[Apuntes: Arquitectura de Software](https://docs.google.com/document/d/1ayrbE-f2nP2EplBe2zJu7o2F1fIxb6-N8LKSCjvJ9U0/edit?usp=sharing)

Sebastián Rojas Cortez

# 

# Índice

| [**1**](#_7v6ixciryy3z) | [**El desarrollo de software**](#_7v6ixciryy3z) | [**3 - 4**](#_7v6ixciryy3z) |
| --- | --- | --- |
| [1.1](#_w4jddww47iiw) | [*Etapas del proceso*](#_w4jddww47iiw) | [3](#_w4jddww47iiw) |
| [1.2](#_w4jddww47iiw) | [*Dificultades*](#_w4jddww47iiw) | [3](#_w4jddww47iiw) |
| [1.3](#_l5o42wm2seic) | [*Roles*](#_l5o42wm2seic) | [2](#_l5o42wm2seic) |
| [**2**](#_1lfz81s0eynj) | [**Análisis de Requerimientos**](#_1lfz81s0eynj) | [**5 - 6**](#_1lfz81s0eynj) |
| [2.1](#_lhquitw0sr62) | [*Entender el problema*](#_lhquitw0sr62) | [5](#_lhquitw0sr62) |
| [2.2](#_4ctedycv4d61) | [*Requerimientos del producto*](#_4ctedycv4d61) | [5](#_4ctedycv4d61) |
| [2.3](#_x2dxtjv5cbh) | [*Requerimientos del proyecto*](#_x2dxtjv5cbh) | [5](#_x2dxtjv5cbh) |
| [2.4](#_7chkydxbfocp) | [*Requerimientos funcionales*](#_7chkydxbfocp) | [5](#_7chkydxbfocp) |
| [2.5](#_8qago3vufdjq) | [*Requerimientos no funcionales*](#_8qago3vufdjq) | [6](#_8qago3vufdjq) |
| [2.6](#_364gy8i1fjoa) | [*Riesgos*](#_364gy8i1fjoa) | [6](#_364gy8i1fjoa) |
| [2.7](#_4s7cm6fn0qau) | [*Restricciones*](#_4s7cm6fn0qau) | [6](#_4s7cm6fn0qau) |
| [**3**](#_86h8xfhtwq8c) | [**Estilos de arquitectura**](#_86h8xfhtwq8c) | [**7**](#_86h8xfhtwq8c) |
| [3.1](#_c7zwccvb2p6a) | [*Comparación de estilos*](#_c7zwccvb2p6a) | [7](#_c7zwccvb2p6a) |
| [**4**](#_gdsnat6167cg) | [**Atributos de calidad**](#_gdsnat6167cg) | [**8 - 11**](#_gdsnat6167cg) |
| [4.1](#_u8nkz6tqaukf) | [*Idoneidad funcional*](#_u8nkz6tqaukf) | [8](#_u8nkz6tqaukf) |
| [4.2](#_5erp8xiyr7cw) | [*Eficiencia de ejecución*](#_5erp8xiyr7cw) | [8](#_5erp8xiyr7cw) |
| [4.3](#_bbmw5b5jm4pg) | [*Compatibilidad*](#_bbmw5b5jm4pg) | [8](#_bbmw5b5jm4pg) |
| [4.4](#_vb0w6e1brtdx) | [*Usabilidad*](#_vb0w6e1brtdx) | [9](#_vb0w6e1brtdx) |
| [4.5](#_y1m2u975qxuc) | [*Confiabilidad*](#_y1m2u975qxuc) | [9](#_y1m2u975qxuc) |
| [4.6](#_z9jb25y7fdpl) | [*Seguridad*](#_z9jb25y7fdpl) | [9](#_z9jb25y7fdpl) |
| [4.7](#_lee3xq1rqzqz) | [*Mantenibilidad*](#_lee3xq1rqzqz) | [10](#_lee3xq1rqzqz) |
| [4.8](#_hm2mfoai3ehw) | [*Portabilidad*](#_hm2mfoai3ehw) | [10](#_hm2mfoai3ehw) |
| [4.9](#_fw22ut8aj4yd) | [*Tensiones entre atributos*](#_fw22ut8aj4yd) | [10 - 11](#_fw22ut8aj4yd) |
| [**5**](#_o6wufdzba4y0) | [**Patrones de arquitectura**](#_o6wufdzba4y0) | [**12**](#_o6wufdzba4y0) |
| [**6**](#_jrr852psv5we) | [**Diseño de arquitectura: Herramientas**](#_jrr852psv5we) | [**13 - 18**](#_jrr852psv5we) |
| [6.1](#_ugyo9vh6q6x6) | [*Componentes*](#_ugyo9vh6q6x6) | [13](#_ugyo9vh6q6x6) |
| [6.2](#_p3cywcae4m15) | [*Conectores*](#_p3cywcae4m15) | [13 - 14](#_p3cywcae4m15) |
| [6.3](#_9fbdr96v04s) | [*Escenarios y tácticas*](#_9fbdr96v04s) | [14 - 18](#_9fbdr96v04s) |
| [**7**](#_2mxo9em79k8i) | [**Validación del diseño**](#_2mxo9em79k8i) | [**19 - 20**](#_2mxo9em79k8i) |
| [7.1](#_7zd640nog9n3) | [*ATM*](#_7zd640nog9n3) | [19](#_7zd640nog9n3) |
| [7.2](#_w50g9sz18a44) | [*Métricas de evaluación*](#_w50g9sz18a44) | [20](#_w50g9sz18a44) |
| [7.3](#_i87xszvp4mgv) | [*Arquitectura en evolución*](#_i87xszvp4mgv) | [20](#_i87xszvp4mgv) |
| [**8**](#_nsfkzy7jvo0f) | [**Modelado y documentación**](#_nsfkzy7jvo0f) | [**21**](#_nsfkzy7jvo0f) |
| [**9**](#_nrt6vabfqo14) | [**Fuentes**](#_nrt6vabfqo14) | [**22**](#_nrt6vabfqo14) |

# 

# El desarrollo de software

## Etapas del proceso

| Etapa | Entrada | Acción | | Salida |
| --- | --- | --- | --- | --- |
| Planificación | | | | |
| 1. Análisis de requerimientos | Problema a resolver | Descubrimiento. Toma de requerimientos y documentación | | Comprensión del problema |
| 1. Diseño de la solución | Comprensión del problema | Abstracción. Producen modelos de solución, limitando desarrollo | | Solución detallada |
| Implementación | | | | |
| 1. Desarrollo y evaluación | Solución detallada | Codificación, evaluación (tests) e integración. | | Software implement. |
| Despliegue | | | | |
| 1. Despliegue | Software implement. | Se pone el software a disponibilidad del usuario (con servidores, store, etc) | | Software disponible |
| 1. Mantenimiento y evolución | Software disponible | Se monitorean errores y evalúan nuevas funcionalidades | | Sistema deprecado |

## Dificultades

| Esenciales | Especificación, diseño y comprobación de concepto. |
| --- | --- |
| Accidentales | Detalles de la implementación y producción actual.  (Plataforma, tecnología, lenguaje, framework, etc) |

### Problemas esenciales

* Complejidad: Complejidad inherente al problema
* Conformidad: Adecuación al contexto de uso
* Tolerancia al cambio: Variabilidad de la problemática que aborda el sw (p.e. leyes)
* Invisibilidad: Falta de imagen tangible del software

### Problema accidental

* Lenguajes de alto nivel: Adaptabilidad de lenguaje binario a humano (legibilidad)
* Multiprocesamiento: Disponibilidad de ejecución/procesamiento
* Entorno de programación: Librería, compilados, apis, etc.

## Roles

***Nota****: no necesariamente está conectado al puesto de trabajo*

***Nota****: En Ágil, la arquitectura emerge desde un equipo autodisciplinado, por tanto redefine los* ***roles preestablecidos*** *(en negrita) por nuevos roles (en rojo)*

1. **Experto** del dominio (Partes interesadas/Stakeholders)
2. **Analista** (Funcional o de negocio). Es quién indaga qué resolver y definir problemas/requerimientos (dueño del producto/cliente)
3. **Administrador** de sistema (DevOps/SRE: site reliability engineer)
4. Equipo de desarrollo
   1. **QA - Tester**
   2. **Desarrollador**
   3. **Arquitecto**
5. **Gestor de proyecto** (Facilitador de entregas. pe: scrum master)

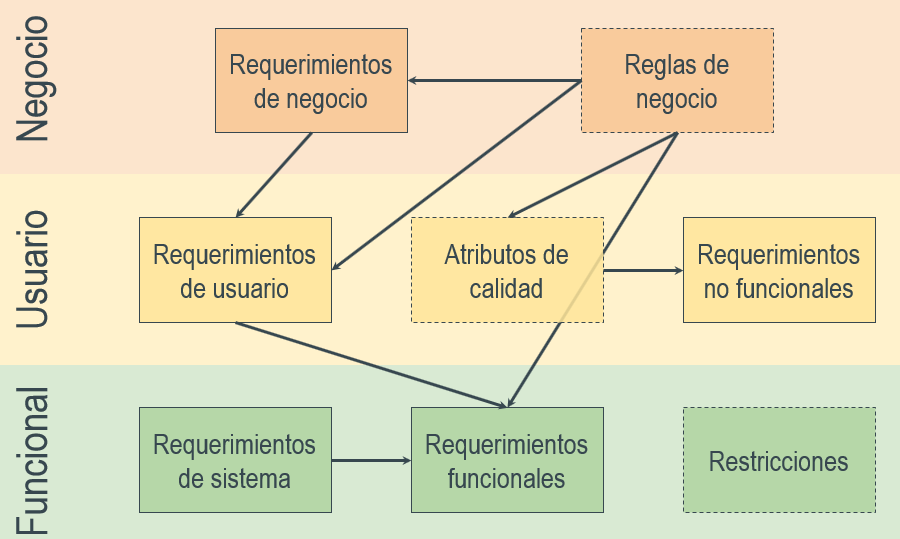
# Análisis de requerimientos

Considera los **requerimientos** del cliente y las **restricciones** de este o el equipo de desarrollo como la etapa inicial en la realización de un proyecto informático. Los requerimientos pueden ser diferenciados entre **funcionales y no funcionales.**

## Entender el problema.

Considera las ideas de solución, criterios de éxito e historias de usuario

## Requerimientos de producto



## Requerimientos de proyecto

1. Recursos
2. Capacitación
3. Certificaciones
4. Documentación de usuario
5. Infraestructura
6. Licencias
7. Plan de despliegue
8. Plan de transacción
9. Acuerdo de servicio

## Requerimientos funcionales

Se relacionan a la implementación de funcionalidades

## Requerimientos no funcionales

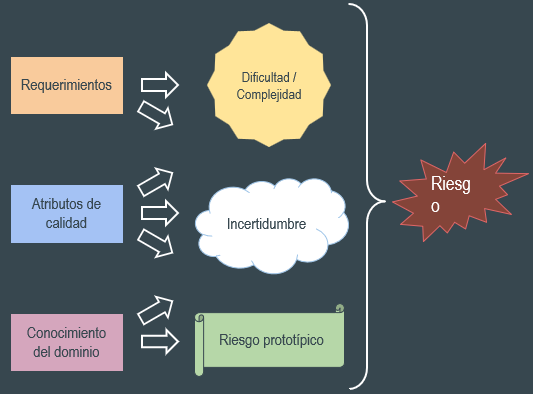
Se asocian a la calidad y antiguamente a la arquitectura (ahora se usa requerimientos significativos en arquitectura)

## Riesgos

Ayuda a priorizar los problemas y así garantizar las decisiones arquitectónicas. Para esto, se recomienda

1. Describir riesgo: definir escenarios de fracasos que sean medibles y accionables
2. Identificar el tipo de riesgo
   1. R. de Ingeniería: Se relaciona al análisis, diseño e implementación
   2. R. del proyecto: Relacionado con la temporalización, entregas, personal, etc.
3. Priorizar riesgos

### ¿Cómo identificar riesgos?



## Restricciones

Están dadas por los stackholders, la integración de sistemas y el ciclo de vida.

# Estilos de arquitectura

Se entiende por estilo de arquitectura al conjunto de decisiones de diseño aplicables a un contexto específico (Ya sea por comodidad de desarrollo o adecuación al proyecto). Estos restringen las decisiones arquitectónicas. Algunos estilos conocidos son:

| Familia | Estilos | Descripción |
| --- | --- | --- |
| Llamado y retorno | Programación principal y subrutinas | Realiza instrucciones o llamados a subrutinas hasta acabar todos los llamados. |
| Orientado a objetos | Se utiliza en sw de alta mantenibilidad. Llamado entre obj |
| Multinivel (Cliente-servidor) | Componentes intercomunicados con lógica específica, aplica en n niveles. |
| Flujo de datos | Lote secuencia | Programas se divide en partes que interactúan en tiempo real. generando salida solicitada |
| Tubos y filtros | Ejecuta un conjunto de programas en tiempo real generando la salida solicitada |
| Centrada en datos | Pizarrón | Centraliza la información, luego entrega salida |
| Centrado en bases de datos | Comunica componentes a través de la escritura y lectura de una bd (divide el sw) |
| Sistema experto (basado en reglas) | Utiliza un comp. de inferencia (capaz de identificar consultas y reglas), que utiliza una base de conocimiento basado en reglas, pudiendo ir rellenando las reglas y dar mejores respuestas a las solicitudes del usuario. |
| Componentes independientes | Invocación implícita (Basado en eventos.) | Los componentes publican eventos en un bus de eventos (*Publicar y subscribir o Orientado a servicios 2.0...*) y estos lo comunican a los interesados sin conocerse. |
| invocación explícita  (Orientado a servicios 1.0) | Ocurre similar al anterior, pero aquí los componentes si se conocen entre sí, entregando vía de comunicación disponible (pe. prod. enterprice, ecommerce, sist. pago) |

## Comparación de estilos

| Tipo | Favorecen |
| --- | --- |
| Monolíticos | Eficiencia, curva de aprendizaje, capacidad de test y de modificación |
| Distribuidos | Modularidad, disponibilidad, use de recursos y adaptabilidad |

# Atributos de calidad

## Idoneidad Funcional

Conecta las necesidades del usuario con lo implementado dentro del software.

| Carac. (3) | Descripción | Medición (comparación de) |
| --- | --- | --- |
| **Completitud funcional** | Completitud en la implementación de los requerimientos | Funcionalidades implementadas  y requerimientos funcionales |
| **Exactitud funcional** | Precisión en la implementación de los requerimientos | Resultado esperado  y resultado obtenido |
| **Pertinencia funcional** | Cohesión de lo implementado con lo requerido | Objetivos cumplidos y  Objetivos del software |

## Eficiencia de ejecución

Mide el comportamiento del software a la hora de responder los requerimientos del usuario con los recursos necesarios.

| Característica (3) | Descripción |
| --- | --- |
| **Tiempo a comportamiento** | Tiempo entre la solicitud y la respuesta, comparado con el tiempo esperado o máximo tolerado. |
| **Uso de Recursos** | Consumo de recursos y consumo esperado (ram, hdd, etc). |
| **Capacidad** | Tolerancia detectada y tolerancia esperada de solicitudes. |

## Compatibilidad

| Atributo (2) | Descripción | Medición |
| --- | --- | --- |
| **Interoperabilidad** | Facilidad de conexión con sistemas internos y externos | a través de cumplimiento de estándares, p.e: HATEOAS, JSON Schema, SOAP, Open AP |
| **Coexistencia** | Funcionalidad a la hora de compartir con otros sistemas. | Medible a través de fallos ocurridos con sistemas externos |

## Usabilidad

| Características (6) | Descripción. |
| --- | --- |
| **Reconocimiento de idoneidad** | Relación entre conceptos de dominio y acciones del sistema. Palabras claves. |
| **Curva de aprendizaje** | Cantidad y tipo de consultas a soporte. Intuitividad. |
| **Operabilidad** | Pasos a realizar para lograr un objetivo. Métricas de conversión. |
| **Protección a errores** | Comparación entre cantidad de fallas e interacción. |
| **Estética de interfaz** | Relativo. Medible mediante encuestas de apreciación estética. |
| **Accesibilidad** | Busca asegurar el acceso de usuarios vulnerables o discapacitados a través de la adhesión a distintos estándares. |

## Confiabilidad

| Atributo (4) | Descripción |
| --- | --- |
| **Madurez** | Tiempo medio entre fallos |
| **Disponibilidad** | Cantidad de tiempo fuera de servicio (en %) |
| **Tolerancia a fallos** | *Chaos testing (Netflix)* |
| **Capacidad de recuperación** | Tiempo medio hasta la recuperación |

## Seguridad

| Atributos (5) | Confidencialidad |
| --- | --- |
| **Confidencialidad** | Autorización del sistema al usuario |
| **Traza de responsab.**  **Integridad** | Conectar acciones con los responsables |
| **Integridad** | Recaudos para proteger información |
| **Comprob. de hechos** | Conecta estado actual con hechos producidos por un tercero |
| **Autenticidad** | Identificación del usuario |

***Nota****: La técnica más importante para medir la seguridad es el* ***Penetration testing***

## Mantenibilidad

| Características (5) | Descripción |
| --- | --- |
| **Modularidad** | Capacidad de ser separado sin afectar su funcionalidad |
| **Reusabilidad** | Utilizar herramientas en áreas/proyectos distintos |
| **Cap. de análisis** | Conexión entre implementación y requerimientos. |
| **Cap. de Modificación** | Dificultad en cambiar el comportamiento sin romper funcionalidad (uso de test automatizado) |
| **Cap. de Prueba** | Dificultad en la creación de test. Recomendación: diseño por contrato (objetos puros) |

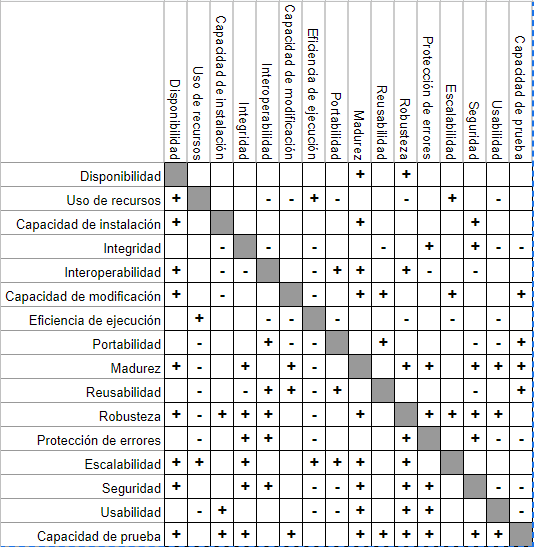
***Recomendaciones****: Cobertura de código en test y análisis estático de código.*

## Portabilidad

| Atributo (3) | Descripción |
| --- | --- |
| **Adaptabilidad** | Dependencias a entornos específicos (p.e.: SO) |
| **Capacidad de Instalación** | Requerimientos de despliegue (pasos del usuario) |
| **Capacidad de reemplazo** | Estándares o herramientas de carga/conversión de datos. Interfaces comunes. Versiones. |

## Tensiones entre atributos (ejemplos)

* Seguridad v/s Seguridad
* Modularidad v/s Instalación
* Eficiencia v/s tolerancia a fallos (resiliencia)
* Estética v/s funcionalidad



# Patrones de arquitectura

Los patrones de arquitectura se dividen en dos grupos:

1. Monolítico: El software se despliega(funciona/ejecuta) como una sola unidad
2. Distribuido: Distintos software pueden funcionar independientemente y ser modificados sin afectar a otros

| Patrón (10) | Descripción |
| --- | --- |
| **Big ball of mud** | Todas las partes interactúan entre ellas. No existe criterio. |
| **MVC** | Separa la aplicación en: Vista, Modelo y Controlador (representa las acciones del usuario) |
| **Capas** | Separa conceptos de la aplicación en capas (según requiera el sw). La comunicación debe ir de arriba hacia abajo (las inferiores no pueden solicitar información a las capas superiores) |
| **Orientado a eventos** | Conecta componentes a través de eventos. Un bus de eventos se convierte en el método principal de comunicación (lo que dificulta la lectura de estados) |
| **Microkernel o Plug-ins** | Puede ser monolítico o distribuido. Se integran distintos módulos/herramientas. p.e. IDEs |
| **Comparte-nada** | Sin unión entre component, potencia el procesamiento de información. |
| **Microservicios** | Componentes distribuidos donde c/u expone una funcionalidad al resto |
| **CQRS (Consultas - Comandos)** | Basado en sistemas de alta consulta (reportes)...Busca priorizar una de estas características. |
| **Hexagonal** | Para sistemas con dependencias externas que utilizan puertos y/o adaptadores.Facilita los test de unidad. |
| **Orientado a Dominio** | Guía el diseño hacia el lenguaje del negocio (no tan técnico). Enfoca los módulos según los contextos del producto |

***Nota****. Cada módulo puede tener interno su propia arquitectura.*

# Diseño de arquitectura: herramientas

Durante un proceso de diseño de arquitectura es necesario tener en consideración los puntos tratados con anterioridad: **Requerimientos**, **Atributos de calidad** y, Escenarios (frameworks) y tácticas (resumidos en un **Patrón de diseño**). Estos nos servirán herramienta a la hora es generar un buen diseño de arquitectura.

***Nota****: Es recomendable que antes de comenzar a diseñar un producto tengamos en cuenta las herramientas y modelos ya existentes que puedan facilitarnos el trabajo. En los casos donde nuestro problema está resuelto, será interesante buscar nuevos enfoque de solución que mejoren la ya existente.*

Podemos dividir la arquitectura en dos importantes partes:

## **Componentes**

Partes del sistema que cumplen una tarea/función específica. Estos pueden estar divididos en sub componentes y son quienes implementan las funciones de nuestro producto.

## **Conectores**

Hacen de comunicación entre los componentes. Estos no se limitan a un problema específico, pudiendo ser analizados de forma separada de los problemas a resolver.

| **Tipos** | Descripción |
| --- | --- |
| **Llamado a procesamiento** | Realizan la invocación de un componente a otro y esperan una respuesta |
| **Enlace** | Vinculan fuertemente un componente con otro, haciéndolo necesario para la compilación. |
| **Evento** | Permiten la escucha y notificación entre componentes (Listener) |
| **Adaptador** | Permite conectar/compatibilizar la interfaz de un comp. con la de otro. |
| **Acceso a datos** | Permite el acceso a recursos compartidos (db, archivos, apis, etc) |
| **Flujo** | Permite el acceso a datos que están en constante flujo |
| **Arbitraje** | Permite la coordinación entre componentes en acceso, trabajo, etc. |
| **Distribuidor** | Permite distribuir información de un componente a varios otros. |

También podemos clasificar nuestras conexiones según los siguientes parámetros:

| **Llamados** | Descripción |
| --- | --- |
| **Síncrono** | Emisor envía una solicitud a otro y espera la respuesta para seguir su ejecución. Delegando así el procesamiento. |
| **Asíncrono** | Un componente llama a otro sin detener su ejecución y evaluando la respuesta en algún momento determinado. |
| **Cliente - Servidor** | Comunicación desde el cliente al servidor, cliente puede no saber quien es el servidor. |

| **Conexión** | Descripción |
| --- | --- |
| **Enrutador** | Facilita conexión entre emisor y un grupo de receptores identificados |
| **Difusor** | Difunde mensaje a todos los receptores identificados |

| **Estructura** | Descripción |
| --- | --- |
| **Cola** | Busca compatibilizar la velocidad del productor con la del consumidor |
| **Pizarra o Repositorio** | Orientado a leer y/o escribir datos. Buscan garantizar estos procesos. |
| **Publicar - subscribir** | Permite enviar mensajes de eventos y que otros componentes suscriban a sin tener que conocerse los componentes. |

## Escenarios (Frameworks) y tácticas

**Framework de diseño orientado a atributos**: Busca priorizar en el diseño, la solución e implementación los requerimientos, riesgos y atributos de calidad que fueron definidos para el proyecto. Plantea una estructura de escenarios y tácticas, en dónde cada escenario relaciona un atributos de calidad con distintas técnicas de implementación buscando mejorar la calidad.

### Escenario de un framework genérico

| 1. **Estímulo**: cualquier hecho que afecte la calidad |
| --- |
| 1. **Técnicas de respuesta**: buscará controlar la respuesta al estímulo |
| 1. **Respuesta**: respuesta esperada o caso de éxito al implementar la técnica |

***Nota****: Algunos patrones de arquitectura ya implementan algunas tácticas.*

### Escenario de Disponibilidad

| 1. Falla |
| --- |
| 1. Técnicas para controlar la disponibilidad |
| 1. Falla ocultada o sistema reparado |

| Familia | Técnica | Descripción |
| --- | --- | --- |
| **Detección** | **Ping o Eco** | Se envía mensaje al componente de fallo |
| **Latido** | Comp. de fallo envía mensajes periódicamente |
| **Excepciones** | Catch. Notifica el cuándo y por qué del fallo |
| **Recuperación: Preparar o reparar** | **Votación** | Evalúa un cluster para identificar el que falla |
| **Redundancia activa** | Garantiza recibo del mje a todos los comp. Identificando en el que recibe el estímulo el fallo |
| **Redundancia pasiva** | Garantiza recibo del mje a través de un comp. líder que identificar el comp. seguidor de fallo |
| **Repuesto** | Considera un módulo backup para el cambio |
| **Recuperación: Reintroducción** | **Modo sombra** | Esconde comp. de fallo y lo evalúa en paralelo |
| **Sincronización de estado** | Actualiza o sincroniza el estado del componente que falla |
| **Punto de control o retroceso** | Marca estados consistentes del comp. para así retroceder cuando falle. |
| **Prevención** | **Quitar de servicio** | Se quita y no repara al comp. (o lo reinicia) |
| **Transacciones** | Usa bloques de cambios evitando inconsistencia |
| **Monitoreo de procesos** | Al monitorear los procesos, puede este reiniciarlos en caso de fallos |

### Escenario de Mantenibilidad

| 1. Pedido de cambio |
| --- |
| 1. Técnicas para controlar la mantenibilidad |
| 1. Cambio hecho, probado y desplegado |

| Familia | Técnica | Descripción |
| --- | --- | --- |
| **Confinar modificaciones** | **Coherencia semántica** | Respon. entre módulos (acoplamiento/cohesión). p.e: Extraer servicios comunes/genéricos. |
| **Generalizar** | Separa lo específico de lo general. |
| **Limitar opciones** | Limita a familia de comportamiento similar |
| **Anticipar cambios** | Dejar estrategias de probables cambio |
| **Prevenir efectos dominó** | **Ocultar información** | Encapsulamiento |
| **Mantener la interfaz** | Busca mantener formatos de entrega de datos |
| **Restringir comunicación** | Ley de demeter |
| **Intermediarios** | Compatibiliza módulos evitando acoplamiento |
| **Diferir enlace** | **Registro en ejecución** | Despliegue individual, une comp. en ejecución |
| **Archivo de configuración** | Identifica interfaces a usar durante ejecución. pe: plug-in (busca archivo compatible). |
| **Polimorfismo** | Objeto funciona según instancia de creación. |
| **Reemplazo de componentes** | Despliega un comp en ejecución y lo cambio con otro a través del reemplazo de enlace |
| **Adherir a protocolos** | Permite definir esquemas entre módulos |

### Escenario de Eficiencia de ejecución

| 1. Eventos |
| --- |
| 1. Técnicas para controlar la eficiencia |
| 1. Respuesta dentro del tiempo esperado |

| Familia | Técnica | Descripción |
| --- | --- | --- |
| **Demanda de recursos** | **Mejorar la eficiencia computacional** | Analizar el algoritmo, busca notación, rendimiento y mejora. Separar procesos |
| **Reducir sobrecarga** | Minimiza el procesamiento de eventos a través de datos almacenados u otros métodos |
| **Manejar tasa de eventos** | Reduce tasa de eventos (pe: unificandoles) |
| **Frecuencia de muestreo** | Agrupa eventos a procesar. (pe: Reportes) |
| **Gestión de recursos** | **Concurrencia** | Utiliza procesos paralelos. Distribuye trabajo |
| **Réplicas** | Duplica datos y facilita el acceso. ej: caché |
| **Aumentar recursos** | Mide y aumenta recursos a base de umbral |
| **Arbitraje de recursos** | **Políticas de planificación de tareas** | Ordena y prioriza el acceso a recursos. |

### Escenario de Seguridad

| 1. Ataque |
| --- |
| 1. Técnicas para controlar la seguridad |
| 1. Detección, resistencia o recuperación |

| Familia | Técnica | Descripción |
| --- | --- | --- |
| **Detectar ataques** | **Detectores de intrusos** | Implementa identificadores de anomalías (acciones, tráfico, uso, secuencias, etc). Usa: análisis de paquetes, limitar acceso, otros |
| **Resistencia al ataque** | **Autenticación** | Identifica al usuario |
| **Autorización** | Limita el acceso del usuario asignando roles |
| **Confidencialidad de datos** | Limita el acceso. ej: encriptación |
| **Integridad** | Garantiza que el mensaje igual al de envío |
| **Límite de exposición** | Aisla datos sensible de datos de uso común en el sistema |
| **Límite de acceso** | Limita acceso remoto usando un pto inicial |
| **Recuperación de ataques** | **Restauración** | Volver a estado consistente. |
| **Identificación** | Reproduce ataque con una traza de auditoría. Recuperando así el estado normal. |

### Escenario: Capacidad de prueba

| 1. Funcionalidad |
| --- |
| 1. Técnicas para controlar la capacidad de prueba |
| 1. Fallas detectadas |

| Familia | Técnica | Descripción |
| --- | --- | --- |
| **Entrada y salidas** | **Captura y reproducción** | Reutiliza comunicación en unit test. Garantizando el uso normal. ej:fallo. API: VCR |
| **Separar interfaz de implementación** | Realiza test con implementaciones controladas. Test-double/spy, Mock/Dummy-obj, etc |
| **Acceso exclusivo para pruebas** | Acceso específico para pruebas externas |
| **Monitoreo interno** | **Monitoreo incorporado** | Incorpora el motitoreo a la app y así saber cuando ocurra el error. |

### Escenario: Usabilidad

| 1. Pedido de un usuario |
| --- |
| 1. Técnicas para controlar la usabilidad |
| 1. Información y asistencia adecuada al usuario |

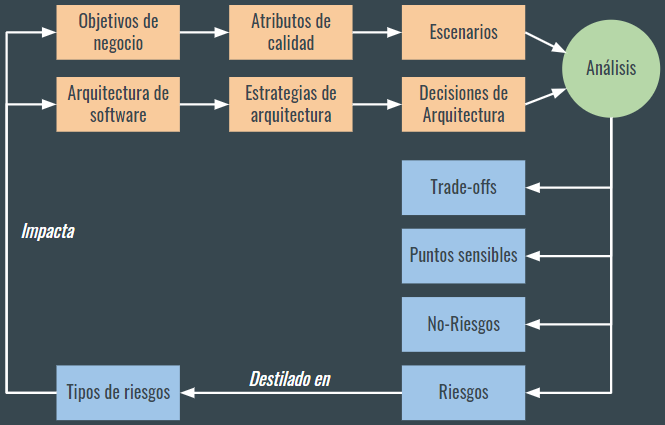
| Familia | Técnica | Descripción |
| --- | --- | --- |
| **Separar la interfaz de usuario** | **Mantenibilidad: coherencia semántica** | Coherencia semántica orientada a la GUI |
| **Iniciativas del usuario** | **Cancelar** | Dado una acción permite ser cancelada por el user |
| **Deshacer** | Dado una acción permite volver al estado anterior |
| **Agregación** | Permite la repetir o agrupar acciones |
| **Múltiples vistas** | Presenta la información necesaria al user |
| **Iniciativas del sistema** | **Modelo del usuario** | Verifica la pertinencia y validación de lo mostrado |
| **Modelo del sistema** | Brinda notificaciones del sistema al usuario (error, ejecucion, otros) |
| **Modelo de la tarea** | Permite la asistencia al usuario durante la ejecución de una tarea, permitiendo la realización exitosa |

# Validación del diseño: Arquitectura en evolución

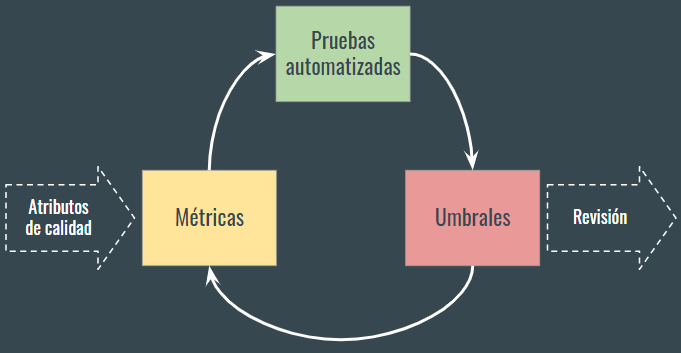
La validación de arquitecturas es un proceso que se debe realizar bien previo al desarrollo o durante cada iteración de éste (ágil). Existen distintos métodos y métricas para la toma de esta decisión, estos dependen del contexto de desarrollo.

### **ATAM** *(Architecture Trade-off Analysis Method)*

Es un ejemplo de método de validación de arquitectura, aquí los objetivos del negocio, atributos de calidad y escenarios se combinan con la arq., sus estrategias y decisiones. Cada stackholders da su opinión y voto para el análisis y toma de decisiones del proyecto.

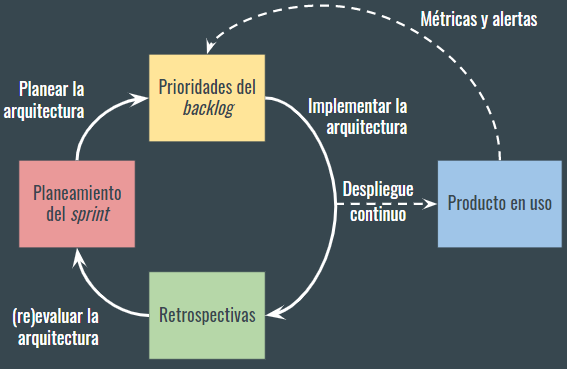


### Métricas de evaluación



### Arquitectura en evolución

La arquitectura tradicional asume que el primer diseño es suficientemente detallado para funcionar durante todo el desarrollo. Esto pocas veces es verídico, ya que muchos software van en evolución, por lo que se hace necesario una reevaluación constante de la arquitectura. A continuación algunas recomendaciones para incluirla en cada iteración.



# Modelado y Documentación de arquitecturas

* **Arquitectura restrictiva**: Restringe las decisiones a tomar.
* **Arquitectura descriptiva**: Documenta las decisiones tomadas

# Fuentes

* [Curso: Fundamento de arquitectura de software - Platzi](https://platzi.com/clases/arquitectura-software/)
* [El proceso de desarrollo de software](https://docs.google.com/presentation/d/1grQkWQaTYYoQHApmja51Chlg9gZhaDhwlhnXr-WsLrg/edit#slide=id.p)
* [Introduccion a arquitectura de software](https://docs.google.com/presentation/d/14_HifWbpVoGcXavIR-2s1zAd1aRH89gUgIaMQ0D5x-Y/edit#slide=id.p)
* [Análisis de Requerimientos](https://static.platzi.com/media/public/uploads/fdas-3-analisis-de-requerimientos_fe250e43-75b3-48b7-ab8d-90ac7fe7bbeb.pptx)
* [Estilos de arquitecturas](https://static.platzi.com/media/public/uploads/fdas-4-estilos-de-arquitectura_0a2426dc-ce58-4a99-a55a-972912d64a42.pptx)
* [Curso: Arquitectura de software - Platzi](https://platzi.com/clases/pro-arquitectura/)
* [Atributos de calidad](https://docs.google.com/presentation/d/1q00CqNXxAPn3J0YuZ1rdYAoqxL5EuB1W0Hhj1NZvceI/edit#slide=id.g30f07ce5c8_0_58)
* [Ejemplos de calidad](https://docs.google.com/presentation/d/1maSgvM4mXn7bldGlYEBHnOC_Lm9PY9B7CnfKXvg_lmg/edit#slide=id.g3416818d71_0_70)
* [Patrones de arquitectura](https://docs.google.com/presentation/d/1fP3vMpxz36MqI7MBGnmB90_FvtOQx6eXda1RzTu7jkU/edit)
* [Ejemplos de patrones (combinados)](https://docs.google.com/presentation/d/1maSgvM4mXn7bldGlYEBHnOC_Lm9PY9B7CnfKXvg_lmg/edit#slide=id.p)
* [Diseño de una arquitectura de software](https://docs.google.com/presentation/d/1SrNVT_j8De1TipH8GgGt-aq8vctd55wyinuEEO--UcE/edit#slide=id.g3309ebe1a7_0_83).
* [Modelado y documentación de arquitectura](https://docs.google.com/presentation/d/1tJsbh1b4w-JPzvCAJm-FB3ZXUg2DIMHtU2lWgDErCLw/edit#slide=id.p)