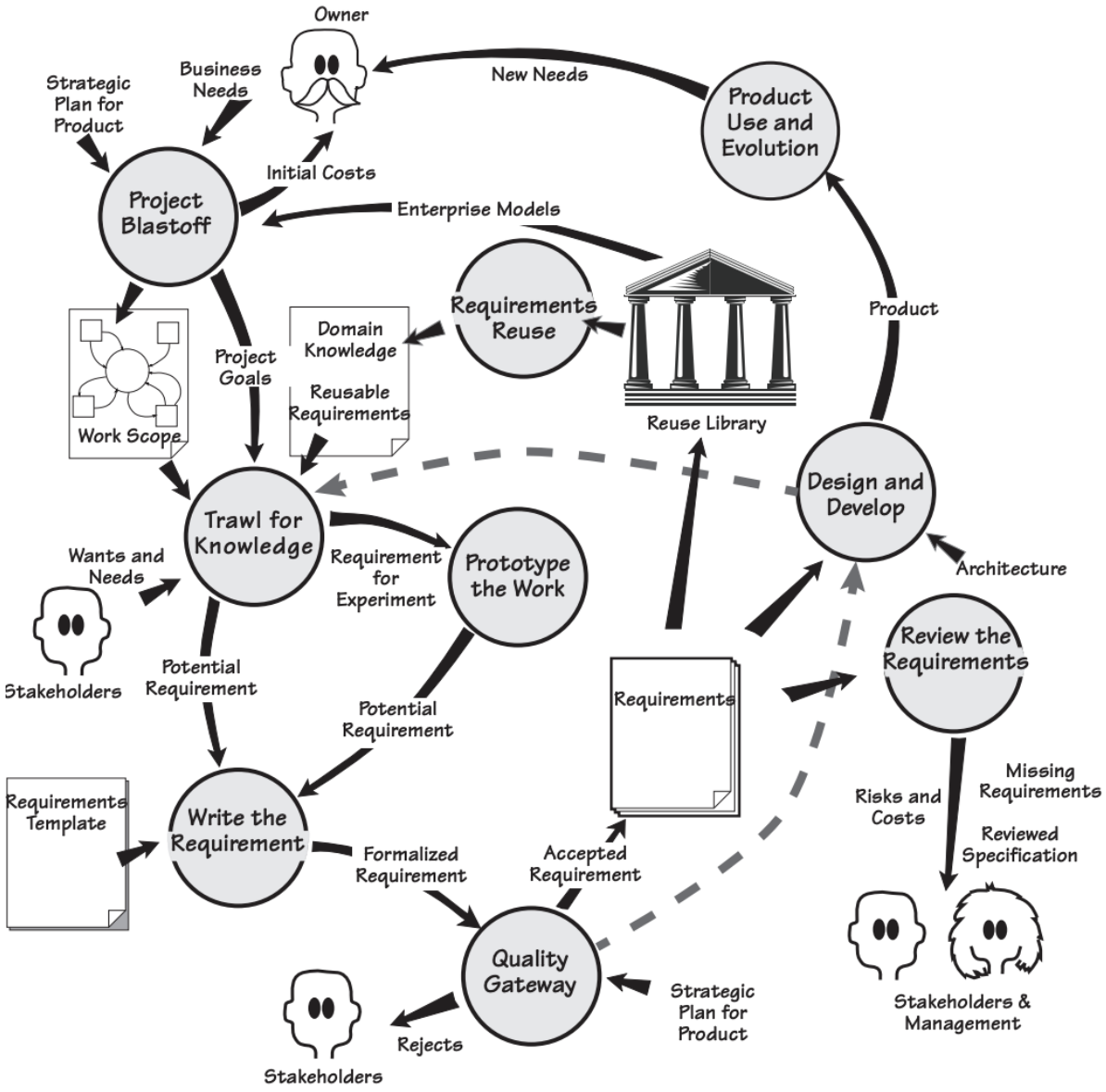
The Volere Requirements Process



El proceso de requerimiento en sí

No es un proceso cuya intención es realizarse en cascada.

Project Blastoff

Sentar las bases para el descubrimiento de requisitos y garantizar todo lo necesario para el éxito. Las partes que participan son las partes interesadas como:

- Sponsor
- Usuarios clave
- Analista de requisitos

- Expertos tecnicos
- Empresarios
- Consultores
- Gerentes: functional beneficiary, political beneficiary, and financial beneficiary.
- Core team: designers, developers, testers, business analysts, systems analysts, systems architects, technical writers, database designers, and anyone else who is involved in the construction.
- Expertos legales
- Inspectores
- Interesados negativos: los que no quieren que el proyecto tenga éxito

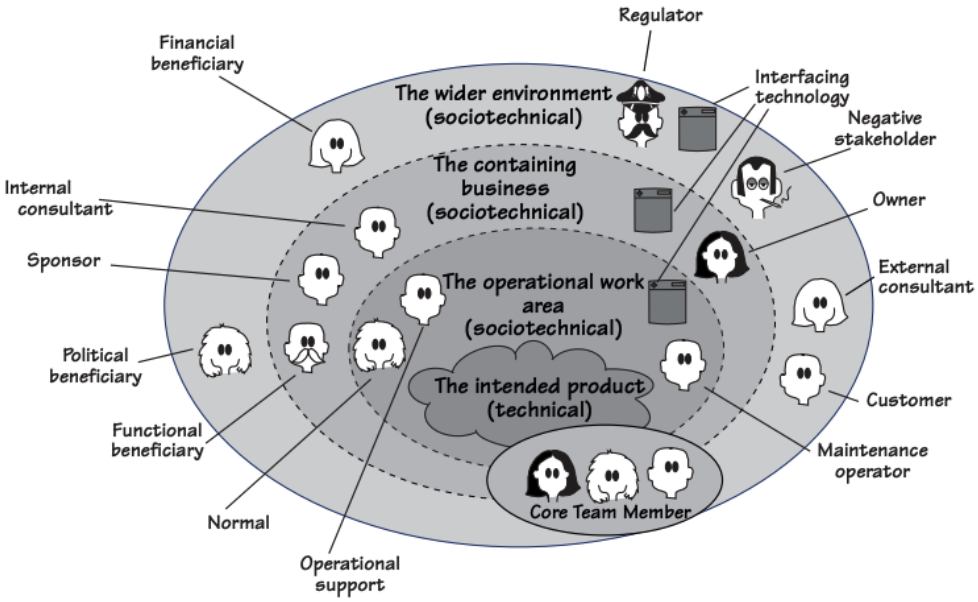
Se define el alcance del problema empresarial realizando un diagrama de contexto el cual muestra las funciones que habrá en el trabajo y que se debe tomar en cuenta.

También se confirma el objetivo del proyecto. Y el grupo llega a un acuerdo sobre la razón de negocio para hacer el proyecto.

Verific a que el proyecto sea viable y valga la pena

Buenas prácticas en esta fase:

- Se deciden los stakeholders
- Estimación de costos
- Propósito del proyecto
- Posibles problemas que pueden surgir y restricciones
- Nombres y terminologías
- Desventajas del proyecto
- Estudia si es viable el producto y si tiene beneficios que compensen los costos y riesgos.
- Si quedan dudas se puede decidir iniciar la investigación de los requisitos.

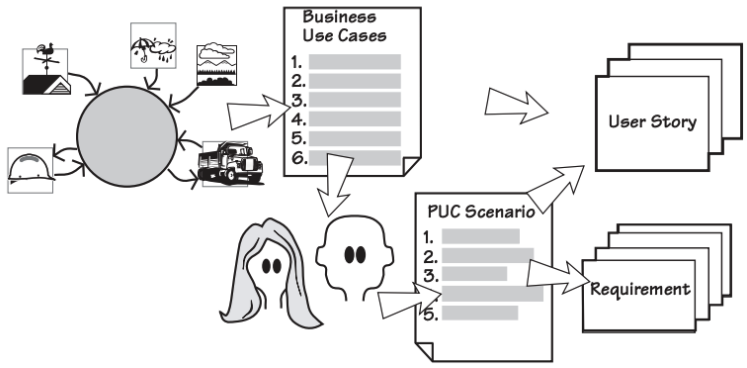


Rastreo de requisitos

Analista de requerimientos es asignado a cada uno de los casos de uso del negocio, pueden trabajar de manera independiente. Utilizan técnicas de trabajo como:

- Escenarios
- Talleres de caso de uso
- Aprendizaje

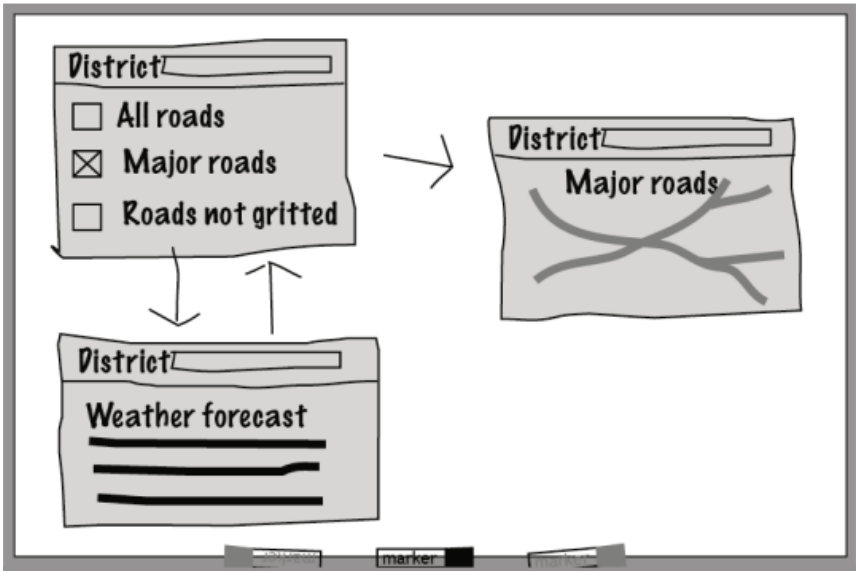
Para ofrecer un producto realmente útil, el equipo analítico debe trabajar con las partes interesadas para innovar, es decir, para desarrollar una forma mejor de hacer el trabajo. Se debe encontrar la esencia del problema.



Quick and dirty modeling

Es el prototipo del trabajo. Existen modelos formales como UML o BPMN pero la mayoría de veces los analistas usan sketches rápidos y pequeños.

Una técnica de modelado rápida y sucia que es el uso de notas Post-it para modelar la funcionalidad, donde cada post it representa una actividad y se pueden reorganizar. Las partes interesadas les genera interés y ganas de participar en esta actividad.

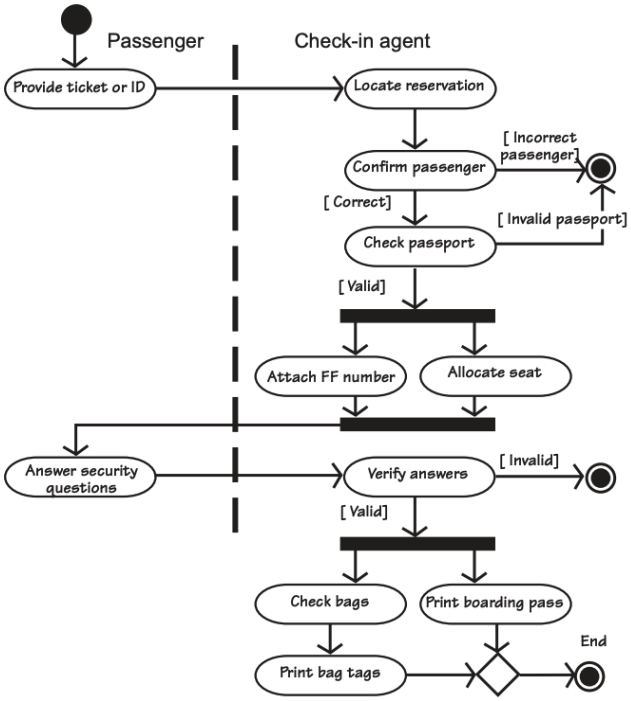


Scenarios

Son muy útiles porque muestran la funcionalidad del negocio transformándolo en pasos reconocibles y accesibles a todas las partes interesadas.

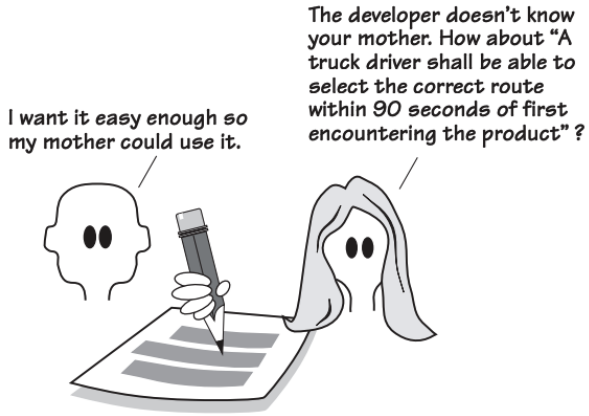
Los analistas de negocio logran que todos entendieran y llegaran a un consenso sobre cuál debía ser el trabajo.

Diagramando un escenario



Escribir los requerimientos

Para evitar malentendidos, los analistas escriben los requerimientos en una manera comprobable y asegura que la parte interesada de origen entiende y está de acuerdo con el requisito escrito antes de transmitir a los desarrolladores. Asegura que la esencia del requerimiento fue capturada y comunicada correctamente



Se le agrega un racional a los requerimientos ya que muestra la razón detrás de ese requerimiento.

Utilizan dos formas para la redacción:

1. Plantilla de especificación de requisitos: Esquema y guía para redactar, es como una lista de comprobación de los requisitos que deben pedir.
2. Tarjeta de nieve o shell: es un diseño práctico para garantizar que cada requisito tenga los atributos necesarios. para asegurarse de que cada requisito tiene los componentes correctos.

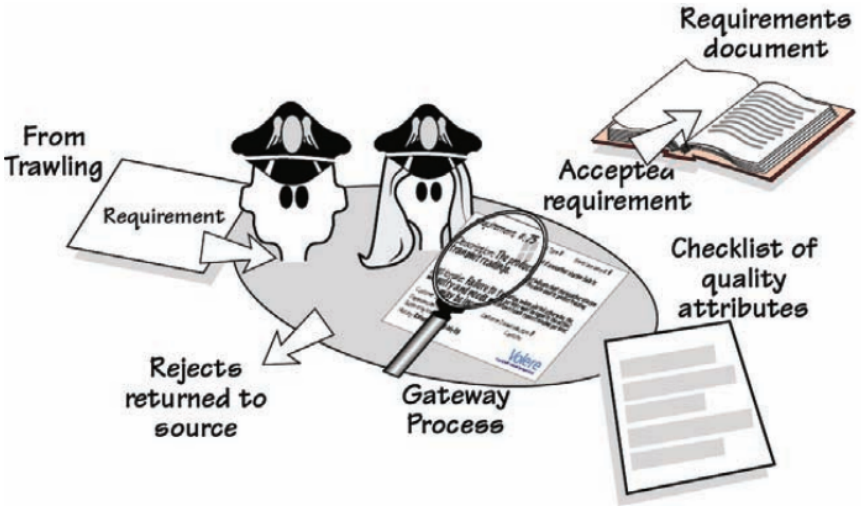
No es una actividad independiente, está rodeado de rastreo, creación de prototipos, pasarela de calidad.

Los métodos de desarrollo iterativo usan las historias de usuario como una forma de transmitir los requerimientos.

Quality gateway

Tiene la razón de que si el producto correcto es construido entonces los requerimientos deben estar bien. Para asegurar eso, los cuality gateway prueba los requerimientos. Establecen un punto que cada requerimiento debe pasar. Esto es llevado a cabo por dos personas: el **leader de analista de requerimientos y el tester** y son los únicos autorizados a pasar los requerimientos a través de la gateway.

Trabajando juntos, comprueban que cada requisito sea completo, pertinente, comprobable, coherente, trazable y otras cualidades antes de permitir que se transmita a los desarrolladores.



Reutilizando requerimientos

Se recomienda tomar requerimientos especificados en otros proyectos previos y buscar potencial material reutilizable. En general, los requisitos no funcionales son muy estándares.

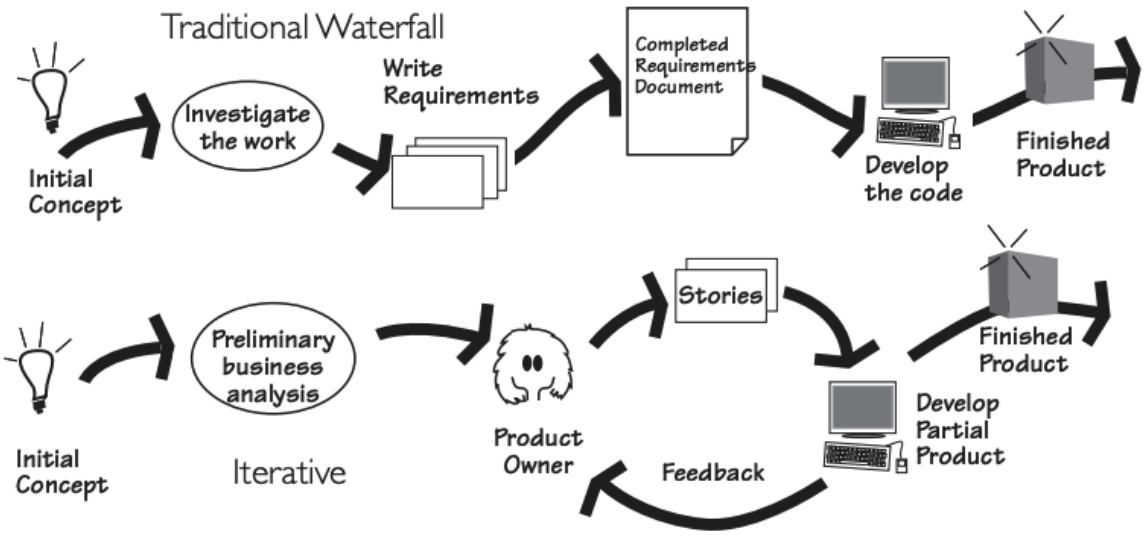
Reviewing the requirements

Esta revisión final comprueba que no faltan requisitos, que todos los requisitos son consistentes entre sí y que cualquier conflicto entre los requisitos han sido resueltos. Ahora se conoce más del producto, y permite analizar los costos y riesgos del proyecto. También a esta altura se conocen los requerimientos asociados a los riesgos más grandes.

Proceso iterativo e incremental

Algunas veces es necesario hacerlo estilo cascada, es decir, se crean todos los requerimientos antes de pasar a la siguiente etapa pero no siempre.

Por un lado, si va a subcontratar o si el documento de requisitos constituye la base de un contrato, es evidente que necesita una especificación de requisitos completa. Por otro lado, si se conoce la arquitectura general, la construcción y la entrega pueden empezar antes de que se conozcan todos los requisitos.



Retrospectiva de requerimientos

Consisten en una serie de entrevistas con las partes interesadas y sesiones de grupo con los desarrolladores. La intención es todas las personas implicadas en el proyecto y plantear estas preguntas:

¿Qué hemos hecho bien?

¿Qué hemos hecho mal?

Si tuviéramos que volver a hacerlo, ¿qué haríamos de forma diferente?

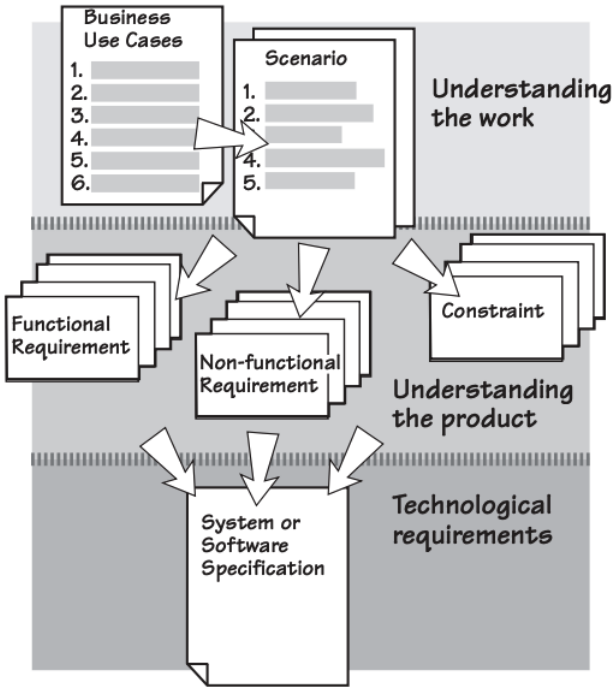
La idea es *hacer más de lo que funciona y menos de lo que no lo hace* . Puede ser informal o más formal.

Evolución de requerimientos

La comprensión del trabajo evoluciona y madura, y en algún momento es posible que las partes interesadas, guiadas por los analistas de negocio y los arquitectos de sistemas, determinen el producto óptimo para mejorar ese trabajo.

Cuando se llega a esta fase, los analistas de negocio determinan la funcionalidad detallada del producto. funcionalidad detallada para el producto (hay que tener en cuenta que no toda la funcionalidad del trabajo se incluiría en el producto) y redactar sus requisitos.

Especifican los requerimientos del producto para apoyar al negocio, se llama “requerimientos de negocio”.



Template

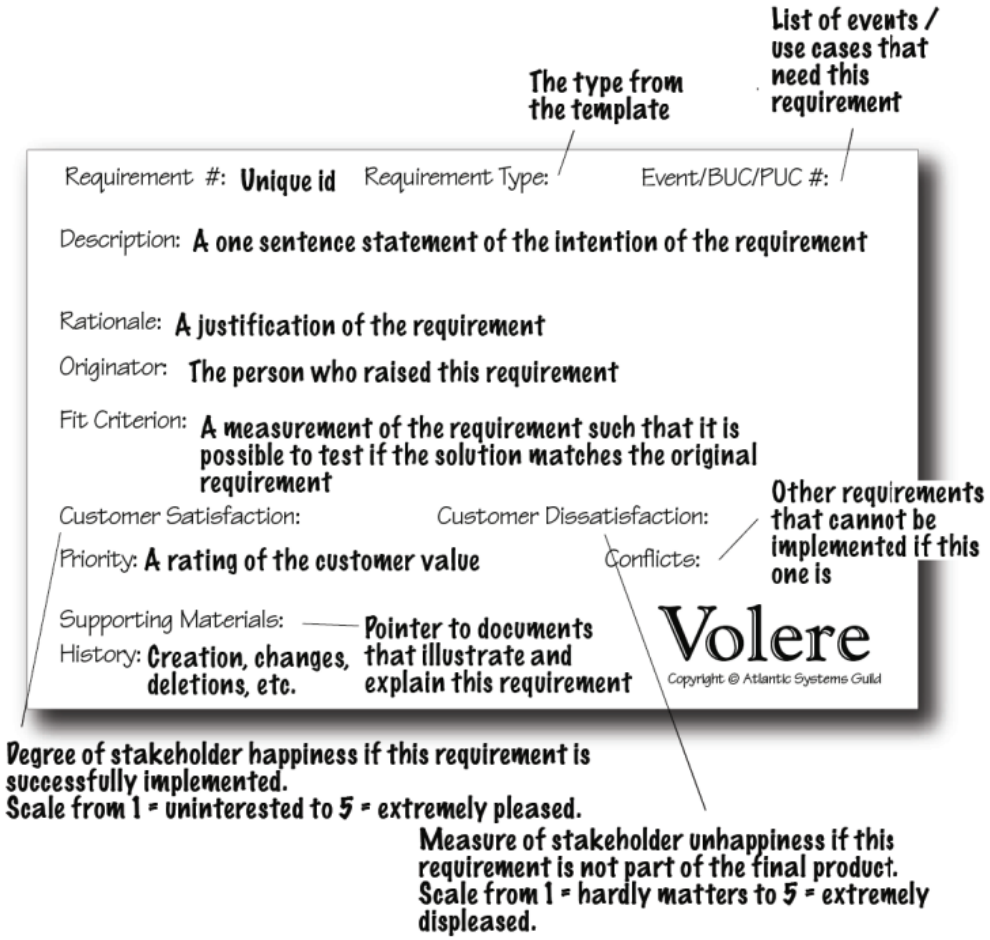
Es una plantilla donde se describen las funcionalidades y capacidades del producto. Sirve como un checklist sofisticado. El contenido de la plantilla es:

- Project drivers: Motivadores y razones del proyecto
    - Propósito, cliente, usuario y otras partes interesadas, usuarios del producto.
- Project constraints: Restricciones del proyecto y del producto
    - Limitaciones y restricciones del diseño del producto, convenciones de nombres y definiciones, datos relevantes y assumptions.
- Requerimientos funcionales: Funcionalidades del producto
    - El alcance del trabajo como el área del negocio o de estudio, el alcance del producto, requerimientos funcionales y datos requeridos.
- Requerimientos no funcionales: Calidad del producto
    - Apariencia esperada, que tiene que ser el producto si es un éxito para la audiencia, requisitos de performance como exactitud, confiable, seguro, escalable, durable. Requisitos operacionales, ambientales, mantenibles, de seguridad, culturales, legales
- Project issues: Problemas relevantes al proyecto que construye el producto

- Cuestiones pendientes, soluciones estándar, tareas, nuevos problemas, riesgos, costos, documentación, requisitos que pueden quedar en la “sala de espera”, ideas para soluciones

The snow card

Es un template sobre cómo escribir los requerimientos.



Guia formal

1. **Rabbit - small, fast and short-lived:**

   Generalmente tienen menos stakeholders. Suelen ser iterativos y descubren requerimientos en unidades pequeñas. No gastan mucho tiempo escribiendo los requerimientos pero tienen conversaciones con stakeholders para escribir las tarjetas de historia, utiliza post-it, sketches. Blast Off meeting corta y pequeña. Deben usar casos de uso para explorar el dominio del problema antes de formular una solución.

Deben comprender el trabajo tal y como es actualmente, así como lo que el trabajo va a ser. Como suelen ser iterativos, es probable que se visite el trabajo actual en pequeñas partes y luego se proceda a encontrar su esencia y, en última instancia, su solución.
Los escenarios en este proyecto suelen pasar por alto los requisitos no funcionales, y los capturan más tarde escribiendo tarjetas de historia no funcionales separadas.

2. **Horse - fast, strong, dependable:**
   Punto medio de la formalidad, tienen longevidad media e implican a una docena de partes interesadas. Factores que requieren documentación coherente. Tienen una blastoff meeting. Deben considerar partir el trabajo según casos de uso.
   Es más probable que se ocupen de sistemas de infraestructuras críticas, por lo que el conocimiento de la obra y sus objetivos es importante para el proyecto.
   Consideran los escenarios como una alternativa para escribir requerimientos funcionales, se puede usar escenarios para descubrir requerimientos pero no deben ser usados como especificación final.
3. **Elephant - solid, strong, long life and a long memory:**
   Especificación de requisitos completa, muchos stakeholders aparecen en este tipo de proyecto. Hay un gran número de desarrolladores, lo que requiere formas más formales de comunicación. Tienen mucho para perder de no hacer blast off meeting. Deben dar el paso adicional de hacer que LA pruebe sus resultados: Los proyectos de los elefantes son críticos, y costosos si cometen errores.
   Utilizan escenarios como una herramienta de descubrimiento.

FALTA CAPITULO 9 - Strategies for today's business analyst

Requisitos en ingeniería de software y sistemas

En ingeniería de sistemas existen tres tipos de requisitos.

- Un requisito funcional puede ser una descripción de lo que un sistema debe hacer. Este tipo de requisito especifica algo que el sistema entregado debe ser capaz de realizar.
- Un requisito no funcional: de rendimiento, de calidad, etc; especifica algo sobre el propio sistema, y cómo debe realizar sus funciones. Algunos ejemplos de aspectos solicitables son la disponibilidad, el testeo, el mantenimiento, la facilidad de uso, rendimiento, disponibilidad, documentación, etc.

Se suelen clasificar en:

- Requisitos de calidad de ejecución, que incluyen seguridad, usabilidad y otros medibles en tiempo de ejecución.
- Requisitos de calidad de evolución, como testabilidad, extensibilidad o escalabilidad, que se evalúan en los elementos estáticos del sistema software.[2]

- Otros tipos de limitaciones externas, que afectan en forma indirecta al producto. Estas pueden ir desde la compatibilidad con cierto sistema operativo hasta la adecuación a leyes o regulaciones aplicables al producto

Una colección de requisitos describe las características o atributos del sistema deseado. Se omite el cómo debe lograrse su implementación, ya que esto debe ser decidido en la etapa de diseño por los diseñadores.

En la ingeniería de software se aplica el mismo significado, sólo que el énfasis está puesto en el propio software.

Caracteristicas

Los requisitos bien formulados deben satisfacer varias características. Si no lo hacen, deben ser reformulados hasta hacerlo.

- **No ambiguo:** El texto debe ser claro, preciso y tener una única interpretación posible.
- **Conciso:** Debe redactarse en un lenguaje comprensible por los inversores en lugar de uno de tipo técnico y especializado, aunque aun así debe referenciar los aspectos importantes.
- **Consistente:** Ningún requisito debe entrar en conflicto con otro requisito diferente, ni con parte de otro. Asimismo, el lenguaje empleado entre los distintos requisitos debe ser consistente también.
- **Completo:** Los requisitos deben contener en sí mismos toda la información necesaria, y no remitir a otras fuentes externas que los expliquen con más detalle.
- **Alcanzable:** Un requisito debe ser un objetivo realista, posible de ser alcanzado con el dinero, el tiempo y los recursos disponibles.
- **Verificable:** Se debe poder verificar con absoluta certeza, si el requisito fue satisfecho o no. Esta verificación puede lograrse mediante inspección, análisis, demostración o testeo.

INGENIERIA DE SOFTWARE

La ingeniería de *software* se puede considerar como la ingeniería aplicada al *software*, esto es, por medios sistematizados y con herramientas preestablecidas, la aplicación de ellos de la manera más eficiente para la obtención de resultados óptimos

Recursos:

- Humanos: Personas que intervienen en la planificación de cualquier instancia. Puede ser determinado después de la estimación
- De entorno: El hardware proporciona el medio físico para desarrollar el software.

Notaciones:

- UML: Lenguaje de modelado, vistas elementos de un modelo, conjunto de reglas sintácticas y semánticas.
- BPMN: Business Process Model and Notation. Proporcionar de una manera fácil de definir y analizar los procesos de negocios públicos y privados simulando un diagrama de flujo.
- DFD: Diagramas de Flujo de Datos. Representar el movimiento de datos a través de un sistema por medio de modelos que describen los flujos de datos, los procesos que transforman o cambian los datos, los destinos de datos y los almacenamientos de datos a la cual tiene acceso el sistema.

Herramienta CASE

Son herramientas computacionales (*software*) que están destinadas a asistir en los procesos de ciclo de vida de un *software*, facilitan la producción del *software*, varias se basan principalmente en la idea de un modelo gráfico.

Etapas del proceso = Ciclo de vida

1. Obtención de los requisitos: Identificar tema principal que motiva el proyecto y recursos. Dominio de la información del problema → incluye datos fuera de software como usuarios, dispositivos externos, datos que salen del sistema y almacenamiento. Ver los puntos críticos, como será el comportamiento del software ante situaciones inesperadas.
2. Análisis de requisitos: reconocer requisitos incompletos, ambiguos o contradictorios, tomando en cuenta las necesidades del usuario final. El resultado del análisis de requisitos con el cliente se plasma en el documento ERS (Especificación de requisitos del sistema) con una estructura definida como CMMI.

3. Limitaciones:Una posible limitación puede ser que el software solo posee ciertas funciones predefinidas o también proviene del proceso totalmente mecánico que requiere de un mayor esfuerzo y tiempos elevados de ejecución.
4. Especificación: Describe el comportamiento esperado en el software una vez desarrollado. Parte del éxito de un proyecto de *software* radica en la identificación de las necesidades del negocio. Técnicas para la especificación de requisitos:
    - Casos de uso
    - Historias de usuario
5. Arquitectura: El diseño arquitectónico debe permitir visualizar la interacción entre las entidades del negocio y además poder ser validado. (Diagramas de clase, de base de datos, de despliegue, de secuencia). Los diagramas de clases y de base de datos son los mínimos necesarios para describir la arquitectura de un proyecto que iniciará a ser codificado. Dependiendo del alcance del proyecto, complejidad y necesidades, el arquitecto elegirá cuáles de los diagramas se requiere elaborar. → herramientas para el diseño y modelado se denominan CASE (Computer Aided Software Engineering)
6. Programación: La complejidad y la duración de esta etapa está íntimamente relacionada al o a los lenguajes de programación utilizados, así como al diseño previamente realizado.
7. Desarrollo de la aplicación: Se consideran 5 fases de desarrollo→
    - Infraestructura
    - Adaptación del paquete: Entender de una manera detallada el funcionamiento del paquete, esto tiene como finalidad garantizar que el paquete pueda ser utilizado en su máximo rendimiento, tanto para negocios o recursos.
    - Unidades de diseño: Se realizan los procedimientos que se ejecutan por un diálogo usuario-sistema.
        - Establecer específicamente las acciones que debe efectuar la unidad de diseño.
        - La creación de componentes para sus procedimientos.
        - Ejecutar pruebas unitarias y de integración en la unidad de diseño
    - Unidades de diseño batch: Plasmar de manera clara y objetiva las especificaciones para el programador, como diagramas de flujo, diagramas de estructuras, tablas de decisiones, etc.

    - Unidades de diseno manuales: Procedimientos administrativos que desarrollarán en torno a la utilización de los componentes computarizados

8. Pruebas de software: Una técnica es probar por separado cada módulo del *software* (prueba unitaria), y luego probarlo de manera integral (pruebas de integración), para así llegar al objetivo proporcionar realimentación a los desarrolladores.
9. Implementación: Proceso de convertir una especificación del sistema en un sistema ejecutable. El modelo es una colección de componentes y los subsistemas que contienen. Componentes tales como: ficheros ejecutables, ficheros de código fuente y todo otro tipo de ficheros que sean necesarios para la implementación y despliegue del sistema.
10. Documentación: Con el propósito de eventuales correcciones, usabilidad, mantenimiento futuro y ampliaciones al sistema.
11. Mantenimiento: Fase dedicada a mantener y mejorar el *software* para corregir errores descubiertos e incorporar nuevos requisitos. Constituye alrededor de 2⁄3 del ciclo de vida de un proyecto.

Ventajas desde distintos puntos de vista

- Gestión: Facilitar tarea de seguimiento y optimizar recursos. Facilitar comunicación y evaluación de resultados.
- Ingenieros de software: Comprender el problema, permitir reutilización, facilitar mantenimiento, optimizar el conjunto y cada una de las fases del proceso de desarrollo.
- Cliente o usuario final: Garantizar nivel de calidad, confianza en los plazos del tiempo.

Procesos:

Definición de objetos → Análisis de los requisitos → Diseño general → Diseño en detalle → Programación → Prueba de unidad → Integración → Prueba beta (validación) → Documentación → implementación → Mantenimiento

Modelos

- Cascada o clásico: Aproximación secuencial. Se inicia con la especificación de requisitos del cliente y continúa con la planificación, el modelado, la construcción y el despliegue para culminar en el soporte del *software* terminado.
- Prototipos: Se utiliza en general cuando el responsable no está seguro de la eficacia de un algoritmo, de la adaptabilidad del sistema o de la manera en que interactúa el hombre y la máquina.

- Espiral: El *software* se desarrolla en una serie de entregas evolutivas (ciclos o iteraciones), cada una de estas entregando prototipos más completos que el anterior. Uso en el ámbito es escaso → El cálculo de riesgos puede ser complicado.
- Desarrollo por etapas: *software* se muestra al cliente en etapas refinadas sucesivamente. Fases: Especificación conceptual → análisis de requisitos → diseño inicial → diseño detallado.
  Ventajas: Detecta problemas, elimina tiempos de informes, puede cada versión es un avance, cumplimiento a la fecha.
- Incremental / iterativo: aplica secuencias lineales como el modelo en cascada, pero de una manera iterativa o escalada según cómo avance el proceso de desarrollo y con cada una de estas secuencias lineales se producen incrementos (mejoras) del *software*. busca la entrega de un producto operacional con cada incremento que se le realice al *software*.
- RAD (Rapid Application Dev): Es un modelo de proceso de *software* incremental. Construcción de prototipos centrados en el usuario y el seguimiento lineal y sistemático de objetivos, incrementando la rapidez con la que se producen los sistemas mediante la utilización de un enfoque de desarrollo basado en componentes
- Modelo orientado a objetos: Raices en la POO. Permite reutilizacion de software, facilita el desarrollo de herramientas informaticas de ayuda al desarollo.
- Modelo estructurado: facilita la comprensión de la estructura de datos y su control. Diferencia de una manera más clara los procesos, existen métodos que se enfocan en ciertos datos, abstracción del programa es de un nivel mayor.
  Desventajas: Datos repetidos, código extenso o grande, el manejo se complica.
  Técnicas generalmente utilizadas: MER, diagrama de flujo.
- Desarrollo concurrente: todas las personas actúan simultáneamente o al mismo tiempo.
- Unificado: asegurar la producción de *software* de muy alta calidad que satisfaga las necesidades de los usuarios finales, dentro de un calendario y presupuesto predecible. Tiene dos dimensiones:
  1. Eje horizontal: representa el tiempo y muestra los aspectos del ciclo de vida del proceso a lo largo de su desarrollo. Aspecto dinámico del proceso.
  2. Eje vertical: representa las disciplinas, las cuales agrupan actividades de una manera lógica de acuerdo con su naturaleza. Aspecto estático del proceso

El refinamiento más conocido y documentado del proceso unificado es el RUP (proceso unificado racional). Es un marco de trabajo extensible que puede ser adaptado a organizaciones o proyectos específicos.

HISTORIAS DE USUARIO

Una historia de usuario es una explicación general e informal de una función de software escrita desde la perspectiva del usuario final. Su objetivo es explicar cómo una función del software aportará valor al cliente. Son lenguaje non-tecnical y provee contexto para el desarrollo del equipo y sus esfuerzos.

- Mantienen el foco en el usuario
- Permite colaboración ya que el objetivo está definido.
- Llevan a soluciones creativas
- Crean momentos

Escribir historias de usuario:

- La historia esta “done” una vez que el usuario completa la tarea
- Tiene subtasks o tasks: Son como pasos que deben completar
- Para quien es la historia de usuario: Considerar crear múltiples historias
- Escuchar feedback
- Tiempo

Ejemplo: “As a [persona], I [want to], [so that].”

CASO DE USO

Descripción de las actividades que deberá realizar alguien para llevar a cabo un proceso. Se denominan actores.

Conjunto de interacciones que se desarrollarán entre casos de uso y entre estos y sus actores en respuesta a un evento que inicia un actor principal.

Su uso es común para la captura de requisitos funcionales, especialmente con el paradigma de la programación orientada a objetos.

Un caso de uso debe:

- Describir tarea del negocio
- Nivel apropiado del detalle
- Ser sencillo

Tipos de relaciones

- Comunica (<<*communicates*>>): Relación (asociación) entre un actor y un caso de uso que denota la participación del actor en dicho caso de uso.
- Usa (<<*uses*>>) (o <<*include*>> en la nueva versión de UML): Relación de dependencia entre dos casos de uso que denota la inclusión del comportamiento de un escenario en otro.
- Generalización (sin estereotipo) (En UML 1.3) Indica que un caso de uso es una variante de otro. El caso de uso especializado puede variar cualquier aspecto del caso de uso base.
- Extiende (<<*extend*>>, <<*extiende*>>)(En UML 1.3): Es un estereotipo de dependencia. Ofrece una forma de extensión más controlada que la relación de generalización. El caso de uso base declara un conjunto de puntos de extensión, El caso de uso especializado solo puede alterar el comportamiento de los puntos de extensión marcados.

Pasos para definición de un caso de uso:

- ID
- NOMBRE

- REFERENCIAS CRUZADAS
- CREADO POR
- ÚLTIMA ACTUALIZACIÓN POR
- FECHA DE CREACIÓN
- FECHA DE ÚLTIMA ACTUALIZACIÓN
- ACTORES
- DESCRIPCIÓN
- TRIGGER
- PRE-CONDICIÓN
- POST-CONDICIÓN
- FLUJO NORMAL
- FLUJOS ALTERNATIVOS
- INCLUDES
- FRECUENCIA DE USO
- REGLAS DE NEGOCIO
- REQUISITOS ESPECIALES
- NOTAS Y ASUNTO

Situaciones:

- Actor comunica con un caso de uso
- Caso de uso extiende otro caso de uso
- Caso de uso utiliza otro caso de uso

Limitaciones:

No establecen completamente los requisitos funcionales ni permiten determinar los requisitos no funcionales. La ingeniería del funcionamiento especifica que cada caso crítico del uso debe tener un requisito no funcional centrado en el funcionamiento asociado.