***Notas del Curso.***

**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.

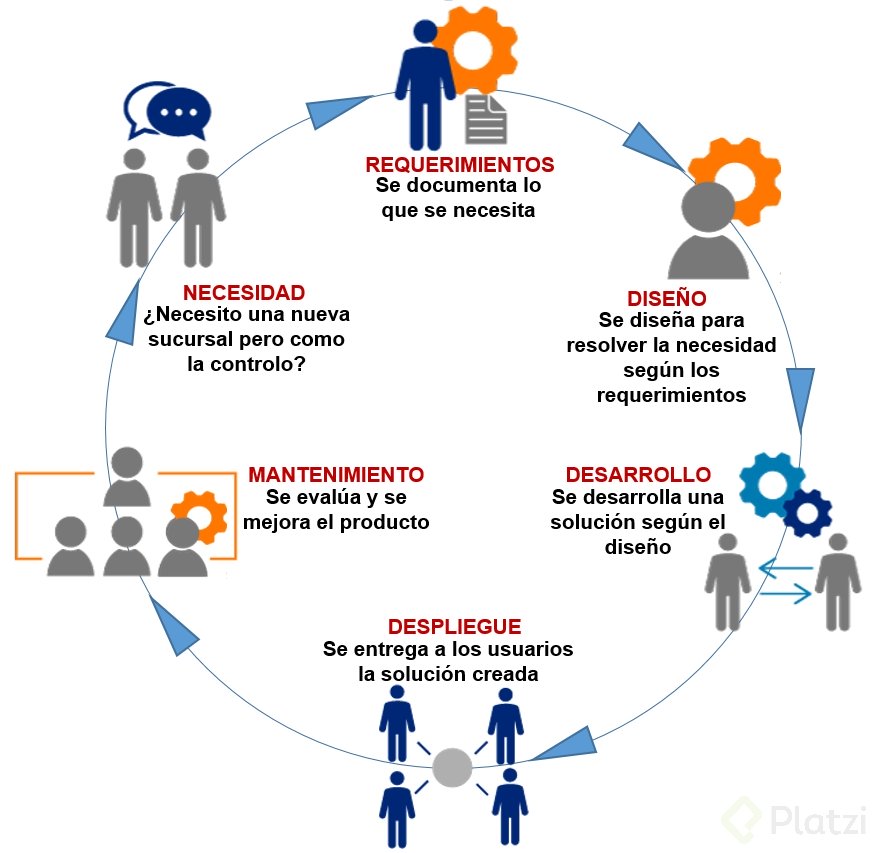
**Análisis de requerimientos:** Todo nace de un disparador que nos crea la necesidad de crear un artefacto o un sistema. Necesitamos entender cuál es el problema que queremos resolver. Hay requerimientos de negocio, requerimientos funcionales, requerimientos no funcionales.

**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 esta 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 posible mejoras que se hacen al sistema. En esta etapa el software se mantiene hasta que el software ya deja de ser necesario.



# 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 soluciones 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 abtraer el problema y que realiza soluciones elegantes, de manera simple, con la mejor calidad posible en los componentes que lo **necesitan**.

**Accidentales**: Está relacionado con la plataforma que vamos a implementar, tecnología, lenguajes, frameworks, integraciones, entre otros, que tienen 3 Entornos:  
• **Lenguajes de alto nivel**  
• **Multi-procesamiento**  
• **Entornos de programación**  
“Concidero 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 \*\*(**Frederick P. Brooks Jr., 1986**)

# 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 --> stakeholders.

**Analista:** funcional/de negocio, la persona responsable de definir los requerimientos que van a llevar al software a u 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 que 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.

# ¿Qué es arquitectura de software?

**Arquitectura de Software es:**

“*La estructura del sistema, compuesta por elementos de* ***software****, sus propiedades visibles y sus relaciones*”

* Software Architecture in Practice

“*El conjunto de decisiones principales de diseño tomadas para el sistema*”

* Software Architecture: Foundations, Theory and Practice

# La importancia de la comunicación - Ley de Conway

**Ley de Conway**

“Cualquier pieza de software refleja la estructura organizacional que la produjo.”

**Relacionar la estructura de un equipo con las estructuras de comunicación resultantes del software.**  
**Monolítico:**  
Un solo equipo, se comunica informalmente sin restricciones.  
**Distribuido:**  
Múltiples equipos. Vías definidas de comunicación, dependencias más evidentes (idealmente, evitadas).

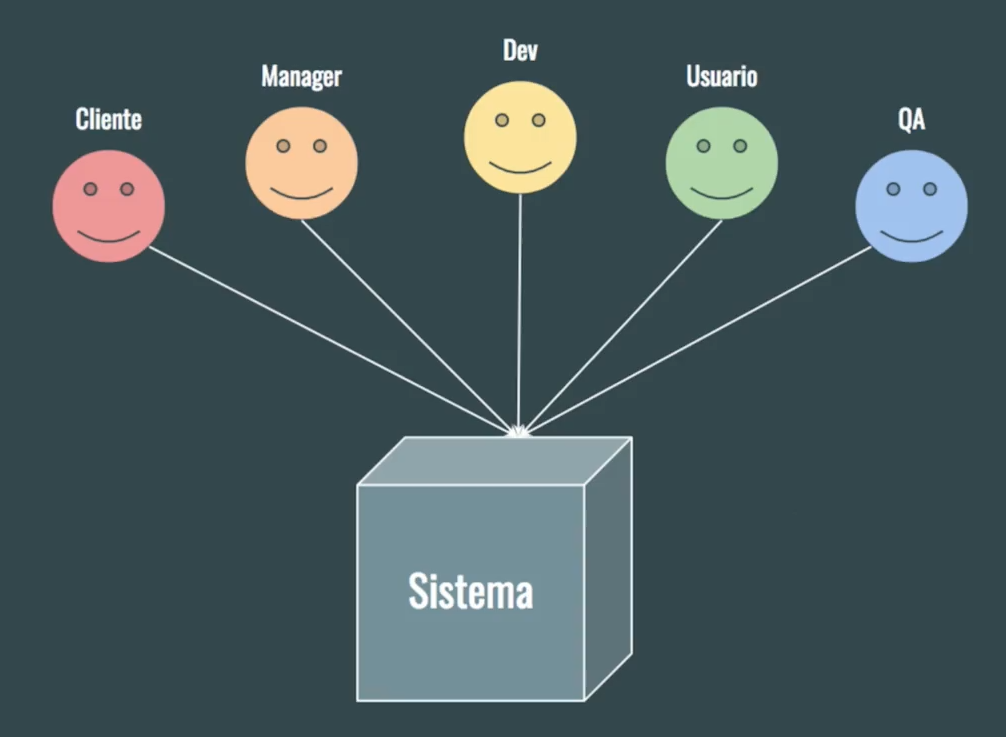
# Objetivos del arquitecto.

Cada uno de los stakeholder tiene que ser conectado por el Arquitecto con sus requerimientos.  
***Stakeholder -> Arquitecto -> Requerimientos = Implementaciónes 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.



# Arquitectura y metodologías

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.



# Entender el problema.

A la hora de tomar requerimientos y procesarlos es muy importante entender el problema a resolver.

La parte más importante es separar la comprensión del problema de la propuesta de solución, ya que si no se separan estos dos puntos se tiende a confundir varios aspectos.

**¿Cómo hacemos para entender bien un problema?**

Hay que separar el espacio del problema del espacio de la solución.

• **Espacio del Problema:** Es la idea que se va hacer y que se quiere resolver.

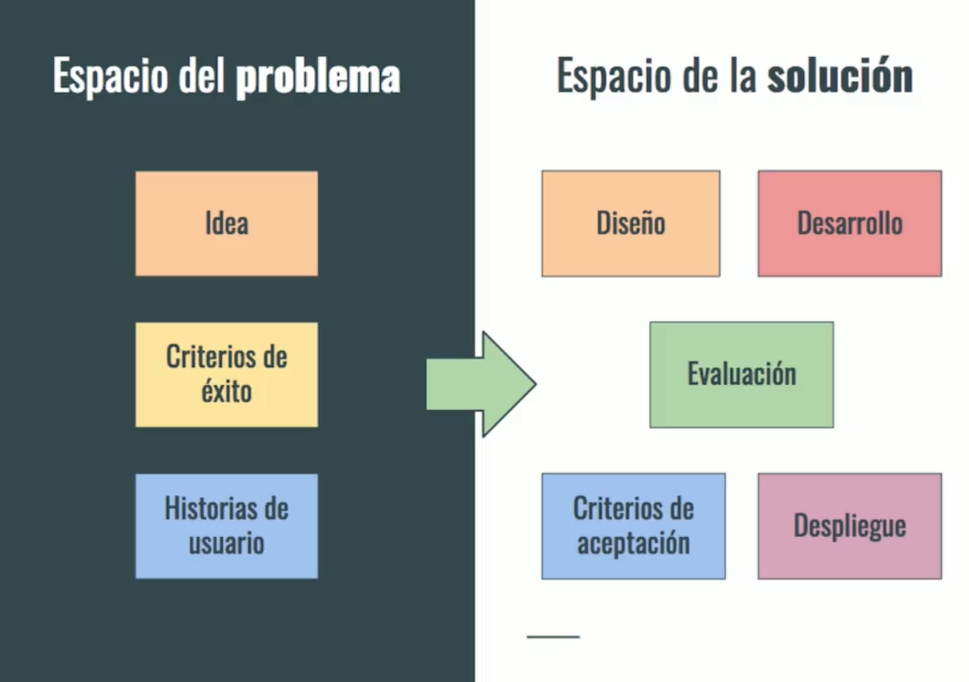
El espacio del problema nos ayuda a entender que es lo que vamos a resolver y exactamente como imaginamos como esto va agregar un valor a nuestros usuarios sin entrar en detalle de cómo lo va a resolver el sistema.

• Idea  
• Criterios de éxito  
• Historias de usuario

• **Espacio de Solución:** Consta de un desarrollo, diseño, evaluación de desarrollo.

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

• Diseño  
• Desarrollo  
• Evaluación  
• Criterios de aceptación  
• Despliegue

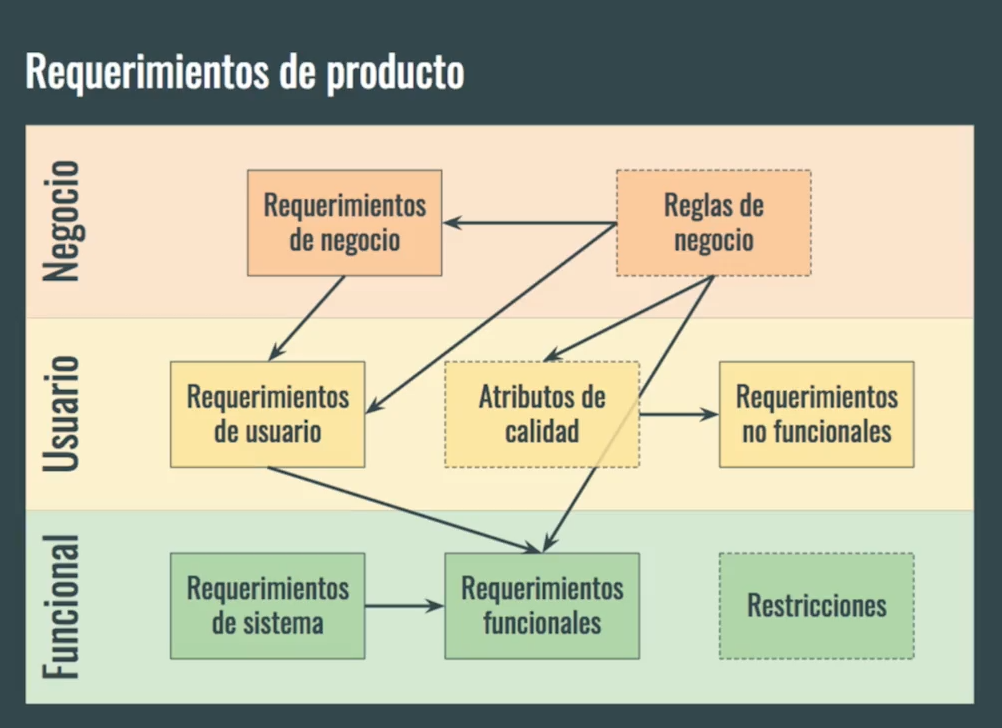


# Los Requerimientos.

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 tres (03):

**• 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.

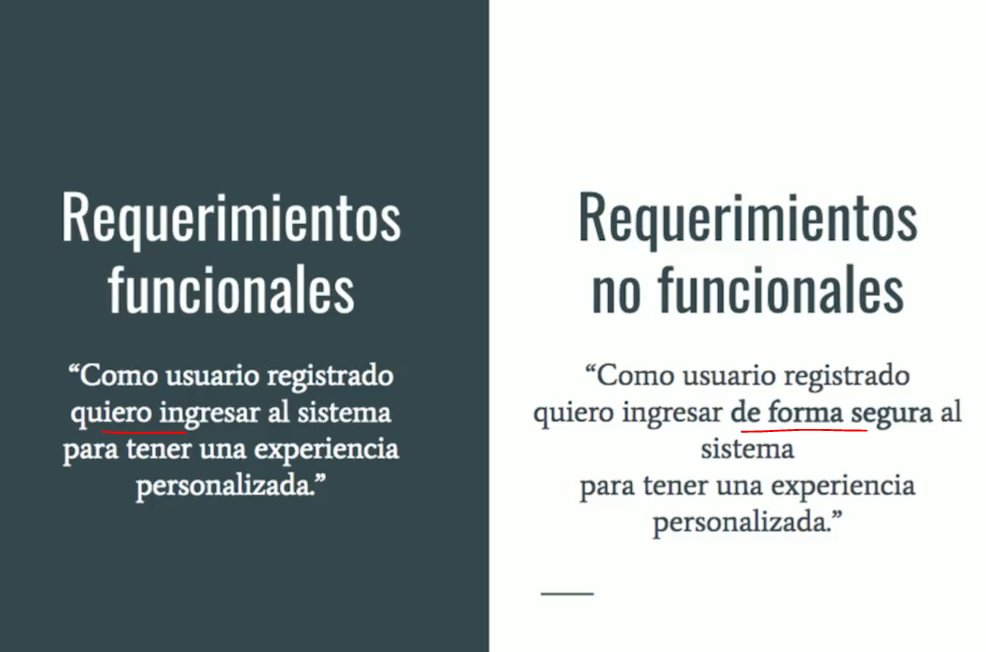


**Requerimientos Significativos para la Arquitectura del Producto:**

**• Requerimientos funcionales:** (Funciones indispensables) 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:** (Atributos de calidad): son aquellos que agregan cualidades al sistema, por ejemplo que el ingreso de ese usuario sea de manera segura.

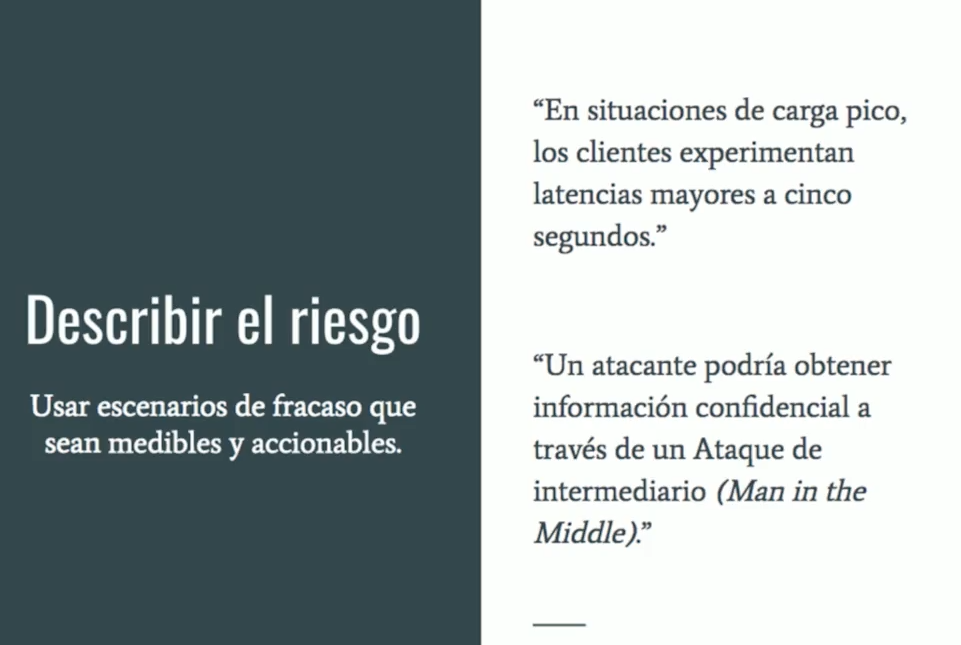
**Requerimientos de proyecto**

• 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.  
• Tiene que ver con requerimientos logísticos, que no tienen que ver con el desarrollo del software.



**Los riesgos.**

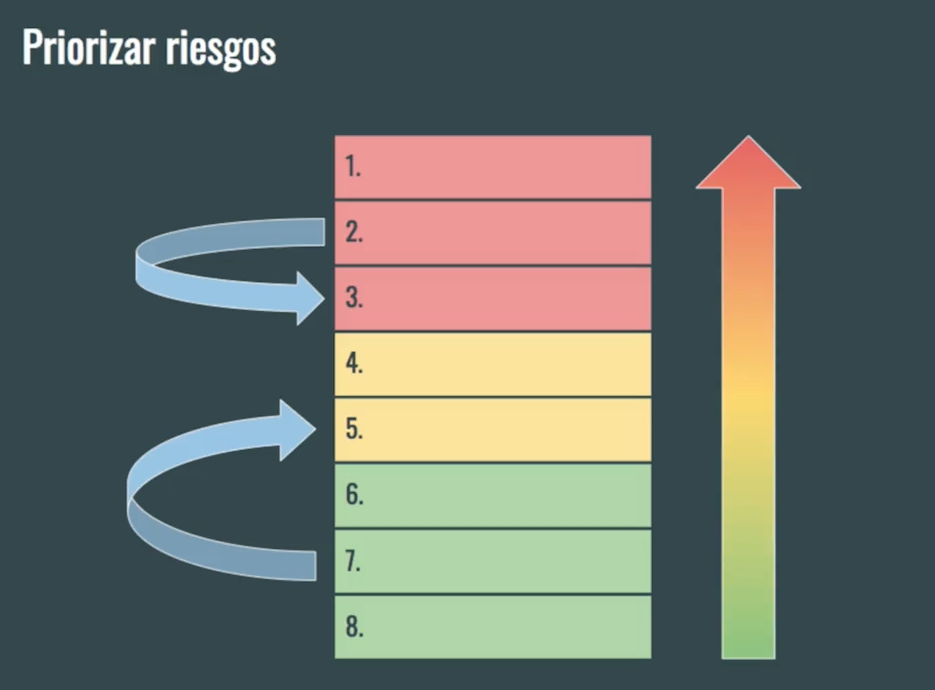
Es necesario identificar los riesgos para poder priorizarlos y atacarlos en orden y asegurar que las soluciones arquitectónicas que propongamos resuelvan los problemas más importantes.



**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, cuanto mas incertidumbre hay, mas alto es el riesgo.  
**• Conocimiento del dominio:**  
Riesgo prototípico, son aquellos que podemos atacar de forma estándar.

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.



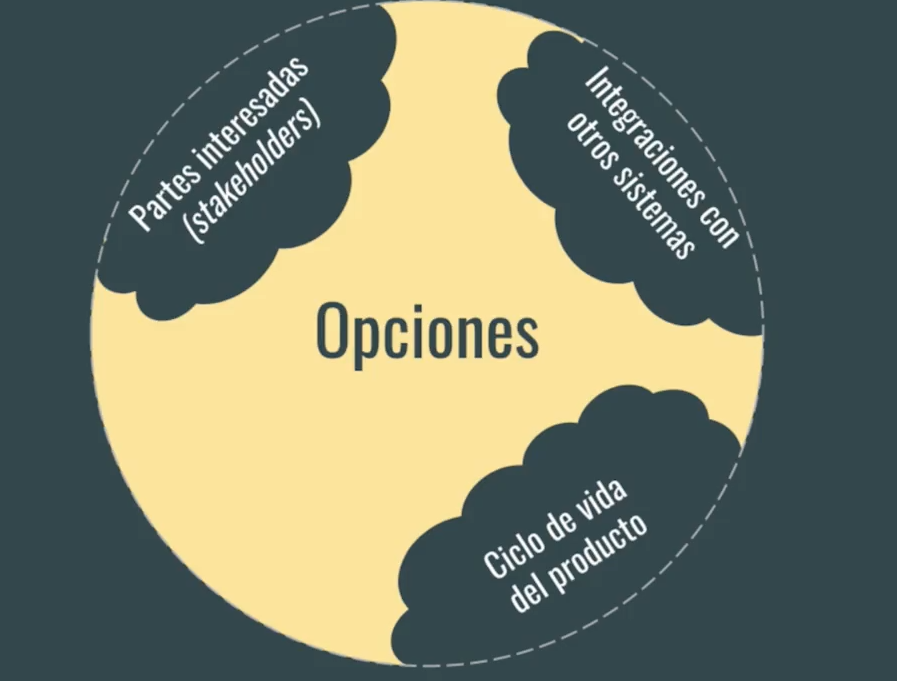
**Las Restricciones.**

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

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

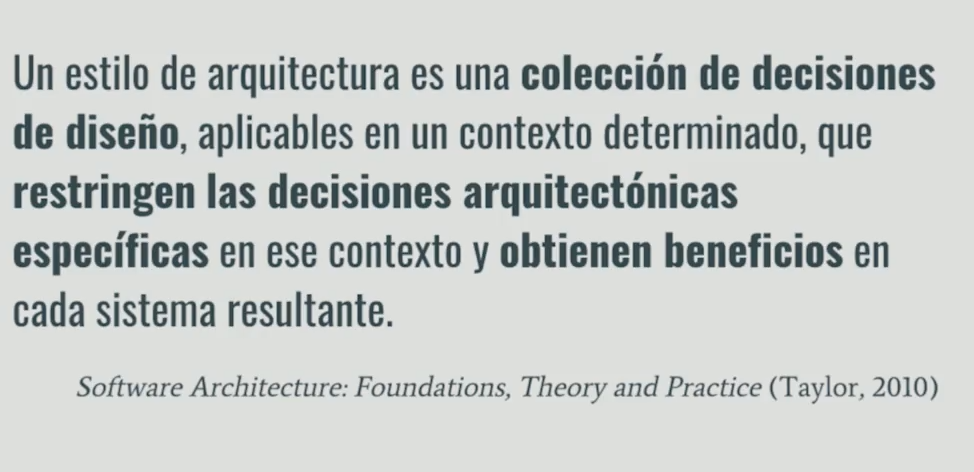
**• Las limitaciones legales**, la implementación de un producto podría tener restricciones en algún país, y esto seria una limitante a considerar para el desarrollo del producto.  
**• Limitaciones técnicas**, relacionadas con integraciones con otros sistemas.  
**• El ciclo de vida del producto**, 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.

**Nota:**  
El arquitecto debe balancear entre los requerimiento y las restricciones.



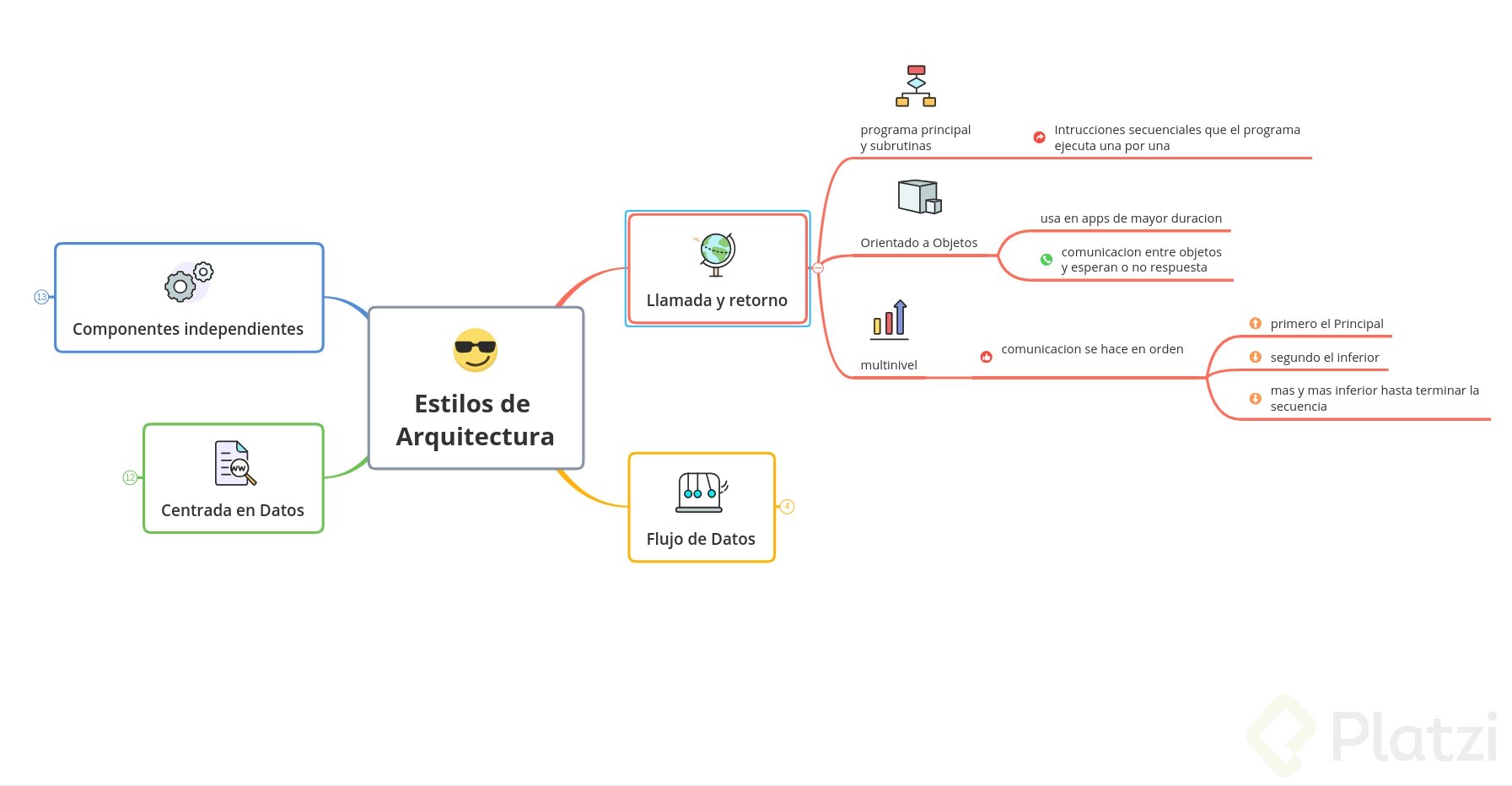
**Estilo de Arquitectura.**

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



**Estilos. Llamado y retorno.**

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



**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 string 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:**

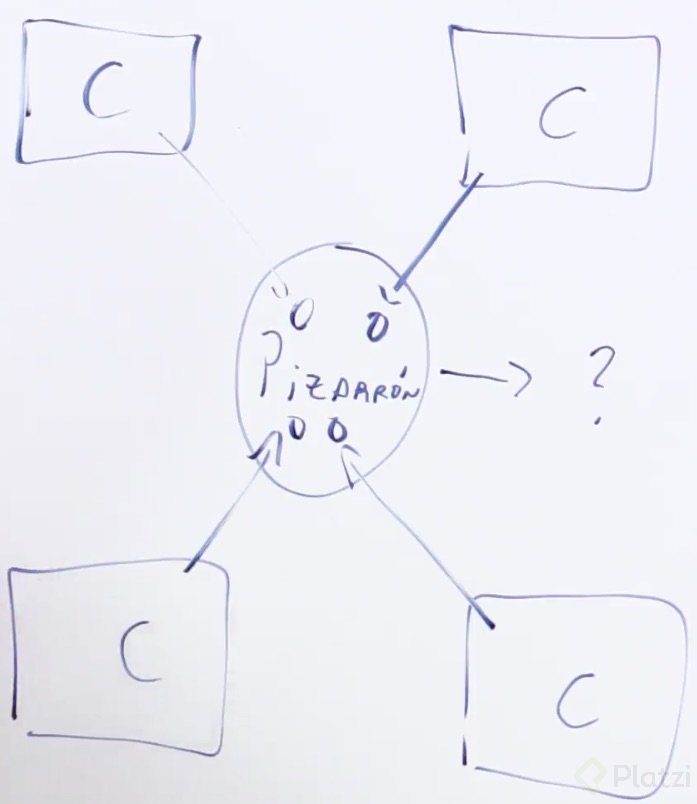
En el estilo de flujo de datos lo que se tiene son diferentes aplicaciones que van a estar conectadas en general en tiempo real por lo tanto ya no se necesita interacción con el usuario para decidir cuándo empieza un proceso o cuando termina otro.

**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 string de entrada parte a parte ir procesándolo y tener una salida al final del túnel.

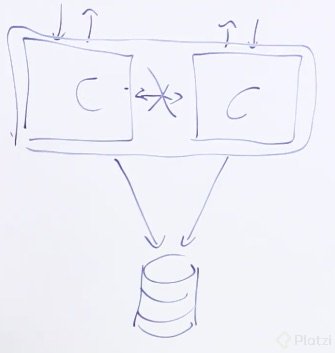
**Estilos. Centradas de datos.**

**Pizarrón:**

* El pizarrón es el núcleo de la arquitectura. Donde componentes externos a el se encargarán de procesar un dato y escribirlo en el pizarrón(Este funciona como centralizador). Cuando el pizarrón ya tiene todos los datos necesarios; el mismo podría generar una salida.
* 

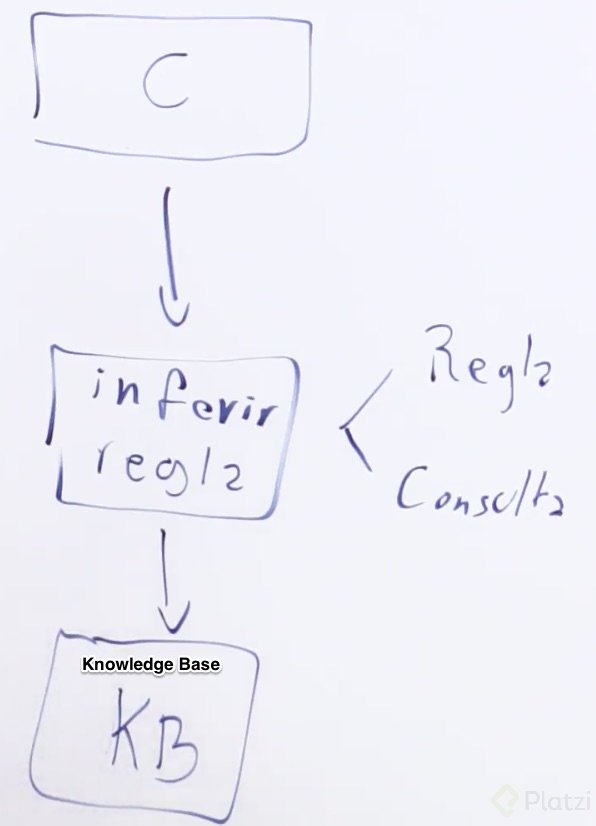
**Centrado en base de datos:**

* Es un estilo común; Se trata de que una cantidad de componentes comparte una misma base de datos. de Ejemplo: aplicaciones que poseen comunicación por Internet.



**Sistema experto Basado en reglas:**

* Este sistema no se ve muy seguido en aplicaciones modernas; un componente A (Tipo Cliente) consulta a uno B, donde este se encargará de tratar de entender si la petición del cliente es una **consulta** o **regla**. Para que el componente B logre resolver la petición se va a comunicar con un tercer componente © este trabajara como KDB: **Knowledge DataBase**.



**Estilos. Componentes independientes.**

Este tipo de estilo de arquitectura es para desarrollar aplicaciones independientemente 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 independientemente. 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:**

* Es más fácil darle prioridad a la eficiencia de las comunicaciones.
* Son más fáciles de probar.
* Curva de aprendizaje son más fáciles, todas las piezas estan en el mismo lugar. (Los microservicios son fáciles de entender).
* La capacidad de modificación es más fácil.
* La modularización es más fácil de romper, por lo que es más fácil no garantizar esa separación a largo plazo.
* En la usabilidad, es mas costoso, porque habría que respaldar toda la aplicación y no pequeños microservicios.
* Puede ser un desafío para el despliegue, porque habría que garantizar que toda la aplicación o sistema se adapta a ese contexto específico.

**Estilos Distribuidos:**

* Es más fácil darle prioridad a la eficiencia de las comunicaciones.
* Para hacer una prueba de principio a fin hay que tener todos los componentes disponibles .
* La curva de aprendizaje es más difícil, porque habría que entender todas las piezas de los componentes.
* Al ser desplegadas independientemente, son versionadas independientemente, y esta variación de serviones hace mas complejo su modificación.
* La modularidad, es más fácil porque los componentes que son desplegados independiente.
* la disponibilidad se pueden tener multiples copias del sistema. por lo que este disponible es mas barato.
* La adaptabilidad es más fácil en el despliegue porque los componente se despliegan independientemente en múltiples contextos.



*Ejemplo como contextualización de todo lo anterior.*

# Desarrollo del proyecto: PlatziServicios Fase Startup

**PlatziServicios**

**Situación/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 volúmen 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 quien contacto.
3. **Experiencia de un profesional x:** quiero cobrar mi trabajo realizado para seguir prestando el servicio.
4. **Experiencia de un profesional y:** necesito tener más repertorio de personas para ampliar mi currículo de trabajo y flujo del mismo.

**Requerimientos más técnicos:**

1. **Etapas de la prestación de servicio:**
   1. 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 evalua 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**

Limites que tiene nuestro proyecto de acuerdo a variables.

**Ejemplo:**

* **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 exito como sistema y así mismo hemos de necesitar de nuevos requerimientos para llegar a la mayor cantidad de clientes posible.

**Análisis de requerimientos:**

**Criterios de exito:**

* 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 cliente y necesita que halla un grupo de profesionales preferidos para nunca perder la disponibilidad de nuestro servicio.
* El empresario prestador x quiere medir el redimiento de sus profesionales.
* El empresario prestador y quiere posicionarse mejor en el mercado para obtener más clientes

**Requerimientos:**

* **Reportes:** Gastos por período, por el tipo de servicio contratado, ingresos, horas trabajadas por el profesional por período y tipo de servicio prestado.
* **Autorización:**
  + **Gestión de usuario:** El tipo de empleado que va a tener la empresa.
  + Roles
  + Permisos asociado a accines del sistema
* **Posicionamiento y comunicación:** Ranking de prestadores por evaluación, lista priorizada de prestadores por tipo de prestación.

**Riesgo**

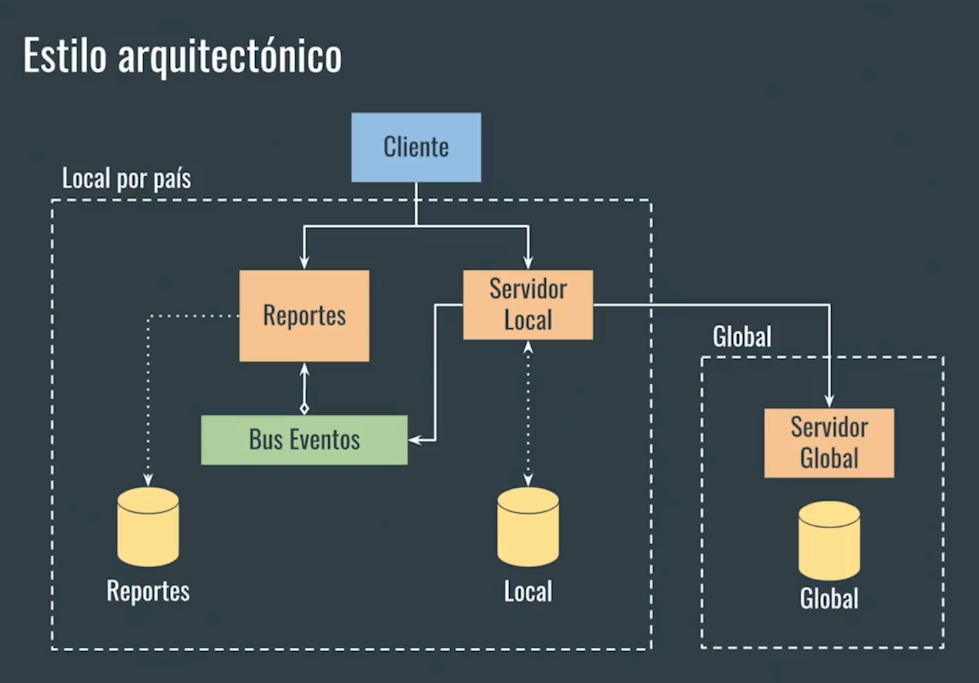
* **Las empresas clientes no tienen como extraer información del sistema** debido a que estas manejan sus propia información.
* Los juicios hechos por las misma empresas prestadoras de acuerdo con los fraudes.

**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 su 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 global



**Sinopsis:** somos una empresa a gran escala y por ende vamos a necesitar de nuevos requerimientos.

**Análisis de requerimientos:**

* **Criterios de exito:**
  + Se desea conectar tanto a empresas locales como globales con los mejores prestadores del servicio.
  + Como visión se busca dar mayor facilidad de crecimiento y globalización de las empresas prestadoras.
* **Historias de usuario:**
  + **El cliente a:** quiere entender el sistema en su propio idioma. **Razón:** quiere garantizar el buen uso del servicio.
  + **El cliente b:** quiere acceder tanto a servicios globales como locales. **Razón:** Quiere estandarizar a los prestadores en diferentes ubicaciones.
  + **El usuario** necesita acceder a los servicios en cualquier momento sin tener problemas de los husos horarios que existan dentro del mismo
  + **La empresa prestadora:** quiere brindar el servicio de fomar global. **Razón:** ampliar su alcance a escala internacional
* **Requerimientos**
  + **Internacionalización:**
    - Traducciones de contenido
    - Registro de prestadores globales y su capacidad de busqueda local o global.
  + **Disponibilidad de datos:**
    - Que sean medidos mediante reportes en tiempo real
* **Riesgos**
  + Dificultad de transmisión de conocimiento y productividad de nuestros equipos de desarrollo.
  + Perdida de datos de manera parcial o total por imprevistos
  + Dificultad de acceso a los mercados por sus distinción de idiomas.
* **Restricciones**
  + Husos horarios
  + La información que se provea a escala internacional  
    En conclusión: Se necesitan servidores globales y locales que conecten la información necesaria para la toma de decisiones.
  + A su vez se los reportes se conectan con bus de eventos para ofrecer en tiempo real información acerca de la empresa.