgo.
- **Conocimiento del dominio:** riesgo prototípico, son aquellos que podemos atacar de forma estándar.

Pregunta de examen: los riesgos hay que, identificados para poder priorizarlos, recuerda que no es necesario mitigarlos todos, debemos siempre tener en cuenta y dar prioridad a aquellos riesgos que ponen en peligro la solución que se está construyendo.

RESTRICCIONES

Las restricciones en el contexto de un proceso de desarrollo de software se refieren a las restricciones que limitan las opciones de diseño o implementación disponibles al desarrollar.

Los stakeholders, nos pueden poner limitaciones relacionadas con su contexto de negocio, ejemplo:

- **Las limitaciones legales:** la implementación de un producto podría tener restricciones en algún país, y esto sería una limitante a considerar para el desarrollo del producto.

- **Limitaciones técnicas:** relacionadas con integraciones con otros sistemas.
- **El ciclo de vida del producto:** agregará limitaciones al producto, por ejemplo, a medida que avanza el proceso de implementación el modelo de datos va a ser más difícil de modificar.

Notas: el arquitecto debe balancear entre los requerimientos y las restricciones.

> ESTILOS DE ARQUITECTURA

ARQUITECTURA, PANORAMA Y DEFINICIÓN

Un estilo de arquitectura es una colección de decisiones de diseño, aplicables en un contexto determinado, que restringen las decisiones arquitectónicas especificar en ese contexto y obtienen beneficios en cada sistema resultante.

¿QUÉ ES LO QUE ESTÁ PASANDO ARQUITECTÓNICAMENTE EN EL SOFTWARE?

Hay muchas librerías, muchos frameworks y conocimiento arquitectónico implícito en las comunidades. Por ejemplo, si hablamos de palabras como MVC o FLUX (con React) estamos hablando de arquitectura, sin embargo, está implícito dentro del uso de una tecnología específica, y de repente si hablamos de FLUX estando en Java o C# no tiene ningún sentido, ya que no es una arquitectura que se suele encontrar en esa tecnología, sin embargo, arquitectónicamente tiene el mismo valor y podría ser implementado en otra tecnología. Así también, hay decisiones arquitectónicas tales como si empezar un proyecto como un monolito o iniciarlo con una estructura de micro servicios, que se dan por sentado que cualquier cosa seria de mucho más valor iniciarlo como un micro servicio ¿Por qué? Puede ser un porque es la tendencia, porque es lo que se da más fácil para el equipo de desarrollo o porque las herramientas más modernas de hoy están orientadas a micro servicios, sin embargo, falta un análisis más profundo de que es lo que define ese estilo o patrón de arquitectura y cuáles son los payloads o sacrificios que estamos pagando por usarlos y cuáles son los beneficios que esperamos que traigan.

Ningún patrón tiene solo beneficios, cuando hablábamos de que no hay balas de plata, recordemos que ninguno de estos patrones ni estilos nos va a solucionar todos los softwares, siempre hay beneficios y consecuencias de las decisiones de diseño que tomamos.

¿QUÉ ES UN ESTILO DE ARQUITECTURA DE SOFTWARE?

Al hablar de un estilo de arquitectura hablamos de algo genérico. Por ejemplo, podríamos entrar en detalles sobre diferentes páginas de internet: Facebook, Twitter, WordPress, Wikipedia, etc. Todas esas páginas de internet implementan diferentes arquitecturas. Sin embargo, todas esas páginas son una página web, por lo tanto, tienen una arquitectura cliente-servidor donde hay un navegador web que a través de un sistema de DNS y TCP/IP logra conseguir un documento en formato HTML que se lo muestra a través de un navegador al cliente. Esa estructura genérica define el estilo de una arquitectura, en donde, el estilo no nos va a hablar en detalles que problemas está resolviendo del dominio del problema, sino de que problema está resolviendo arquitectónicamente a nivel de los conectores entre diferentes aplicaciones. Como dijimos recién podrían ser por ejemplo un navegador web y servidor, o podría ser una red de peer-to-peer, dos sistemas que están intentando intercomunicarse, o también una aplicación móvil que trata de comunicarse a una IP a través también de TCP/IP y HTTPS. Todo esto define algo genérico que, si nos permite reutilizarlo a través de diferentes softwares, nos va a ayudar a poder reutilizar este conocimiento y aprender de soluciones anterior que tuvieron éxito implementando esas comunicaciones o esos componentes con esos conectores. Si tuviéramos que bajarlo a una definición podemos decir que un estilo de arquitectura es una colección de decisiones arquitectónicas o decisiones de diseño que dado un contexto nos permite ya restringir las decisiones arquitectónicas, es decir, nos da un set de decisiones ya tomadas y nos restringe el resto de las decisiones arquitectónicas para un beneficio ya estimado, podemos usar estas decisiones ya tomadas en el pasado y que tuvieron éxito y aplicarlas en nuestro sistema que comparte un sistema general y esperar tener un éxito parecido al que tuvo quien lo implemento anteriormente.

ESTILOS: LLAMADO Y RETORNO

Cada uno de los componentes hacen invocaciones a los componentes externos y estos retornan información.

Cada componente hace un llamado y espera una respuesta

- **Programa y subrutinas -->** Instrucciones secuenciales que el programa ejecutaba una por una. Luego se hacían instrucciones de salto, de aquí surgieron las funciones que son bloques de código que podemos invocar en cualquier momento. Es el estilo más básico donde se tiene una rutina y se manda a llamar otra subrutina en donde la subrutina puede retornar o no un resultado, pero la rutina principal continua hasta que acabe la subrutina.

- **Orientado a objetos -->** La abstracción es mayor en comparación con el paradigma anterior, se usa para aplicaciones que ya sabemos que vamos a usar durante mucho tiempo. La abstracción ya no es la subrutina, ahora tenemos objetos que se hacen llamados entre sí y esperan respuestas. Una versión con esteroides del estilo anterior. Se utiliza para aplicaciones que vamos a mantener por mucho tiempo. Tratamos de juntar el estado de la aplicación creando objetos los cuales tienen una interfaz publica (interfaz en esto caso se refiere a una definición de funciones o estructura que esta clase puede implementar) donde la llamada no es solo una subrutina, sino objetos que interactúan entre sí.
- **Arquitectura multinivel:** son diferentes componentes que se van a comunicar en un orden en específico donde un componente principal crea el llamado a un componente inferior en algún momento, un ejemplo de esto son las aplicaciones cliente – servidor.

ESTILOS: FLUJO DE DATOS

En este caso no estamos tan preocupados por cual es la secuencia de ejecución sino como los datos fluyen de un punto a otro.

FLUJO DE DATOS

- **Lote secuencial:** lo importante es ejecutar una pieza de código y que el final de esa pieza ya procesada pase a una siguiente etapa.
- **Tubos y filtros:** se tiene un stream o un flujo de datos continuo en donde cada aplicación recibe continuamente esos datos los procesa y los hace como salida a otra aplicación o al final de la ejecución.

Nota: en el estilo de flujo de datos lo que se tiene son diferentes aplicaciones que van a estar conectadas en general en tiempo real por lo tanto ya no se necesita interacción con el usuario para decidir cuándo empieza un proceso o cuando termina otro.

CUANDO USAMOS EL ESTILO DE ARQUITECTURA DE FLUJO DE DATOS

- Cuando tenemos un proceso que tiene que tener una salida clara pero que puede ser separado en partes en donde tenemos parte a parte lo que necesitamos hacer.
- Cuando necesitamos un stream de entrada parte a parte ir procesándolo y tener una salida al final del túnel.

ESTILOS: CENTRADAS EN DATOS

En el estilo de arquitectura centrados en datos se puede observar es que la qaplicacion tieen multiples componenetes, pero alguno de ellos se va aconcentrar en almacenar los datos, ponerlos disponibles y que hacer para que los datos sean correctos.

- **Pizarron:** son diferentes componentes que interactúan con un componente central y cada componente tiene la responsabilidad de procesar, calcular o recibire un dato y escribirlo al componente central ya sabe que tiene todos los datos necesarios puede tener una salida.
- **Centrado en datos:** consiste en compartir informacion de una base de datos y que varios componentes puedan acceder a la misma, es decir, distintos componentes comparten una misma base de datos.
- **Experto:** en este caso el sistema que centraliza los datos, tiene la capacidad de entender los datos y consultas que realiza el cliente, generando salidas inteligentes. (inteligencia artificial).

Nota: en los estilos de arquitectura basados en datos siempre vamos a tener esta concentracion de cómo hacer para consultar o guardar algo, en cada uno de los estilos vistos se puede ver como el dato tiene diferentes formas o diferentes importancias, en algunos casos se centraliza la informacion, en otros se tienen procesos claros donde se sabe cuántos datos faltan y cuantos datos se necesitan mientras que en otros casos también los datos puede que no sean conocidos.

ESTILOS: COMPONENTES INDEPENDIENTES

Este tipo de estilo de arquitecdtura es para desarrollar aplicaciones independientente y no tener acoplamiento fuerte entre cada uno de nuestros componentes.

- **Invocación implicita:** suele ser basada en eventos, habla de como hacerle para que nuestras aplicaciones se manden mensajes entre si. Cuando se tiene eventos, naturalmente se tienen componentes y un bus de eventos donde los componentes van a publicar eventos y luego el bus de eventos los va a notificar. Aquí se encuentra **Publicar y Suscribir** que trata de un componente que publica y otro componente que suscribe, todo a través del bus de eventos. También existe el **Enterprise Service Bus** el cual tiene componentes registrados los cuales se pueden comunicar con el bus, los componentes no se conocen entre si, pero están programados para cumplir con su objetivo.
- **Invocación explicita:** tiene que ver con el desarrollo de componentes que sí se conocen entre si, pero se han desarrollado independiente. Aquí se encuentra **Orientado al Servicio** en donde todos los componentes se registran al “Registro central” y después indican donde comunicarse.

COMPARANDO ESTILOS: ¿CÓMO ELIJO?

ESTILOS MONOLÍTICOS

Estilos donde se despliegan un artefacto de software.

1. **Eficiencia:** al tener un solo artefacto se puede ser optimizado de manera más personalizada. En estilos distribuidos, es un problema debido a los canales de comunicación, red, internet que comunican los componentes.
2. **Curva de aprendizaje:** el monolítico contiene toda la información allí. Un monolítico bien diseñado permite tener todas las piezas en el mismo lugar, por lo que se facilita la lectura y entendimiento. En el caso distribuido hay que entender cada componente. Nota: un componente interno en un distribuido puede ser visto como un monolítico. Es la base de los microservicios.
3. **Capacidad de prueba:** son más fáciles de probar una funcionalidad de principio a fin. En distribuidos necesito tener todos los componentes disponibles, incluyendo los BUS de eventos.
4. **Capacidad de modificación:** un cambio que se despliega todo junto garantiza menos estados intermedios. Las versionas nunca coexisten. En distribuidos diferentes componentes tienen diferentes versiones, por lo que requiere de compatibilidad entre versiones. Una modifcación en un distribuido es más dificil hacer llegar.

ESTILOS DISTRIBUIDOS

Componentes que luego de ser desplegados se conectan de alguna forma.

1. **Modularidad:** es separa componentes que prestan servicios.
2. **Disponibilidad:** es mayor que en monolítica, podemos tener multiples copias de un componente, que esté disponible significa que sea más barato, tener una copia entera de un monolitico es muhco más caro que copiar el componente distribuido que necesitamos que escale. Microservicios aprovecha recursos.
3. **Uso de recursos:** es más facil de gestionar los recursos del sistema.
4. **Adaptabilidad:** al ser distribuido se peude detectar mucho más facil que componente neceista ser adaptado del sistema y es más fácil de realizar esa actualización. En monolíticos puede ser mucho más complicado, como lanzar una app en un sistema operativo diferente.

¿COMO ELIJO QUÉ NECESITO?

Tener en cuenta los requisitos, los objetivos de negocio / arquitectura de software, atributos de calidad / estrategias de arquitectura, escenarios / decisiones arquitectonicas. Con el fin de analizar que sacrificios, riesgos y no riesgos cuento y como impacdta en mi proyecto.

> DESARROLLO DEL PROYECTO

DESARROLLO DEL PROYECTO: PLATZISERVICIOS FASE STARTUP

PLATZISERVICIOS

SITUACION / PROBLEMA

La bañera de nuestra casa está dañada debido a que se rompió nuestra cañería, es necesario los servicios de un plomero que me permita arreglar dicho problema y sea de nuestra confianza.

Entonces se comienza con los requerimientos del sistema:

- **Criterio de éxito:**
    1. Conectar rápidamente a un cliente con un profesional de confianza.
    2. Garantizar el aumento del volumen de trabajo al profesional.
- **Idea:** definición de una forma ideal de como se satisface una necesidad. **Ejemplo**: tener una forma mucho más sencilla de solicitar un servicio de plomería que llegue a mi casa con un plomero que se conozca.
- **Historias de usuarios:** definir las experiencias que los usuarios han tenido respecto a la solución de su necesidad. Ejemplo:
    1. **Experiencia de un cliente x:** quiero contactar a un profesional en el momento para reparar un problema en mi hogar.
    2. **Experiencia de un cliente y:** quiero conocer la experiencia del profesional para decidir a quién contacto.
    3. **Experiencia de un profesional x:** quiero cobrar mi trabajo realizado para seguir prestando el servicio.
    4. **Experiencia de un profesional y:** necesito saber más repertorio de personas para ampliar mi currículo de trabajo y flujo del mismo.

Requerimientos más técnicos:

1. **Etapas de la prestación de servicio:**
    a. Solicitar, aceptar y finalizar una prestación de servicio de forma segura.
2. **Comunicación:** la forma en como el cliente solicita el servicio a su hogar.
3. **Evaluación:** como se evalúa a los profesionales y clientes para futuros tiempos.

Riesgos: son referentes a historias de los usuarios. Ejemplo:

- El cliente utiliza un servicio y no completa el pago en un tiempo determinado.
- La persona que solicita el servicio no puede confirmar quien es la persona.

Restricciones:

- **Recursos disponibles para el desarrollo:** programadores, equipos de cómputo, energía, comida, lugar de trabajo, servicios públicos, etc.

- **Registro de impuestos del profesional:** que el profesional cumpla con el pago de impuestos ante las instituciones.
- **Antecedentes penales:** que el profesional cuente con ser un ciudadano ejemplar dentro de la ley. Teniendo en cuenta todas las restricciones y requerimientos que existe, lo más adecuado es montar una arquitectura cliente-servidor dentro de la web que permite de una manera mucho más sencilla la automatización de procesos.

DESARROLLO DEL PROYECTO: PLATZISERVICIOS FASE PRODUCTO EN CRECIMIENTO

Sinopsis: nuestra startup está teniendo éxito como sistema y así mismo hemos de necesitar de nuevos requerimientos para llegar a la mayor cantidad de clientes posibles.

ANÁLISIS DE REQUERIMIENTOS

Criterios de éxito:

- Brindar con nuestro servicio a empresas clientes estabilidad y control de costos de las prestaciones de servicios que se necesiten.
- Se tiene una visión en el circulo de empresas prestadoras una visión de crecimiento de sus servicios.

Historias de usuario:

- El empresario cliente x quiere llevar control de sus finanzas mediante el reporte de gastos en servicios.
- El empresario y necesita que haya un grupo de profesionales preferidos para nunca perder la disponibilidad de nuestro servicio.
- El empresario prestador x quiere medir el rendimiento de sus profesionales.
- El empresario prestador y quiere posicionarse mejor en el mercado para obtener más clientes.

Requerimientos:

- **Reportes:** gastos por periodo, por el tipo de servicio contratado, ingresos, horas trabajadas por el profesional por periodo y tipo de servicio prestado.
- **Autorización:**
  - **Gestión de usuario:** el tipo de empleado que va a tener la empresa.
  - Roles
  - Permisos asociados a acciones del sistema
- **Posicionamiento y comunicación:** ranking de prestadores por evaluación, lista priorizada de prestadores por tipo de prestación.

Riesgo

- **Las empresas clientes no tienen como extraer información del sistema** debido a que estas manejan su propia información.
- Los juicios hechos por las mismas empresas prestadores de acuerdo con los fraudes.

Restricciones

- Cumplir con estándares de auditoría profesional para que nuestro software sea seguro.
- Garantizar la privacidad de datos de consumo.

Con todo esto se decide que el estilo arquitectónico en la estructura cliente servidor pasa sus transacciones en lote secuencial a los reportes teniendo en cuenta el costo que supone presentar reportes dentro del aplicativo.

DESARROLLO DEL PROYECTO: PLATZISERVICIOS FASE ESCALA