

# El arquitecto de Software



# El camino ...

El camino para lograr una carrera de arquitecto de software es difícil por muchas razones. Convertirse en un arquitecto de software competente puede ser difícil, pero mantener las habilidades suele ser incluso peor.

➤ Cuerpo de Conocimiento

➤ Confusión y Gurus


Evangelista y líder del equipo de desarrollo

➤ Celos profesionales

La competencia profesional puede crear graves problemas en las organizaciones y en su carrera,

➤ Trampa de la Gestión

➤ Crisis del software



Cambios en requerimientos → expectativas del usuario  
Innovación comercial → updates y upgrades  
Computación distribuida → entornos heterogeneos

## Essential Architect Skills

### Business Skills

Pragmatism Vision  
Business Knowledge Innovation

### Personal Skills

Context Switching  
Transparency Passion

### Relationship Skills

Leadership Politics  
Gracious Behavior Communication Negotiation

### Technical Skills (assumed)

Aquí están los 12 skills esenciales

## Essential Architect Skills

### Visionary Skills

Entrepreneurial  
Execution

Technology Strategic  
Innovation Roadmapping

### Technology Skills

Governance Know-how  
Platform Development Architectural Perspective

### Project Skills

Estimation Management  
Partnership Discovery Conceptualization



# Project Skills

# Asociarse (PartnerShip)

- **Alineamiento**
  - Socios, buscar líderes de opinión, conocer a los influenciadores,
  - Tener asesores de confianza
  - Buscar alineamiento antes de tomar decisiones claves
- **Confianza**
  - Evitar sobrecargarse de trabajo
  - Aprender a decir NO
  - Transparencia
- **Contexto**
  - Tener cuidado con el contexto del negocio
  - Las decisiones técnicas requieren de SOCIOS
  - Las decisiones técnicas **en realidad** son políticas
- **Colaboración**
  - Generar valor para el equipo
  - Ser mentor / Buscar un mentor
- **Relaciones**
  - No solo ser socios en el trabajo
  - Buscar socios externos

# Descubrir (Discovery)

CLAVES:

➤ Entender al cliente

Asociarse con marketing, ventas  
Reunirse con el cliente  
Comprender lo que quiere el cliente

➤ Entender el mercado

Aprender acerca de los clientes  
Conocer donde/en qué gastan su dinero  
¿ Qué hace la competencia ?

➤ Entender el negocio

Comprender lo que el negocio hace y lo que necesita hacer  
Desarrollar un contexto de negocio para tomar decisiones

# Conceptualizar

CLAVES:

➤ Generar Ideas

➤ Formar un concepto

¿ que lenguaje estan hablando ?

¿ qué problema están discutiendo ?

Si llegamos tarde a la generación de ideas,  
tener cuidado con el compromiso

➤ Hacer tangible el concepto: Experimental

MVP

➤ Evolucionar el concepto

Estudiar la historia

Ver múltiples perspectivas

Ver oportunidades colaterales

# Estimar

CLAVES:

- Estimar TIEMPO y COSTO
- Identificar Riesgos
- Considerar mitigación de riesgos
- Identificar tecnologías en juego
- Atraer a los ejecutivos clave

Principios

- Conocer el Problema
- Proporcionar opciones
- Dejar las Decisiones de Diseño Abiertas
- Conocer el cronograma
- Evitar ser negativo
- Buscar oportunidades para decir Sí
- Negociar arduamente AHORA, no después
- No ceder
- Confiar en la intuición
- Tener cuidado con los proyectos que otros han estimado



# Administración

## CLAVES:

- Responsabilidades arquitecturales
- Sensibilización de la deuda técnica y financiamiento de la solución correcta.
- Alinearse con el data center y el soporte
- Manejar las expectativas
- Guiar el proceso de desarrollo
- Estar donde están los problemas / Resolver los problemas
- Saber cuando NO es TU problema
- Decir NO, pero dar opciones
- Limitar el número de proveedores en posiciones de liderazgo
- Manejar riesgos con los ejecutivos
- Revisar estimaciones con los ejecutivos
- Asistir a reuniones sólo si eres un participante activo
- Delegar en quienes confías.
- Contratar a los mejores (no sólo cubrir las vacantes)
- Convertirse en experto en un tema
- Buscar proyectos donde puedas mejorar tus skills

# Technology Skills

# Desarrollo de la Plataforma

## ➤ Manejar el ecosistema

- ✓ Usuarios
- ✓ Administradores
- ✓ Desarrolladores
- ✓ Costos asociados con la plataforma
- ✓ Atributos de calidad de la plataforma

## ➤ Tratar de cumplir los principios

- ✓ Buscar Calidad excepcional
- ✓ Buscar excelencia operacional
- ✓ Configurar antes que “hard-codear”
- ✓ Buscar una arquitectura redundante
- ✓ Buscar escalabilidad lineal
- ✓ Mantener actualizadas las tecnologías

# Perspectiva Arquitectural

Principios —→ Para las preocupaciones de arquitectura y la comunicación del modelo

- De la menor sorpresa
- Del Conocimiento Menor (Ley de Demeter)
- Del menor esfuerzo (ley de Zipf)
- Del costo de Oportunidad
- De responsabilidad simple
- De Parsimonia (Navaja de Occam o KISS)
- El Principio de Último Momento Responsable (Costo de Retardo)
- De retroalimentación

El **Principio de la Mínima Sorpresa** se aplica al diseño de interfaces, diseño de 'software' y la ergonomía.

Establece que cuando dos elementos de una interfaz entran en conflicto o son ambiguos, el comportamiento del programa ha de ser el que genere la mínima sorpresa por parte del usuario.

En particular, un programador debería pensar en el comportamiento que menos sorprenda a quien use el programa en lugar del más natural para quien conozca el comportamiento interno del mismo.

En la práctica, conlleva la elección de acciones por defecto adecuadas.

**Principio de Menos Conocimiento** es una directriz utilizada en el desarrollo de software, particularmente en la programación orientada a objetos. En su forma general, la LoD es un caso específico de **loose coupling**.

Esta directiva fue inventada en la Universidad Northeastern (Boston, Massachusetts) a finales del año 1987, y puede ser sustancialmente resumida de las siguientes maneras:

Cada unidad debe tener un limitado conocimiento sobre otras unidades y solo conocer aquellas unidades estrechamente relacionadas a la unidad actual.

Cada unidad debe hablar solo a sus amigos y no hablar con extraños.

Solo hablar con sus amigos inmediatos.

La noción fundamental es que dado un objeto, este debería asumir tan poco como sea posible sobre la estructura o propiedades de cualquier otro (incluyendo sus subcomponentes).

**Principio del Mínimo Esfuerzo** : Postula que los animales, las personas, las máquinas incluso bien diseñados, naturalmente, elegir el camino de menor resistencia o "esfuerzo".

Dos conclusiones:

- Regla 80/20 → Dada toda la información disponible, la gente usa solo el 20% de fuentes para satisfacer el 80% de sus necesidades de información
- La gente escogerá las fuentes de información fácilmente disponibles de relativamente baja calidad antes que gastar tiempo y esfuerzo en acceder a fuentes mas confiables. → Esto explica el éxito de la Wikipedia.

Desde la perspectiva del diseñador, esto ayuda a priorizar tareas e hitos que son enmarcados por el cronograma y el presupuesto.

### **Principio del Costo de Oportunidad**

El costo de hacer algo por dejar de hacer otra cosa. Tiene bases en economía

### **Principio de Simple Responsabilidad**

Defined by Robert C. Martin in his book Agile Software Development, Principles, Patterns, and Practices and later republished in the C# version of the book Agile Principles, Patterns, and Practices in C#, it is one of the five SOLID agile principles. What it states is very simple, however achieving that simplicity can be very tricky.

**A class should have only one reason to change.**

## **Principio de Parsimonia**

KISS = Keep It Simple Stupid

## **Principio de Último Momento Responsable**

Se define como la estrategia de no tomar una decisión prematura sino retrasar el compromiso y mantener las decisiones importantes e irreversibles abiertas hasta que el costo de no tomar una decisión se hace mayor que el costo de tomar una decisión.

## **Principio de Feedback**

La retroalimentación es un método para mantener los sistemas en pista.

En otras palabras, la retroalimentación es una manera de asegurarse de que un sistema se comporta de la manera deseada. Si tenemos alguna métrica de calidad de servicio en mente, a continuación, la retroalimentación es un método fiable para garantizar que nuestro sistema va a lograr y mantener el valor deseado de esta métrica, incluso en presencia de incertidumbre y cambio.

# Governanza

- Evitar “amarrarse” con un proveedor
- Promover el Open SOurce
- Alentar los planes de *recuperación ante desastres* y *continuidad del negocio*
- Asegurar el cumplimiento de Normas
- Alentar la seguridad
- Alentar el principio del menor privilegio
- Buscar el manejo unificado de Identidad y Acceso
- Buscar portabilidad de los datos
- Buscar integración y automatización



# Know-How

- Experiencia RELEVANTE
- Mantenerse ACTUALIZADO
- Buscar la Excelencia

# Visionary Skills

# Innovación Tecnológica

- Prestar atención a las tendencias (pero ser cauteloso)
- Buscar recursos para explorar
- Tener una “sand-box”
- Vigilancia tecnológica de las tendencias

# Hoja de Ruta estratégica

- Usar post-it
- Comenzar del final (ir hacia atrás)
- Usar el principio KISS

Pero HACERLO DIVERTIDO !!!

# Ejecución Empresarial

- Espiritu Emprendedor
- Calcular riesgos
- Generar entregables

## Principios

- Bird-in-the-Hand
- Affordable Loss
- Lemonade
- Patchwork Quilt
- Pilot-in-the-Plane

<http://www.effectuation.org/sites/default/files/documents/effectuation-3-pager.pdf>