

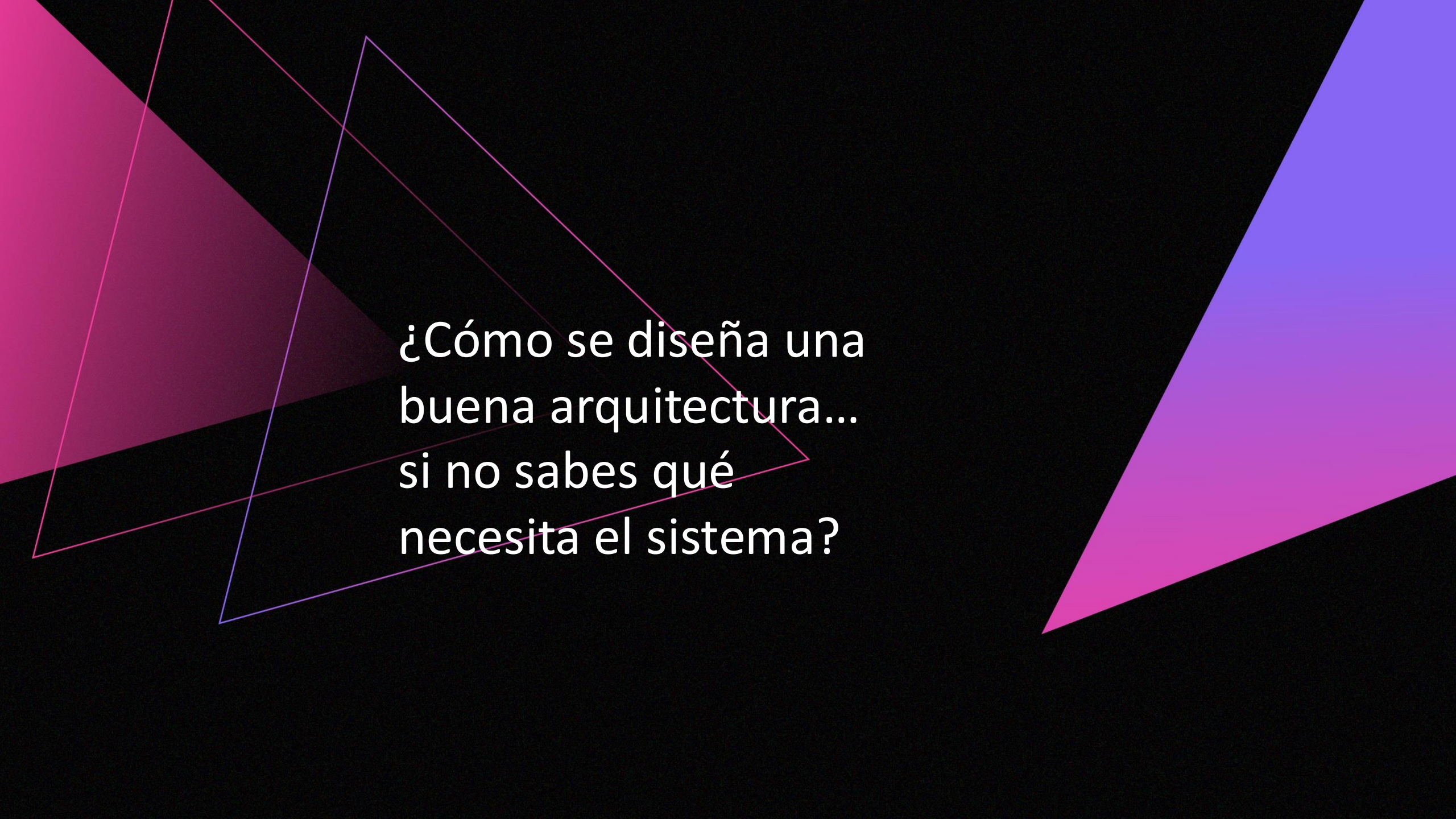
De los Requisitos a la Arquitectura

Architecture is the bridge between requirements and implementation.

— Rozanski & Woods (2011)

Objetivos de la Sesión

- Comprender cómo los **requisitos funcionales y no funcionales** impactan directamente en la arquitectura.
- Aprender a **elicitarlos, analizarlos y priorizarlos** desde una perspectiva arquitectónica.
- Reconocer que **no se puede diseñar arquitectura sin contexto ni propósito.**
- Aplicar principios de modelado de requisitos orientado a arquitectura.



¿Cómo se diseña una
buena arquitectura...
si no sabes qué
necesita el sistema?

La arquitectura no comienza con “qué tecnología usamos” Sino con “qué necesita lograr el sistema”

El error común es diseñar estructuras genéricas sin comprender qué debe resistir, soportar, cambiar... y sin preguntarse qué espera lograr el negocio con esa solución.

Diseñar arquitectura sin comprender **el contexto de uso y los objetivos del negocio** lleva a decisiones desconectadas, costosas o ineficaces. No hay “arquitectura buena” si no está **alineada con el propósito** que debe cumplir.

“Architecture is the bridge between requirements and implementation
— but those requirements must be driven by real business goals.”

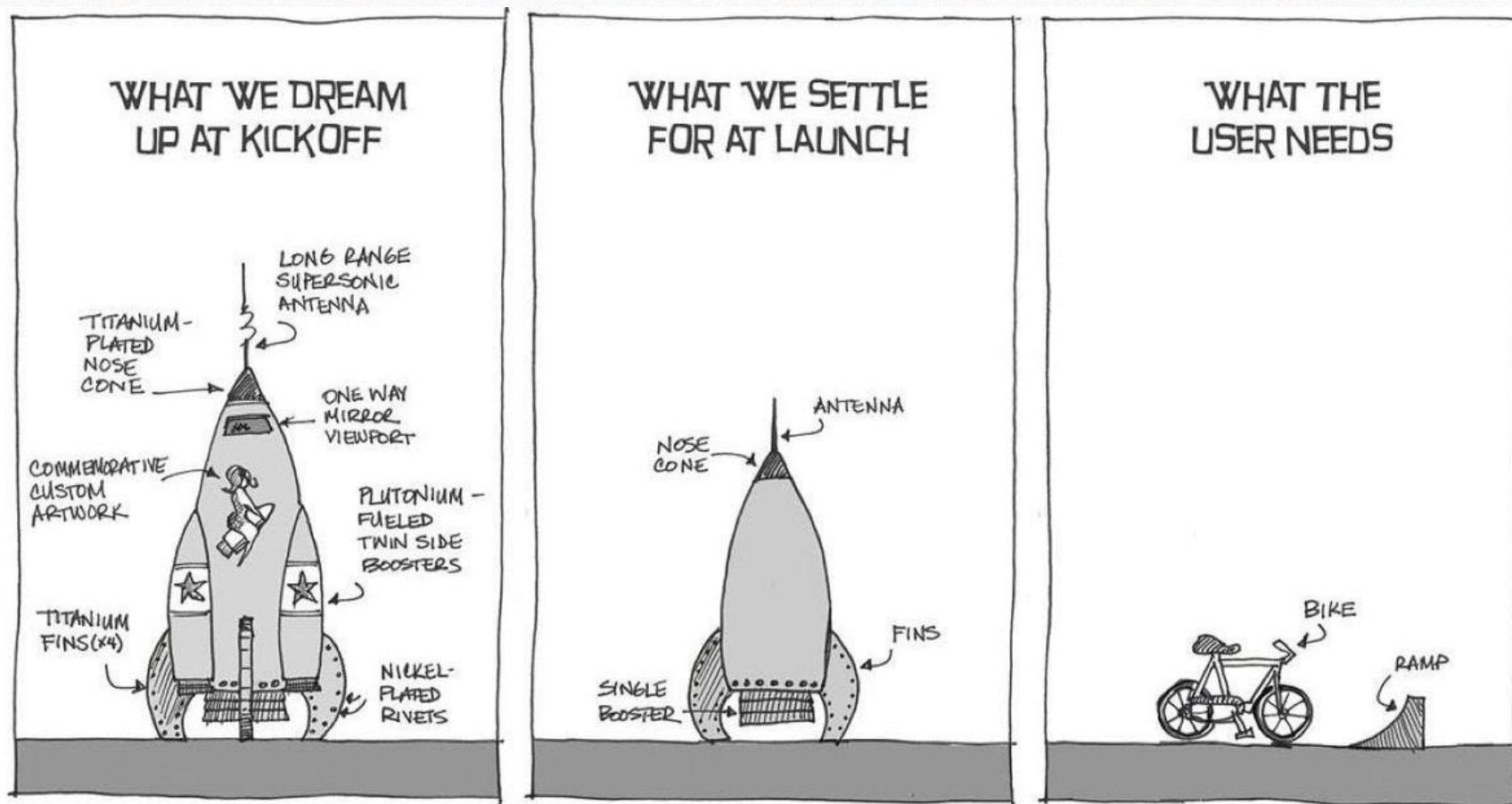
— Rozanski & Woods, *Software Systems Architecture*, 2nd ed., 2011



¿Tiene sentido invertir en una arquitectura altamente escalable si nadie ha definido cuántos usuarios se esperan en el primer año?

¿O hacerla modular si el producto será un MVP que debe salir en 6 semanas?

Arquitectura sin propósito → Overengineering



Arquitectura con propósito

La arquitectura no se diseña para ser “elegante” o “sofisticado”, sino para **cumplir un propósito contextual**: satisfacer restricciones, anticipar riesgos, facilitar la evolución, y **alinearse con lo que el sistema necesita sostener**.



Actividad:

Una startup busca construir una plataforma web donde:

- Freelancers de tecnología se registren, creen perfiles y pasen pruebas técnicas.
- Empresas publiquen proyectos o búsquedas de talento temporal.
- Un algoritmo de *matching* inteligente conecte a ambas partes según habilidades, disponibilidad y experiencia.
- Todo se gestione desde un dashboard: contratos, mensajes, pagos, historial.

Alcances funcionales iniciales

- Registro/login para freelancers y empresas.
- Perfil público y portafolio técnico.
- Sistema de publicación de proyectos.
- Algoritmo de recomendación básica.
- Chat interno y gestión de pagos por Stripe.
- Panel de administración para revisar casos de disputa.

Preguntas Generadoras:

Propósito de negocio:

- ¿Qué ventaja competitiva ofrece esta plataforma?
- ¿Qué tipo de crecimiento espera el negocio (usuarios, regiones, productos)?
- ¿El foco es escalar rápido o validar un MVP?

Contexto técnico:

- ¿Se integrará con plataformas como LinkedIn o GitHub?
- ¿Qué stack domina el equipo de desarrollo actual?
- ¿Hay que soportar navegación móvil nativa desde el inicio?

Condiciones operativas:

- ¿Cuántos usuarios se espera que estén en simultáneo?
- ¿Habrá picos de tráfico (ej. hackathons, lanzamientos)?
- ¿Qué pasa si falla el módulo de pagos?

Cultura organizacional:

- ¿Tienen DevOps implementado? ¿CI/CD?
- ¿Cuánto soporte hay para observabilidad y monitoreo?
- ¿Trabajan con equipos distribuidos?

Evolución esperada:

- ¿Se espera escalar a más industrias o solo TI?
- ¿El algoritmo será sustituido por IA más adelante?
- ¿Se migrará a microservicios en el futuro?



Requisitos

“Un requisito es una propiedad que el sistema debe exhibir para ser considerado aceptable por algún grupo de interesados.”

— Sommerville, *Software Engineering*, 10.^a ed.

“Los requisitos son la fuente de decisiones arquitectónicas. Una arquitectura es buena si permite cumplirlos de forma sostenible.”

— Bass, Clements & Kazman, *Software Architecture in Practice*, 4.^a ed.



Clasificación de los requisitos

Requisitos funcionales:

“Los requisitos funcionales describen el comportamiento del sistema en respuesta a entradas, eventos y estados.”

— Sommerville, *Software Engineering*, 10ª ed.

También se conocen como: funcionalidades, capacidades, “lo que hace el sistema”.

Características

- Son explícitos y fácilmente observables por el usuario.
- Se describen mediante casos de uso, historias de usuario, diagramas de flujo o BPMN.
- Son validados directamente con stakeholders.

¿Cómo los recopila el arquitecto?

Aunque no es su función primaria redactarlos, el arquitecto **participa en su validación** para entender los módulos, procesos y dependencias del sistema.

- Asiste a sesiones con analistas, stakeholders o product owners.
- Puede hacer preguntas como:
 - ¿Cuáles son los flujos críticos del negocio?
 - ¿Qué funcionalidades deben estar disponibles desde el primer día?



Clasificación de los requisitos

Requisitos no funcionales (atributos de calidad):

“Los requisitos no funcionales establecen criterios que juzgan el funcionamiento de un sistema, más allá de comportamientos específicos. .”

— IEEE Std 830-1998

También conocidos como: atributos de calidad, restricciones del sistema, “el cómo” del sistema.

Características

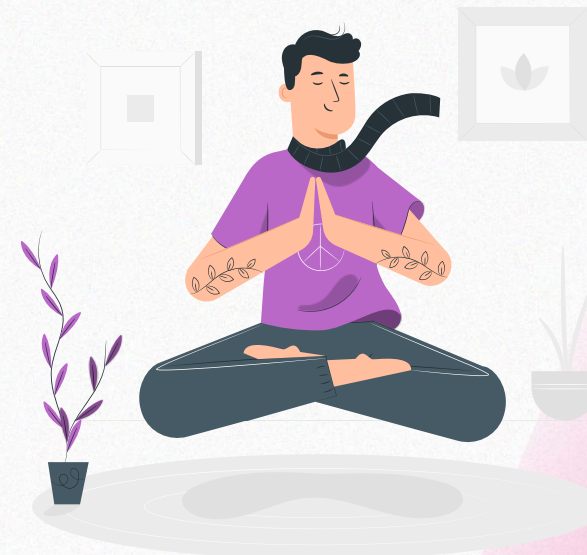
- Son menos visibles para el usuario, pero críticos para el éxito del sistema.
- Incluyen: rendimiento, seguridad, disponibilidad, mantenibilidad, escalabilidad, usabilidad.
- Requieren ser cuantificados mediante *Quality Attribute Scenarios*.

¿Cómo los recopila el arquitecto?

Los stakeholders no los formulan claramente. El arquitecto debe “leer entre líneas”.

Se obtienen mediante:

- Entrevistas técnicas y no técnicas (usuarios, infraestructura, seguridad, negocio).
- Talleres con stakeholders usando plantillas como Stimulus–Response–Environment.
- Revisión de sistemas similares anteriores.
- Experiencia del arquitecto en sistemas comparables.



Atributos de Calidad (Quality Attributes)

Los atributos de calidad son los principales impulsores de las decisiones arquitectónicas.”

— Bass, Clements & Kazman, *Software Architecture in Practice*, 4.^a ed.

No dictan *qué* debe hacer el sistema, sino *cómo debe comportarse* para ser confiable, útil y sostenible en el tiempo.

Una arquitectura sin atención a estos atributos puede ser:

- Rápida pero frágil
- Escalable pero inmantenible
- Segura pero inusable

Seguridad (Security)

Rendimiento (Performance)

Escalabilidad (Scalability)

Mantenibilidad (Maintainability)

Testabilidad (Testability)

Auditabilidad / Trazabilidad

Quality Attribute Scenarios

No basta con decir “quiero que sea rápido” o “debe ser seguro”.

“Los Quality Attribute Scenarios son descripciones estructuradas que expresan cómo debe comportarse un sistema ante una situación que pone a prueba un atributo de calidad.”

— Bass et al., *Software Architecture in Practice*, 4.^a ed.

Un **QAS** define:

1. **Fuente del estímulo** (¿Quién provoca el evento?)
2. **Estímulo** (¿Qué ocurre?)
3. **Entorno** (¿En qué condiciones?)
4. **Artefacto** (¿Qué parte del sistema está involucrada?)
5. **Respuesta** (¿Qué debe hacer el sistema?)
6. **Medida de respuesta** (¿Cuánto, cuándo, con qué éxito?)

Cuando <estímulo>
desde <fuente del estímulo>
en <entorno>
sobre <artefacto>
el sistema debe <respuesta>
con <medida de respuesta>.

Cuando 10,000 usuarios simultáneos realizan búsquedas **desde** la interfaz web **en** hora pico (lunes 9:00–10:00 am) **sobre** el backend de búsqueda **el sistema debe** responder los resultados en menos de 2 segundos **con** 95% de cumplimiento sobre 100 pruebas consecutivas.

Escenario de rendimiento

Quality Attribute Scenarios

No basta con decir “quiero que sea rápido” o “debe ser seguro”.

“Los Quality Attribute Scenarios son descripciones estructuradas que expresan cómo debe comportarse un sistema ante una situación que pone a prueba un atributo de calidad.”

— Bass et al., *Software Architecture in Practice*, 4.^a ed.

Un **QAS** define:

1. **Fuente del estímulo** (¿Quién provoca el evento?)
2. **Estímulo** (¿Qué ocurre?)
3. **Entorno** (¿En qué condiciones?)
4. **Artefacto** (¿Qué parte del sistema está involucrada?)
5. **Respuesta** (¿Qué debe hacer el sistema?)
6. **Medida de respuesta** (¿Cuánto, cuándo, con qué éxito?)

Cuando <estímulo>
desde <fuente del estímulo>
en <entorno>
sobre <artefacto>
el sistema debe <respuesta>
con <medida de respuesta>.

Cuando un usuario no autenticado intenta acceder a /admin **desde** una red externa **en** un entorno productivo **sobre** el módulo de autorización **el sistema debe** bloquear el acceso y registrar el intento **con** un log almacenado en menos de 1 segundo.

Escenario de seguridad

Quality Attribute Scenarios

No basta con decir “quiero que sea rápido” o “debe ser seguro”.

“Los Quality Attribute Scenarios son descripciones estructuradas que expresan cómo debe comportarse un sistema ante una situación que pone a prueba un atributo de calidad.”

— Bass et al., *Software Architecture in Practice*, 4.^a ed.

Un **QAS** define:

1. **Fuente del estímulo** (¿Quién provoca el evento?)
2. **Estímulo** (¿Qué ocurre?)
3. **Entorno** (¿En qué condiciones?)
4. **Artefacto** (¿Qué parte del sistema está involucrada?)
5. **Respuesta** (¿Qué debe hacer el sistema?)
6. **Medida de respuesta** (¿Cuánto, cuándo, con qué éxito?)

Cuando <estímulo>
desde <fuente del estímulo>
en <entorno>
sobre <artefacto>
el sistema debe <respuesta>
con <medida de respuesta>.

Cuando el servidor principal falla
desde un evento de caída de red **en**
el entorno de producción **sobre** el
balanceador de carga **el sistema**
debe redirigir el tráfico a una réplica
activa **con** un tiempo de
recuperación menor a 5 segundos.

Escenario de disponibilidad

Quality Attribute Scenarios

¿Por qué importa para el arquitecto?

- **Obliga a concretar** las propiedades abstractas que todo el mundo menciona, pero nadie define.
- **Permite diseñar decisiones específicas:** patrones, estilos, infraestructura, herramientas.
- **Ayuda a evaluar trade-offs:** ¿qué pasa si priorizo rendimiento sobre mantenibilidad?

Documentos típicos:

- *Architectural Vision Document* (en RUP)
- *Quality Requirements Specification* (en ISO/IEC 25010)
- *Quality Attribute Workshop Reports* (ATAM, SEI)
- *Architecture Decision Records (ADRs)* que referencian los QAS



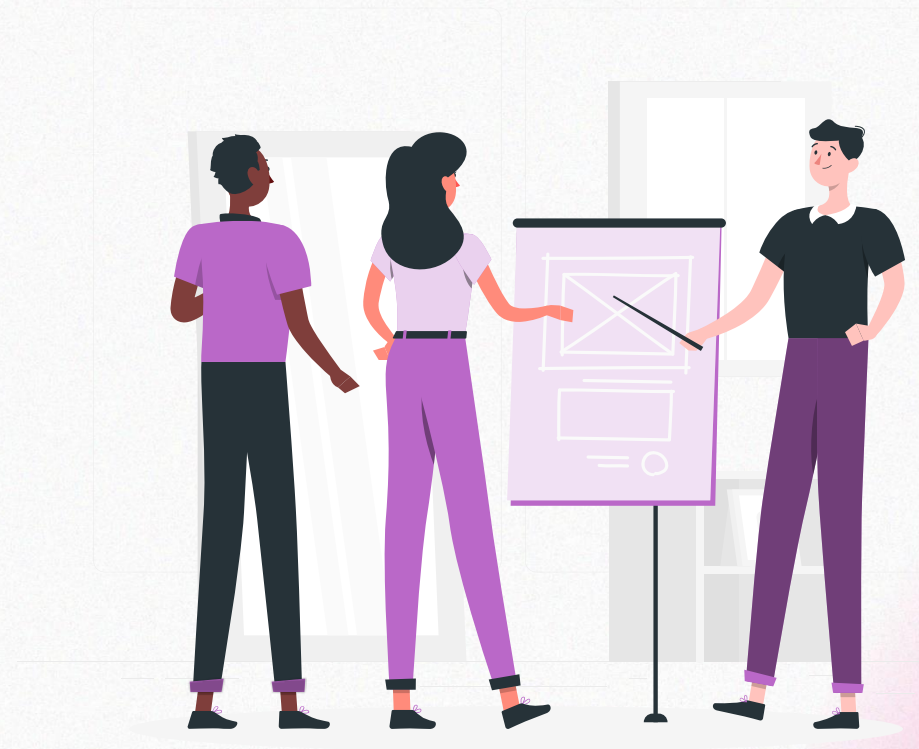
Quality Attribute Workshop (QAW)

El **Quality Attribute Workshop (QAW)** es una técnica desarrollada por el *Software Engineering Institute (SEI)* que se enfoca específicamente en la **elicitación colaborativa de atributos de calidad** relevantes para una arquitectura de software. Es menos riguroso y más ágil que ATAM, y por eso es ideal para:

- Proyectos en etapas tempranas
- Equipos distribuidos o interdisciplinarios
- Arquitectos que necesitan insumos claros para tomar decisiones bien fundamentadas

¿Para qué se usa?

- Obtener **Quality Attribute Scenarios (QAS)** directamente de las partes interesadas.
- Poner sobre la mesa **lo que realmente importa** en términos de comportamiento no funcional.
- Detectar **potenciales riesgos, tensiones y restricciones** antes de definir la arquitectura.
- Alinear expectativas entre tecnología y negocio **desde el día 1**.



¿Cómo se priorizan los QAS si hay cientos?

Workshops con stakeholders (Quality Attribute Workshops, ATAM):

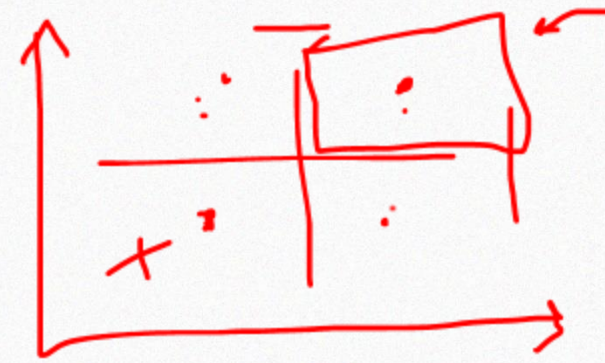
Se identifican *drivers* del negocio y se priorizan escenarios por impacto y riesgo.

Matriz impacto vs probabilidad:

- ¿Qué tan probable es que ese escenario ocurra?
- ¿Qué tan grave sería si fallara?

Asignación de puntajes con criterios como:

- Alineamiento con objetivos de negocio
- Riesgo técnico
- Esfuerzo de mitigación
- Costo asociado



Etapas del Quality Attribute Workshop (QAW)

Preparación (previo al workshop): El arquitecto prepara una descripción básica del sistema y sus objetivos.

Se convoca a un grupo representativo de stakeholders:

- Líderes técnicos
- Responsables de negocio
- DevOps
- QA
- Usuarios clave (si aplica)

Introducción y contexto:

- El arquitecto presenta el sistema propuesto, sus funcionalidades clave y su visión de alto nivel.
- Se repasan los objetivos de negocio.
- Se explican los atributos de calidad más comunes (usabilidad, disponibilidad, rendimiento, seguridad, etc.).

Generación de escenarios:

Aquí ocurre lo más importante del QAW

Cada stakeholder responde:

- ¿Qué situaciones te preocupan?
- ¿Qué pasaría si...?
- ¿Qué comportamientos del sistema son críticos?
- ¿Qué sería inaceptable para ti?

Priorización colectiva:

- Se listan todos los escenarios y se discute cuáles son más relevantes.
- Se pueden usar votaciones, matrices de impacto/riesgo, o simplemente discusión dirigida.
- El objetivo es salir con una **shortlist de atributos críticos** y sus escenarios prioritarios.

Actividad

Convertir frases vagas en QAS completos

- “El sistema debe escalar.”
- “Tiene que ser fácil de mantener.”
- “Debe cumplir con estándares de seguridad.”
- “Tiene que ser rápido.”
- “Debe resistir alta demanda.”



Clasificación de los requisitos

Requisitos organizacionales y contextuales:

Son restricciones impuestas por la estructura, procesos o capacidades de la organización que desarrolla, mantiene o despliega el sistema.

— Rozanski & Woods, *Software Systems Architecture*, 2011

Características

- No están en el backlog ni en la documentación de negocio.
- Afectan decisiones técnicas (infraestructura, lenguaje, herramientas).

Ejemplos:

- “Solo podemos usar AWS porque ya tenemos convenio.”
- “Los equipos están distribuidos en husos horarios distintos.”
- “No tenemos DevOps maduro, los despliegues son manuales.”

¿Cómo los detecta el arquitecto?

- Reuniones con líderes técnicos, IT ops, RRHH, legal y seguridad.
- Participación en revisiones organizacionales o entrevistas internas.
- Exploración del ecosistema donde se desplegará o mantendrá el sistema.

Tip: Aunque muchas veces no se documentan, **son decisivos en la viabilidad real de la arquitectura.**



Clasificación de los requisitos

Requisitos organizacionales y contextuales:

Son restricciones impuestas por la estructura, procesos o capacidades de la organización que desarrolla, mantiene o despliega el sistema.

— Rozanski & Woods, *Software Systems Architecture*, 2011

Características

- No están en el backlog ni en la documentación de negocio.
- Afectan decisiones técnicas (infraestructura, lenguaje, herramientas).

Ejemplos:

- “Solo podemos usar AWS porque ya tenemos convenio.”
- “Los equipos están distribuidos en husos horarios distintos.”
- “No tenemos DevOps maduro, los despliegues son manuales.”

¿Cómo los detecta el arquitecto?

- Reuniones con líderes técnicos, IT ops, RRHH, legal y seguridad.
- Participación en revisiones organizacionales o entrevistas internas.
- Exploración del ecosistema donde se desplegará o mantendrá el sistema.

Tip: Aunque muchas veces no se documentan, **son decisivos en la viabilidad real de la arquitectura.**



Clasificación de los requisitos

Requisitos regulatorios o legales:

Conjunto de normas, estándares o leyes externas que el sistema debe cumplir para operar legalmente o con seguridad.

— ISO/IEC/IEEE 29148:2018

Características

Impuestos por organismos reguladores (nacionales o internacionales).

- No son negociables ni diferibles.

Ejemplos:

- Leyes de protección de datos: GDPR, HIPAA.
- Estándares técnicos: PCI-DSS, ISO/IEC 27001.

¿Cómo los detecta el arquitecto?

- Reuniones con líderes técnicos, IT ops, RRHH, legal y seguridad.
- Participación en revisiones organizacionales o entrevistas internas.
- Exploración del ecosistema donde se desplegará o mantendrá el sistema.

Tip: Aunque muchas veces no se documentan, **son decisivos en la viabilidad real de la arquitectura.**





Elicitación

Elicitación

Es el proceso mediante el cual el arquitecto **descubre, refina y valida** los requisitos que afectarán la estructura del sistema: principalmente, los **atributos de calidad**, restricciones, prioridades organizacionales y escenarios de riesgo.

A diferencia del analista funcional, el arquitecto no solo pregunta “¿qué debe hacer el sistema?”, sino “**¿qué debe soportar?**”, “**¿qué podría fallar?**”, “**qué podría cambiar?**”, “**qué es inaceptable?**”

1. Escenarios de calidad crítica (QAS)
2. Restricciones organizacionales/tecnológicas
3. Tolerancia al fallo y al cambio
4. Condiciones operativas reales
5. Criterios de éxito no funcionales



Técnicas para Elicitar

Entrevistas y preguntas socráticas

No preguntar solo “¿qué quieres?”, sino “¿qué pasaría si...?”, “¿cómo sabrías si falló?”, “¿qué sería inaceptable?”

Herramienta: *Quality Attribute Scenarios (QAS)*

Workshops dirigidos

- Talleres tipo brainstorming o impact mapping
- Discusión sobre prioridades, riesgos y atributos críticos

Técnica recomendada: Quality Attribute Workshop (QAW)

ATAM — Architecture Tradeoff Analysis Method

Metodología formal para descubrir, validar y priorizar atributos de calidad mediante el análisis de trade-offs y escenarios críticos.



ATAM — Architecture Tradeoff Analysis Method

“ATAM permite evaluar de forma sistemática una arquitectura para determinar si satisface los requisitos de calidad esperados, y comprender las decisiones, riesgos y compensaciones involucradas.”

— Clements, Kazman & Klein, *Evaluating Software Architectures*, 2002

¿Por qué es importante?

- Ayuda a descubrir riesgos arquitectónicos **antes de codificar o desplegar**.
- Obliga a hacer visibles las **decisiones implícitas**.
- Facilita el diálogo entre stakeholders técnicos y de negocio.
- Refuerza la trazabilidad entre **requisitos, arquitectura y decisiones**.

Una arquitectura es tan buena como su alineación con el contexto y sus requisitos. ATAM permite evaluar eso con disciplina.”



¿Cuándo aplicar ATAM?

- En proyectos críticos o costosos (industria financiera, salud, defensa).
- Cuando hay **presión por atributos de calidad exigentes**.
- En migraciones estructurales (por ejemplo: monolito → microservicios).
- Cuando hay múltiples stakeholders y objetivos que **pueden entrar en conflicto**.



Fase 1:

Preparación y contexto

Fase 2:

Elicitación de escenarios de calidad

Fase 3:

Análisis estructural

Fase 4:

Conclusiones y recomendaciones



Representación visual: Diagrama de requisitos

¿Qué es un diagrama de requisitos?

Un **diagrama de requisitos** es una representación visual estructurada de los distintos requisitos de un sistema, sus relaciones jerárquicas, dependencias, restricciones y trazabilidad con actores u otros elementos del sistema.

Aunque no es obligatorio en todos los proyectos, es **muy utilizado en ingeniería de sistemas, sistemas críticos, proyectos regulados**, y en metodologías basadas en **MBSE (Model-Based Systems Engineering)**.

Tipo de requisito	Estereotipo (SysML)	Ejemplo
Funcional	<<functional>>	“El sistema debe permitir login”
No funcional (calidad)	<<performance>>, etc.	“Tiempo de respuesta < 2s”
Requisito de interfaz	<<interface>>	“Usar API RESTful” —
Requisito legal/regulatorio	<<regulatory>>	“Cumplir con GDPR” —
Restricción técnica	<<constraint>>	“Base de datos debe ser Oracle” —
Derivado o de sistema	<<derived>>, <<system>>	“Debe escalar horizontalmente”

¿Qué es un diagrama de requisitos?

Un **diagrama de requisitos** es una representación visual estructurada de los distintos requisitos de un sistema, sus relaciones jerárquicas, dependencias, restricciones y trazabilidad con actores u otros elementos del sistema.

Aunque no es obligatorio en todos los proyectos, es **muy utilizado en ingeniería de sistemas, sistemas críticos, proyectos regulados**, y en metodologías basadas en **MBSE (Model-Based Systems Engineering)**.

Elemento	Función
Bloque de requisito	Rectángulo con id, texto, tipo
Relaciones	Permiten modelar jerarquía y dependencias
Trazabilidad	Permite vincular con actores, casos de uso, componentes

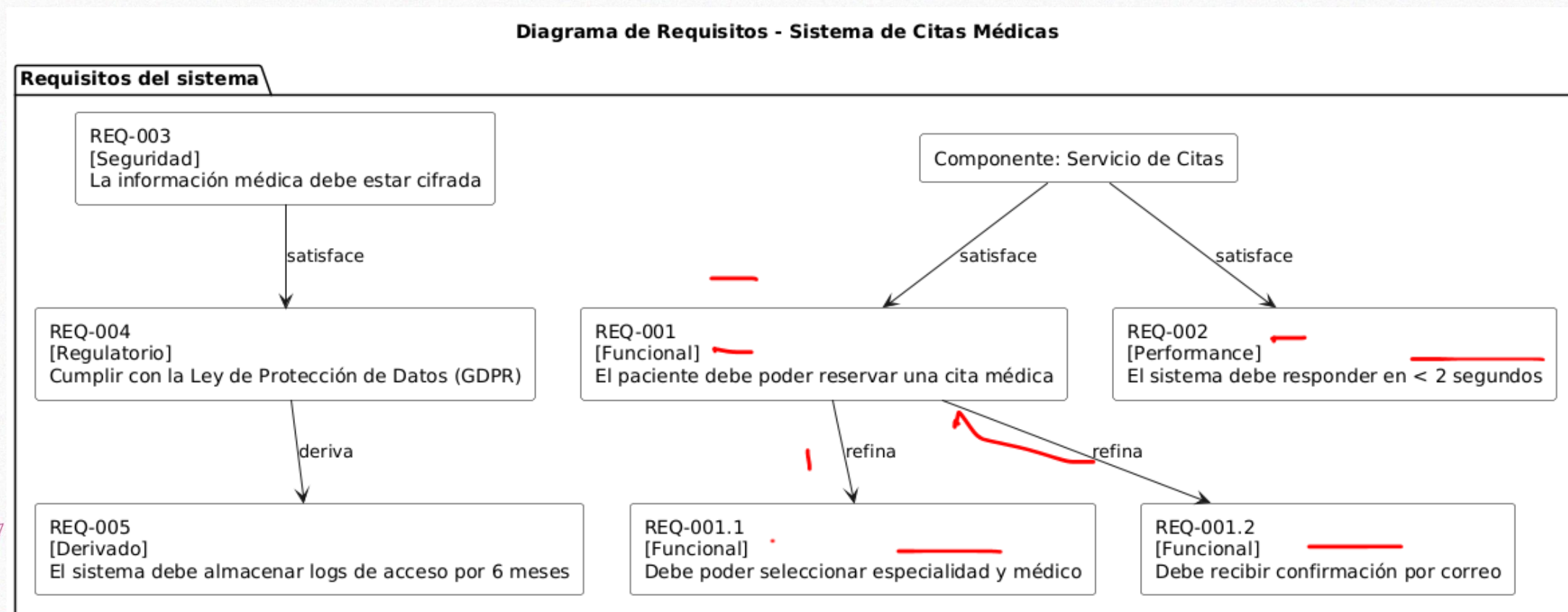
¿Qué es un diagrama de requisitos?


Un **diagrama de requisitos** es una representación visual estructurada de los distintos requisitos de un sistema, sus relaciones jerárquicas, dependencias, restricciones y trazabilidad con actores u otros elementos del sistema.

Aunque no es obligatorio en todos los proyectos, es **muy utilizado en ingeniería de sistemas, sistemas críticos, proyectos regulados**, y en metodologías basadas en **MBSE (Model-Based Systems Engineering)**.

Relación	Descripción
contains	Un requisito contiene subrequisitos más detallados
refines	Un requisito especializado a partir de uno general
satisfies	Un componente o caso de uso satisface un requisito
verifies	Un test verifica un requisito
tracedTo	Conexión con elementos que lo originan (actor, objetivo)

¿Qué es un diagrama de requisitos?





**¿Ya tenemos requisitos,
y ahora qué?**

¿Un requisito funcional basta para
saber cómo debe comportarse un
sistema ante un usuario real?

¿Un requisito funcional basta para saber cómo debe comportarse un sistema ante un usuario real?

- Los requisitos nos dicen *qué se espera*, pero no *cómo se interactúa*.
- El arquitecto necesita escenarios, flujos, actores, excepciones.
- “Los requisitos responden al *qué*, pero la arquitectura debe responder al *cómo* soportarlo.”

Enfoque tradicional

El arquitecto trabaja **con casos de uso detallados desde el inicio**, identifica atributos de calidad a partir de ellos y define la arquitectura en base a vistas derivadas de esos flujos.

- IEEE 830 (especificación de requisitos)
- Jacobson (1992) – Casos de uso como narrativa detallada
- Sommerville (Software Engineering, 10.^a ed.)

“Un caso de uso es una secuencia de interacciones entre un actor y el sistema que produce un resultado observable de valor para el actor.”

Jacobson, 1992

Enfoque ágil

Se elaboran User Stories, que describen mediante una declaración simple la funcionalidad objetivo el usuario, y el propósito, además de los criterios de aceptación que determinaran su estado de completado.

- Cohn (2004) – *User Stories Applied*
- Bass et al. (2021) – *SAiP* capítulo sobre arquitectura en ambientes ágiles
- Ford et al. (2017) – *Building Evolutionary Architectures*

¿Cómo se construyen los casos de uso?

Diferencia entre “lo que se necesita” y “cómo se usa”.

¿Quién participa?

- **Arquitecto de software** (facilitador técnico)
- **Analistas de negocio / sistemas**
- **Usuarios clave o representantes del cliente**
- **Product owner** (si ya existe en el proceso)
- A veces: líderes de QA, diseñadores de UX

¿Dónde ocurre?

- **Workshops de levantamiento funcional**
→ Estructurados, con dinámicas para modelar flujos
- **Entrevistas individuales** a usuarios clave
- **Análisis de documentación existente**
- **Observación del sistema actual** (si hay)

¿Qué actividades o herramientas se usan?

- Plantillas de casos de uso narrativo
- Mapa de actores → roles internos y externos
- Modelado de flujo en papel / pizarra / herramientas UML
- Técnicas como “event storming” o “journey mapping”
- Tormenta de excepciones: *¿Qué puede fallar en este flujo?*



¿Cómo se construyen los casos de uso?

Diferencia entre “lo que se necesita” y “cómo se usa”.

Preguntas clave que orienta el arquitecto:

1. *¿Qué espera lograr el usuario al iniciar esta acción?*
2. *¿Cómo comienza y cómo sabemos que terminó con éxito?*
3. *¿Qué puede fallar o qué escenarios alternativos existen?*
4. *¿Qué actor inicia el flujo? ¿El sistema o la persona?*
5. *¿Qué dependencias existen con otros sistemas?*
6. *¿Qué atributos de calidad se ven implicados en este flujo?*

Resultado esperado:

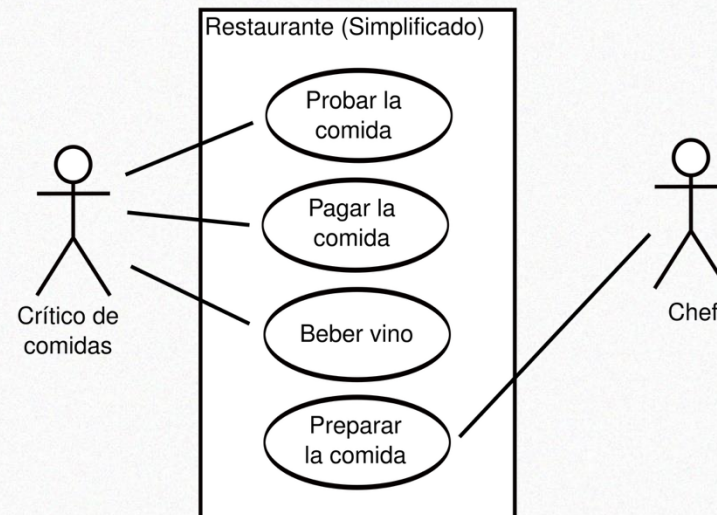
- Lista priorizada de **casos de uso**
- Flujos narrativos o diagramas de casos de uso (UML)
- Conexión con **atributos de calidad**, vistas y decisiones arquitectónicas



Antomia de un Caso de Uso

Elementos principales:

- Nombre (verbo activo)
- Actor(es) involucrado(s)
- Flujo principal
- Flujos alternativos y excepciones
- Precondiciones / postcondiciones
- Requisitos no funcionales implícitos



Especificación del caso de uso: el cliente llama al hotel			
Código	1		
Nombre	Llama al hotel		
Descripción	Este caso de uso permite al sistema ingresar datos del cliente		
Autores	Jorge y yamile		
Fecha creación	mayo - 2009	Fecha última modificación	Mayo20-2009
Actores	repcionista, sistema		
Precondición	El recepcionista debe tener el usuario y contraseña valida para poder acceder al sistema		
Pos condición	El recepcionista puede acceder al sistema		
Flujo normal	1. Ingresar nombre de usuario 2. Ingresar contraseña 3. Validar datos		
Excepciones	Ninguna		
Anotaciones	Ninguna		

Especificación del caso de uso: brinda datos			
Código	2		
Nombre	Brinda datos		
Descripción	Este caso de uso permite al recepcionista verificar datos del usuario para poder saber si es habitual o esporádico		
Autores	Jorge y yamile		
Fecha creación	mayo - 2009	Fecha última modificación	Mayo 20-2009
Actores	Cliente y recepcionista		
Precondición	El usuario tiene que dar el numero de cedula		
Pos condición	Si es habitual el cliente ya debe de estar registrado		
Flujo normal si ya esta registrado	1. Solicitar numero de cedula 2. verificar nombre 3. ofrecer habitaciones 4. confirmar fecha de ocupación 5. ofrecer descuento por ser cliente habitual 6. especificar números de personas 7. reservar habitación en el sistema		
Excepciones	Si el usuario no esta registrado ingresar todos los datos personales y no ingresar descuentos		

Actividad

Del requerimiento al caso de uso

Aplicaremos lo aprendido sobre casos de uso para transformar un conjunto de requisitos funcionales en una representación estructurada de interacciones entre usuario y sistema.

Sistema: App de compras en línea

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.



Actividad

Del requerimiento al caso de uso

Aplicaremos lo aprendido sobre casos de uso para transformar un conjunto de requisitos funcionales en una representación estructurada de interacciones entre usuario y sistema.

Sistema: App de compras en línea

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.

Paso 1:

Identificar actores

¿Quién inicia la acción?

¿Es una persona o un sistema?

¿Hay más de un actor implicado?

Actividad

Del requerimiento al caso de uso

Aplicaremos lo aprendido sobre casos de uso para transformar un conjunto de requisitos funcionales en una representación estructurada de interacciones entre usuario y sistema.

Sistema: App de compras en línea

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.

Paso 2:

Definir el caso de uso

Elemento	Descripción
Nombre	Verbo activo + objetivo
Actor principal	¿Quién inicia la acción?
Flujo principal	Pasos desde inicio hasta éxito
Flujos alternativos	Rutas posibles diferentes al éxito
Excepciones	Fallos, validaciones, errores externos
Pre/postcondiciones	Estado antes y después del caso de uso

Paso 2:

Definir el caso de uso

Ejem RF1 – Buscar Productos

Nombre	Buscar productos
Actor principal	Cliente
Flujo principal	<ol style="list-style-type: none">1. El cliente ingresa término de búsqueda2. El sistema muestra los resultados3. El cliente puede aplicar filtros opcionales4. El cliente selecciona un producto para ver detalles
Flujos alternativos	<p>3a. No hay coincidencias en la búsqueda</p> <p>3b. El cliente borra el filtro y vuelve a intentar</p>
Excepciones	El sistema no responde / error de red
Precondiciones	El cliente está autenticado o navega como invitado
Postcondiciones	Se muestran los resultados al cliente

Actividad

Del requerimiento al caso de uso

Aplicaremos lo aprendido sobre casos de uso para transformar un conjunto de requisitos funcionales en una representación estructurada de interacciones entre usuario y sistema.

Sistema: App de compras en línea

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.

Actividad

Del requerimiento al caso de uso

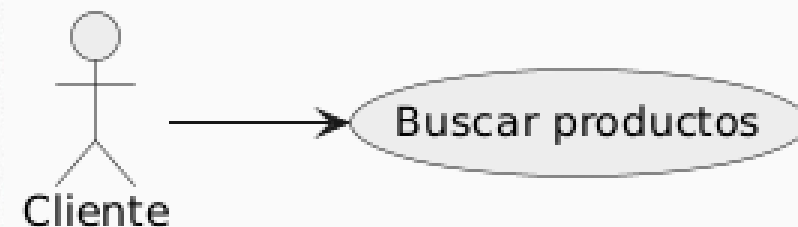
Aplicaremos lo aprendido sobre casos de uso para transformar un conjunto de requisitos funcionales en una representación estructurada de interacciones entre usuario y sistema.

Sistema: App de compras en línea

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.

Paso 3:

Representación Visual (*Si hay tiempo*)



¿Cómo se construye una historia de Usuario?

¿Quién participa?

- **Equipo completo** (PO, devs, UX, QA, arquitecto)
- El **Product Owner lidera**, pero el arquitecto acompaña y plantea preguntas técnicas clave

¿Dónde ocurre?

- **Refinamiento del backlog (grooming)**
- **Workshops de discovery** o inception
- Sesiones de **user story mapping**
- Conversaciones continuas dentro del equipo

¿Qué se hace?

- Redactar historias de usuario en formato: *Como [rol], quiero [acción], para [beneficio]*
- Asociar criterios de aceptación
- Descomponer épicas en historias manejables
- Identificar historias que afectan decisiones estructurales



¿Cómo se construye una historia de Usuario?

Preguntas clave del arquitecto:

1. ¿Qué parte del sistema se verá afectada con esta historia?
2. ¿Esta historia depende de otra historia?
3. ¿Hay implicancias de escalabilidad, seguridad o rendimiento?
4. ¿Qué sucede si esta historia falla o queda incompleta?
5. ¿Necesitamos visibilidad técnica para tomar esta historia?

Título de la Historia

Como <role, persona>

Persona o rol de usuario que tiene la necesidad

Quiero <objetivo, comportamiento>

Lo que se quiere obtener: una funcionalidad, característica, etc.

Para <motivo, razón, valor>

Motivo por el que se necesita, valor que se obtiene como resultado, etc.

¿Cuándo se escriben y detallan?

Inicialmente (Alto nivel)

- Durante la **creación del backlog**
- Como **épicas** o funcionalidades generales
- No se detallan aún → sirven para **explorar alcance** y **priorizar**

Justo a tiempo (Detalladas en refinamiento)

- Se detallan justo **antes de entrar a un sprint**
- Esto se llama **refinamiento del backlog (backlog grooming)**
- Solo se pulen las historias candidatas a ser trabajadas pronto



Actividad

Transformaremos funcionalidades generales del sistema en historias de usuario claras, orientadas al valor y preparadas para su discusión técnica.

Redactar historias de usuario bien estructuradas a partir de requerimientos funcionales, identificando criterios de aceptación, riesgos arquitectónicos y notas técnicas.

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.



Actividad

Transformaremos funcionalidades generales del sistema en historias de usuario claras, orientadas al valor y preparadas para su discusión técnica.

Redactar historias de usuario bien estructuradas a partir de requerimientos funcionales, identificando criterios de aceptación, riesgos arquitectónicos y notas técnicas.

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.

Paso 1:

Redactar Historia de Usuario

Como [rol] **Quiero** [necesidad o acción] **Para** [beneficio o resultado]

Ejemplo:

Como cliente, **quiero** aplicar un cupón de descuento, **para** pagar menos en mi compra.

Actividad

Transformaremos funcionalidades generales del sistema en historias de usuario claras, orientadas al valor y preparadas para su discusión técnica.

Redactar historias de usuario bien estructuradas a partir de requerimientos funcionales, identificando criterios de aceptación, riesgos arquitectónicos y notas técnicas.

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.

Paso 2:

Priorizar (MoSCoW)

M	S	C	W
Must have	Should have	Could have	Won't have
Absolutamente esencial, debe ser incluido a cualquier costo.	Importante y solo debería omitirse luego de un análisis profundo.	Deseable que podrían ser incluidos idealmente si los recursos lo permiten.	Cosas que están fuera de scope, no realizables o contraproducentes.

Actividad

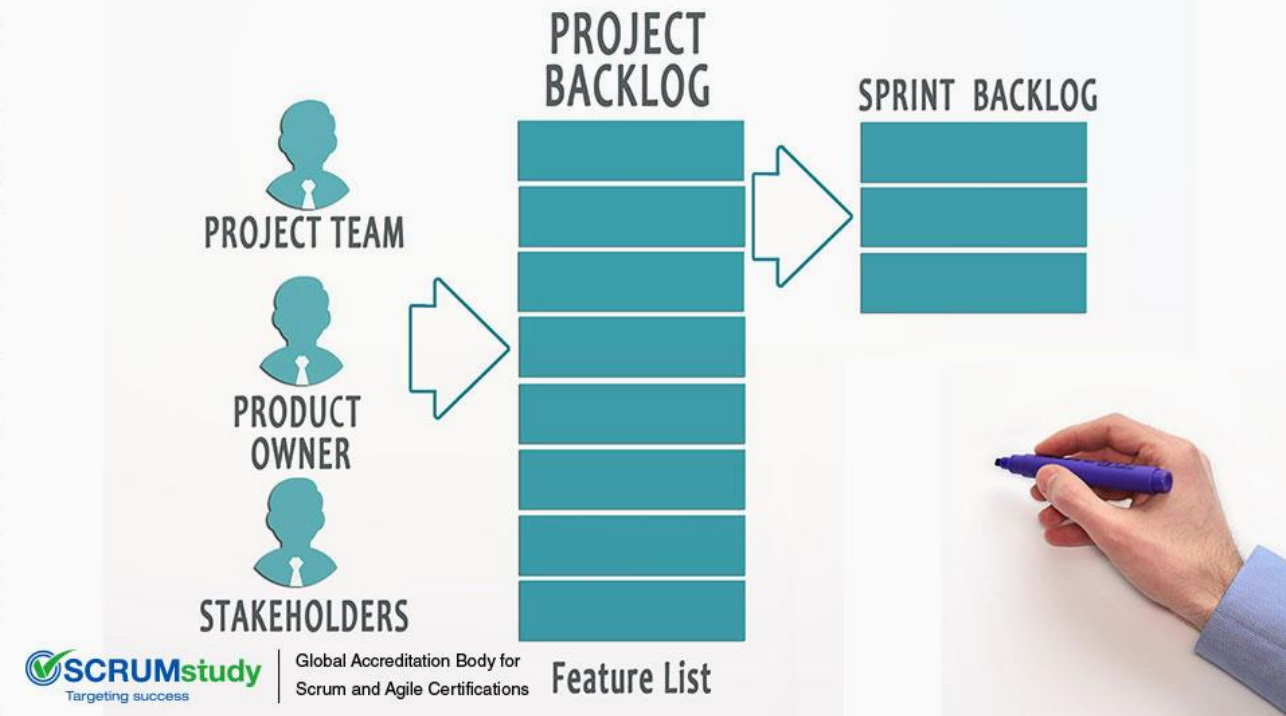
Transformaremos funcionalidades generales del sistema en historias de usuario claras, orientadas al valor y preparadas para su discusión técnica.

Redactar historias de usuario bien estructuradas a partir de requerimientos funcionales, identificando criterios de aceptación, riesgos arquitectónicos y notas técnicas.

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.

Paso 2:

Priorizar (MoSCoW)



Actividad

Transformaremos funcionalidades generales del sistema en historias de usuario claras, orientadas al valor y preparadas para su discusión técnica.

Redactar historias de usuario bien estructuradas a partir de requerimientos funcionales, identificando criterios de aceptación, riesgos arquitectónicos y notas técnicas.

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.

Paso 3:

Agregar criterios de aceptación

Establecer al menos 3 criterios de aceptación claros y verificables:

Ejemplo:

- El cupón debe tener un formato válido.
- El descuento debe aplicarse en tiempo real al total.
- El sistema debe indicar si el cupón ha expirado.

Actividad

Transformaremos funcionalidades generales del sistema en historias de usuario claras, orientadas al valor y preparadas para su discusión técnica.

Redactar historias de usuario bien estructuradas a partir de requerimientos funcionales, identificando criterios de aceptación, riesgos arquitectónicos y notas técnicas.

- RF1: El usuario debe poder buscar productos.
- RF2: El usuario debe poder agregarlos a un carrito.
- RF3: El usuario puede aplicar cupones de descuento.
- RF4: El usuario debe poder confirmar el pedido.
- RF5: El sistema debe enviar un correo de confirmación.

Paso 4:

Identificar notas técnicas / implicancias arquitectónicas

Ejemplo:

- Validación de cupones debe ser asincrónica si hay alta concurrencia.
- Se debe registrar el uso del cupón para auditoría.