Técnicas de análisis, modelado y especificaciones de requisitos

Tabla de contenido

Introducción

Recordemos que el desarrollo de requisitos incluye actividades relacionadas con el descubrimiento, el análisis, la especificación, la verificación y la validación de requisitos:

- El análisis se enfoca en identificar y resolver conflictos entre los requisitos, en determinar cuáles son los límites del sistema y cómo es la interacción entre sistema y su medio ambiente. Como consecuencia del análisis es usual que aparezcan nuevos requisitos: requisitos derivados. El análisis implicará producir modelos, modelos *abstractos* del sistema, software o solución que vamos a desarrollar. Decimos que son abstractos o conceptuales porque estos modelos no incluyen aspectos de diseño.
- La especificación, por otro lado, consiste en producir uno o varios documentos de formalidad variable con diferente nivel de detalle con la descripción del sistema, software o solución a desarrollar. Hay que tener siempre presente que en condiciones normales, la especificación y el análisis se realizan en simultáneo.

Identifique de las siguientes sentencias cuál o cuáles son verdaderas

- ☐ A través del análisis y la especificación se definen las necesidades de los interesados
- ☑ **En análisis de requisitos los modelos se consideran conceptuales o abstractos porque no incluyen aspectos de diseño**
- ☑ **Especificación y análisis se realizan en simultáneo**
- ☐ El análisis de requisitos solamente se enfoca en identificar y resolver conflictos entre requisitos
- ☑ **La especificación consiste en producir uno o varios documentos de formalidad variable con diferente nivel de detalle**

Requisitos

Distintos interesados, distintos puntos de vista

También es importante que tengamos presente que la ingeniería de requisitos es un proceso que, partiendo de las necesidades y expectativas de los usuarios, progresivamente va elaborando una definición de los requisitos que deberá satisfacer el software que debemos desarrollar. Estos requisitos los podremos describir desde distintos puntos de vista:

- Uno de los puntos de vista es el de la organización que necesita el software. Estos tipos de requisitos son normalmente conocidos como **requisitos o requerimientos de negocio**.
- Otro punto de vista es el de los usuarios que van a utilizar el software. Normalmente conocidos como **requisitos de usuario**.
- Y finalmente nos encontraremos con el punto de vista de los desarrolladores que deberán implementar y mantener el software. A estos requisitos se los conoce como **requisitos de software**.
- En algunos casos, puede ser que haya una capa superior. En ese caso, serán los **requisitos del sistema**.

Para cada uno de estos puntos de vista usualmente se producirán distintos documentos llamados genéricamente *especificaciones de requisitos*. Los nombres de estos documentos variarán dependiendo de la audiencia y del alcance, pero su propósito colectivo es invariable: acordar entre las partes cuáles son los requisitos del software a desarrollar.

Por ejemplo, para registrar los requisitos de la organización que nos ha encargado el producto, utilizaremos artefactos tales como el documento de visión, el ConOps (Concepto de Operación), o la especificación de requisitos de negocio (BRD).

Para describir los requisitos desde el punto de vista del usuario (URD), usualmente utilizaremos una estrategia de caja negra. Es decir, vamos a describir únicamente la interacción de los actores de los usuarios con el sistema, no los detalles internos. Para esto hay múltiples alternativas, desde emplear el lenguaje natural hasta técnicas más elaboradas, como casos de uso o historias de usuarios.

Finalmente, para el punto de vista del equipo de desarrollo aparece la especificación de requisitos de software (SRS). Esta sí es una visión interna de comportamiento que deberá tener el software a desarrollar. No es un documento de diseño, sino una descripción abstracta o conceptual de lo que el software deberá hacer, independientemente de cómo lo implementemos.

[**V** / F] El análisis de requisitos se enfoca en identificar y resolver conflictos entre los requisitos.

Alternativas de representación

Los requisitos se pueden representar de varias formas:

- El **lenguaje natural** es una opción, pero debe tenerse cuidado con la ambigüedad. Estos requisitos normalmente van a ser enunciados con los formatos: “el usuario debe…” o “el sistema debe…”.
- También se pueden utilizar **modelos visuales** para ilustrar procesos, para describir estados del sistema y cambios entre ellos, para describir relaciones de datos, grupos lógicos de trabajo, etc.
- Una tercera opción, menos frecuente en aplicaciones convencionales, son las **especificaciones formales**. En este caso, los requisitos se describen mediante el uso de lenguajes matemáticos muy precisos. Normalmente la mayoría de los casos utilizaremos una combinación de las primeras 2 opciones.

Visión (visión y alcance)

Revisemos ahora en profundidad algunos de estos artefactos que acabamos de mencionar. Un documento muy habitual es el *documento de visión*, llamado en ciertos ámbitos también, de visión y alcance. Normalmente se produce en la primera fase o en la primera iteración de los proyectos de desarrollo. Define el sistema que tenemos que construir desde el punto de vista de los interesados de negocio. Es la base para las especificaciones más detalladas que se desarrollarán posteriormente y suele servir muchas veces como base para la elaboración de un contrato. El contenido puede variar, pero usualmente incluye una definición del problema a resolver o de la oportunidad de negocio a aprovechar, una descripción de los usuarios o interesados, una descripción de las prestaciones del producto (a muy alto nivel), riesgos, valor para el negocio, supuestos, restricciones, etc.

SRS (Especificación de Requisitos de Software)

La especificación de requisitos de software, por otro lado, es un documento o grupo de documentos electrónicos, usualmente escrito en lenguaje natural y a veces complementado con gráficos y descripciones semi formales, como por ejemplo caso de uso o reglas de negocio. Se utiliza en los más diversos ámbitos, aunque históricamente está más ligado a ciclos de desarrollo más clásicos, más secuenciales, o en algunos entornos en donde hay contratos de por medio. Para evitar malos entendidos, la prosa debe ser clara, tiene que estar bien estructurado y los términos poco frecuentes deberían estar incluidos en un glosario. Hay varios estándares y varias recomendaciones acerca de cómo estructurar y escribir este tipo de documentos. Hay uno muy conocido que se llama EARS, del cual vamos a hablar un poco más adelante.

Desarrollo ágil

En entornos ágiles es más difícil, sino improbable, encontrar especificaciones como las que acabamos de ver. En general, nos vamos a encontrar con historias de usuario que suelen estar complementadas con criterios de aceptación. Desde el punto de vista estricto para el mundo ágil, las historias de usuario no son requisitos en el sentido clásico, sino que son recordatorios de una conversación que hay que tener con el usuario o con el experto en el dominio. Recordemos que en los desarrollos ágiles, el equipo no solamente está conformado por desarrolladores, sino que tenemos especialistas en el negocio (especialistas o usuarios capacitados) trabajando junto a nosotros.

[**V** / F] En los desarrollos ágiles hay especialistas del negocio o usuarios capacitados que trabajan a la par del equipo, por ende las historias de usuario no son consideradas requisitos puros sino que son un recordatorio de una conversación con dicho especialistas o usuarios.

Atributos de una especificación bien escrita

Más allá del formato que utilicemos, una especificación bien escrita debe reunir las siguientes características:

- **Correcta**: Debe reflejar las verdaderas necesidades de los usuarios. No debe tener ambigüedades, es decir, que debe haber una única interpretación posible para cada uno de los requisitos.
- **Completa**: Esto ocurre cuando se cumplen las siguientes cualidades:
  - Todo lo que se supone que el software debe hacer está incluido en la especificación.
  - Todas las respuestas a todas las entradas al sistema en todas las situaciones posibles están incluidas en la especificación.
  - Todas las páginas del documento, todas las figuras, todas las tablas están numeradas con nombre y referenciadas correctamente.
  - No hay secciones vacías o que se vayan a completar posteriormente.
- **Verificable**: Es decir, que hay criterios para determinar si el software, una vez desarrollado, satisface o no los requisitos.
- **Consistente**: Es decir, no hay requisitos en la especificación que estén en conflicto entre sí, o que estén en conflicto con otros documentos, con otros artefactos.
- **Entendible**: La especificación puede ser entendida por terceros. Los requisitos se van a escribir normalmente una sola vez, pero van a ser leído muchas veces, por eso es muy importante tener una prosa clara y bien entendida.
- **Modificable**: La estructura, el estilo de la especificación, tienen que permitir que se pueda hacer una modificación de manera relativamente fácil ante un cambio en alguno de los requisitos.
- **Independiente del diseño**: Recordemos que la especificación describe el comportamiento del sistema independientemente de las decisiones que tomemos para implementarlo. Con lo cual, no implica ningún tipo de arquitectura en particular, ningún tipo de interfaz de usuario en particular, de ningún tipo de algoritmo o pieza de software.
- **Concisa**: No debe ser innecesariamente extensa.
- **Organizada**: Los requisitos tienen que ser fáciles de localizar en el documento.
- **Trazable**: Se puede identificar el origen de cada requisito.
- **Priorizada**: Los requisitos deben tener una prioridad.

Indique cuál o cuáles de las siguientes oraciones son correctas:

- ☑ **Una característica para considerar una especificación completa es cuando no falta completar secciones**
- ☐ Que una especificación posea estructura facilita que sea entendible
- ☑ **Los requisitos se escriben una vez pero se leen muchas veces**

Análisis, modelado, especificación

Analizar, muchas veces, va a implicar construir modelos del sistema que tenemos que desarrollar. Estos modelos reflejan las abstracciones del dominio del problema y decimos que son abstractos o conceptuales porque describen el sistema en términos tecnológicamente neutros. Estos modelos no describen el diseño del producto. Y aquí hay 2 enfoques que se pueden adoptar: Uno más formal, como por ejemplo ocurre en el ámbito del Model Driven Design; o más informal, como pasa en el desarrollo ágil.

Cómo analizar, modelar y especificar requisitos

Técnicas

A continuación, vamos a presentar las principales técnicas que se pueden utilizar para analizar, modelar y especificar:

Organización

Una técnica fundamental en la **organización** de requisitos. Esta técnica nos permite establecer categorías para identificar y ubicar más rápidamente los requisitos. Hay varias clasificaciones posibles:

- Binaria: funcional, no funcional.
- FURPS+: Clasifica a los requisitos en términos de funcionabilidad, usabilidad, confiabilidad, desempeño, facilidad de soporte, otros.
- Por rol del usuario.
- Por área o proceso de negocio.
- Por evento, caso de uso o escenario.

Hay algunos aspectos que hay que considerar al organizar los requisitos. Por un lado debería haber un repositorio. Cualquier desarrollo no trivial implicará una cantidad significativa de requisitos. Es importante entonces establecer algún tipo de repositorio mediante el cual se puedan compartir entre todos los actores relevantes los requisitos. En las metodologías ágiles es muy usual, la utilización de notas adhesivas que se pegan en algún tipo de tablero físico en el lugar de trabajo. Eso, por supuesto que es totalmente viable, pero en algún momento va a ser importante que haya también una versión electrónica de ese tablero y de esas notas.

Otro tema importante es la **gestión de cambios**. Ya hemos mencionado que los requisitos inevitablemente van a cambiar a lo largo del proyecto. Hay que definir de qué manera vamos a reflejar esos cambios en el repositorio y en los demás mecanismos que vamos a utilizar para registrar los requisitos. Uno de los problemas más graves que podemos llegar a enfrentar es no saber cuál es la línea base oficial de los requisitos que el software tiene que satisfacer.

También es muy importante mantener cierto nivel de **trazabilidad**. Es importante entender la relación que tienen cada uno de los requisitos con el resto de los artefactos del ciclo de vida. Por ejemplo, un requisito, le pueden corresponder una serie de criterios de aceptación o de casos de prueba. También es importante saber cuál es el origen de cada requisito, por ejemplo, siempre viene de una conversación que se tuvo con el usuario de una reunión, una sesión de trabajo. Es muy importante saber quién dijo qué, en cuál reunión, en qué contexto. Esto es sumamente importante porque nos permite hacer análisis de impacto: Ante un cambio, qué otros artefactos vamos a tener que revisar y ajustar.

También necesitaremos registrar algunos **atributos** para cada requisito. Cada requisito debería tener, por ejemplo, una identificación única (aunque esto en las metodologías ágiles es un poco resistido), debería tener una prioridad, puede tener riesgos asociados, puede tener una complejidad. Puede también tener una definición de cuál es el esfuerzo que implicaría implementar ese requisito en el software.

Estos atributos, estos aspectos, es importante que los definamos, independientemente del tipo de herramientas que utilicemos. Como siempre, es mucho más fácil organizar las cosas de entrada y no tener que estar después sufriendo por la falta de estructura en la información.

Identifique de las siguientes sentencias cuál o cuáles son verdaderas respecto a la organización de requisitos:

- ☑ **La organización de requisitos permite establecer categorías para identificar y ubicar fácilmente a los requisitos**
- ☐ Al organizar los requisitos contar con una buena trazabilidad de los mismos suele ser opcional
- ☑ **Gestionar de los cambios en los requisitos ayuda a conocer la línea base de los requisitos**
- ☐ Es poco importante contar con un repositorio de requisitos
- ☑ **Es necesario registrar atributos asociados a los requisitos como su identificación, prioridad, riesgos, complejidad o esfuerzo estimado**

Priorización

La priorización nos permite asegurar que el esfuerzo del equipo esté dirigido hacia el análisis y la implementación de los requisitos más críticos para todos los interesados. Son varios los **criterios** que se pueden utilizar para establecer la prioridad:

- El valor para el negocio, el valor que genera para el negocio.
- Los riesgos técnicos o los riesgos de negocio.
- La dificultad que implica la implementación.
- Regulaciones o las políticas organizacionales.

Hay varias **técnicas** que se pueden utilizar:

- MoSCoW: Must / Should / Could / Won't. Se clasifican los requisitos en categorías: Obligatorios, probablemente necesarios, deseados, innecesarios o postergables. Otra alternativa posible, necesario deseados, o nice to have.
- Votación.
- Time boxing: Se decide la prioridad de los requisitos en función del tiempo de desarrollo disponible.
- Por el resultado de un análisis de riesgo.
- Por el resultado de un análisis de decisiones, a través de mecanismos más específicos de toma de decisiones.

[**V** / F] Para garantizar que el esfuerzo del equipo esté dirigido hacia el análisis y la implementación de los requisitos más críticos para todos los interesados es necesaria la priorización de los requisitos.

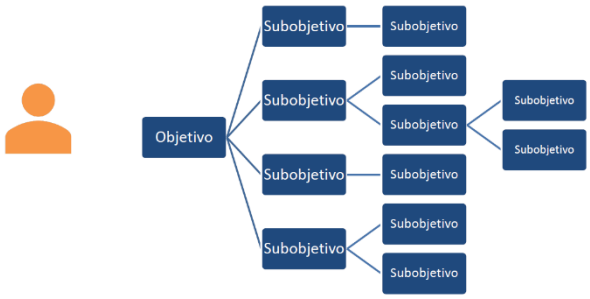
Escenarios

Una familia de técnicas

Vamos a hablar a continuación de la especificación de requisitos mediante escenarios. ¿Qué son los escenarios? Bueno, con este nombre se conoce a una familia de técnicas que están orientadas a contar historias acerca de qué es lo que hace la gente con los sistemas. ¿Cómo es que los utiliza?, o ¿cómo es que piensa que los van a utilizar? Nuestro cerebro social está especialmente adaptado para entender historias, es la forma en la que el hombre primitivo transmitía conocimiento de una generación a la otra, con lo cual este grupo de técnicas nos resulta muy familiar, resulta muy eficiente. Hay varias técnicas incluidas en esta familia, las más conocidas y de las que vamos a hablar a continuación son: casos de uso, historias de usuario y storyboards o guiones.

Usuarios y objetivos

Todas estas técnicas comparten una característica esencial: Para identificar a los escenarios, proponen partir de los objetivos que las distintas categorías de usuarios tienen con respecto a la utilización del sistema. A partir de estos objetivos, se van identificando objetivos de menor nivel. Los objetivos de menor nivel se satisfarán mediante un escenario, un caso de uso, una historia de usuario, un storyboard.



Casos de uso

Un **caso de uso** es una secuencia de acciones realizadas por un sistema que generan un resultado observable y de valor para un actor en particular. Un actor en este contexto es un tipo de usuario que va a utilizar el sistema. A cada acción de un actor, dentro de un caso de uso, le corresponde una respuesta del sistema. Los casos de uso en general son cajas negras, es decir, describen qué es lo que responde el sistema, pero no cómo el sistema elabora esa respuesta.

Historias de usuario

Otra técnica muy popular, es la de **historias de usuarios**. Una historia de usuario es una descripción de la funcionalidad esperada de un sistema desde el punto de vista de un usuario. En sentido estricto según algunos autores, las historias de usuario no son requisitos, son simplemente un recordatorio de que se debe mantener una conversación con el usuario para profundizar los detalles de las funcionalidad asociada a la historia. Son más bien

requisitos de usuario con poca granularidad, si se quiere. Sin embargo, cuando las historias de usuario van acompañadas de criterios de aceptación, constituyen una descripción bastante completa de lo que se espera del comportamiento del sistema.

Guiones (Storyboards)

Y por último, los **storyboards**: Los storyboards o guiones son pequeños relatos en prosa o descritos en forma gráfica, que nos cuentan cómo usar el sistema de determinadas circunstancias.

[V / **F**] Los casos de uso y las historias de usuario son herramientas complementarias.

Modelado

Lenguajes y notaciones

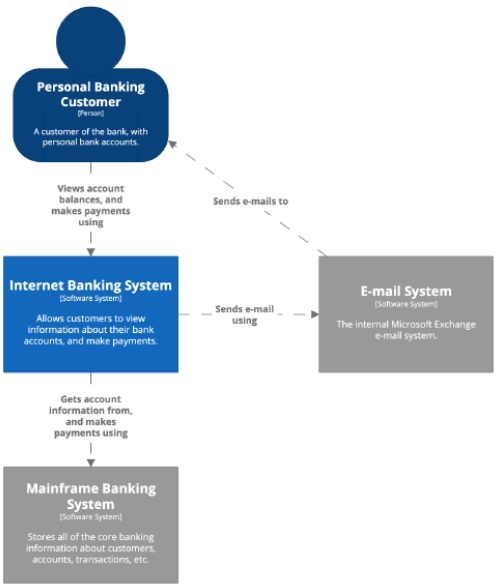
Vamos a continuación a hablar de algunas de las notaciones de lenguaje de modelado más importantes en el ámbito del análisis de requisitos. Como ya hemos mencionado en el video anterior, la preocupación por modelar sistemas viene desde los primeros años de la ingeniería de software. En la década de 1970 y de 1980 eran muy populares distintos enfoques basados en análisis estructurado. La notación IDEF (Integration DEFinition) es de esa época, como así también los diagramas de flujo de datos (DFD: Data Flow Diagram) y los diagramas de identidad-relación (ERD/ERM: Entity-Relationship Diagram/Model). Más recientemente, se han popularizado otros lenguajes de modelado, como por ejemplo UML (Unified Modeling Language), MPMN (Business Process Model and Notation)y SysML (Sstem Modeling Language).

Modelos y diagramas

Vamos a presentar a continuación algunos de los modelos de análisis más populares:

Diagrama de contexto

C4 es una anotación para describir arquitectura de software. Si bien la arquitectura de software es un tema que no tiene que ver directamente con el análisis de requisitos, esta notación nos ofrece un diagrama muy útil para describir la vinculación del sistema con su medio ambiente. Es el llamado, muy adecuadamente, diagrama de contexto.



Como se puede ver en la figura, el diagrama incluye un icono para representar al usuario, un rectángulo (en este caso celeste) que representa el sistema a desarrollar y (en este ejemplo en particular) 2 rectángulos grises que representan los sistemas con los que, el sistema en cuestión, deberá interactuar.

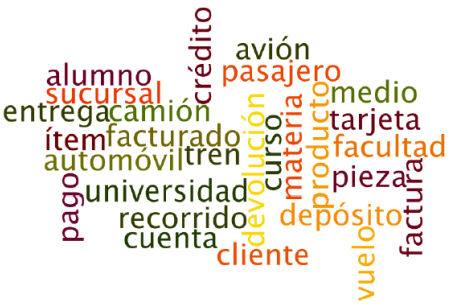
Como puede verse, es un diagrama muy sencillo, tecnológicamente neutro, que es muy útil para describir el medio ambiente en el que se encuentra el sistema y los principales vínculos que mantiene este con las entidades que lo rodean.

Modelo de casos de uso

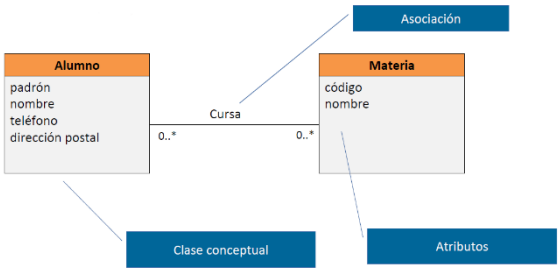
Un diagrama que tiene un propósito similar es el de **caso de uso**. Este es un diagrama que está incluido en el estándar UML, cuyo objetivo es representar los casos de uso del sistema y los actores primarios y secundarios con lo que el sistema interactúa. Estos actores representan tipos de usuarios que utilizan el sistema y también pueden representar otros sistemas que proveen o reciben información de este. Es un diagrama muy sencillo que cuando se utiliza en combinación con las especificaciones del caso de uso, forman parte del modelo de casos de uso.

Modelo de dominio

Un modelo de dominio es una representación visual de los objetos o clases conceptuales del dominio del problema y de las asociaciones que éstos tienen entre sí. En este contexto, los objetos de dominio representan cosas que el sistema en cuestión necesita manejar o conocer (ejemplo: objetos manipulados en una organización: contratos, facturas, pedidos), objetos y conceptos del mundo real que el sistema necesita conocer o monitorear (ejemplo: avión: trayectoria, misil), eventos pasados o futuros (arribo, partida, pago), personas, roles u organizaciones (clientes, socios, alumnos). Construimos este tipo de modelos fundamentalmente para entender y analizar mejor el contexto en el que operará el sistema que tenemos que desarrollar. Usualmente lo utilizaremos en combinación con otras técnicas como casos de uso o historias de usuario.



Para representar gráficamente los modelos de dominio, empleamos un diagrama de clases con la notación UML. La única salvedad que tenemos que hacer es que se deben incluir únicamente clases conceptuales, atributos y asociaciones. Estas clases no representan objetos del lenguaje de programación, no representan clases del lenguaje de programación, sino que representan objetos del negocio. Con lo cual, todos los aspectos que tengan que ver con la implementación deben ser removidos del diagrama. Recordemos, es un modelo de análisis y no un modelo de diseño de software.



[**V** / F] En pos de entender y comprender mejor el contexto del sistema a desarrollar el modelo de dominio es una técnica de análisis adecuada.

Árbol de funcionalidades (prestaciones) [feature tree]

Otro modelo de análisis muy útil es el **árbol de funcionalidades o feature tree**. En el ejemplo lo vemos representado como un diagrama, similar al de causa efecto o Ishikawa. El objetivo es mostrar cuáles son las prestaciones o funcionalidades que componen el sistema y cómo éstas se pueden desglosar en prestaciones más pequeñas, de menor nivel. La estructura de descomposición típica sería: prestación, subprestación (o varios niveles de subprestación), requisitos de usuario, requisitos de software.

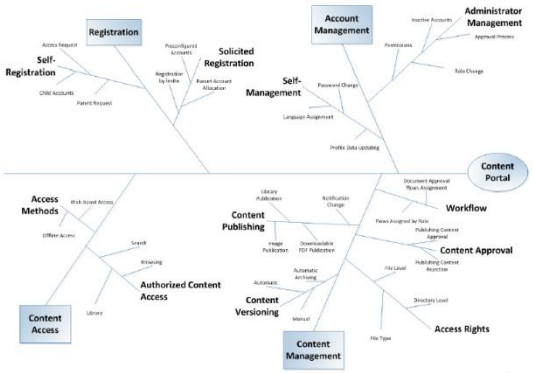


Diagrama de Estados

El **diagrama de Estados** o de estado-transición, es un diagrama que permite representar los diferentes estados, por lo que puede pasar un objeto de dominio o un sistema. El software no solamente implica funcionalidad, sino que también implica manipulación de datos y también cambios de estado. Representar estos cambios de estado mediante lenguaje natural puede ser bastante problemático, por la ambigüedad que plantea, por eso esta herramienta es sumamente útil. Por otro lado, en sistemas de tiempo real, es decir, aquellos que ante un momento determinado en producir una respuesta dentro de un período específico, prácticamente es imprescindible desarrollar un diagrama de estas características.

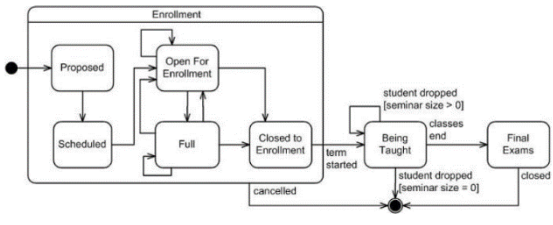
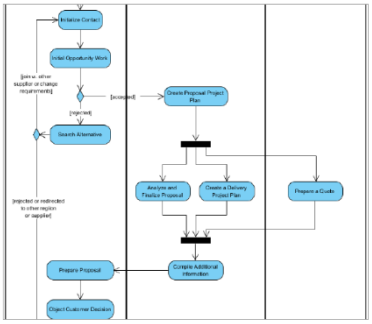
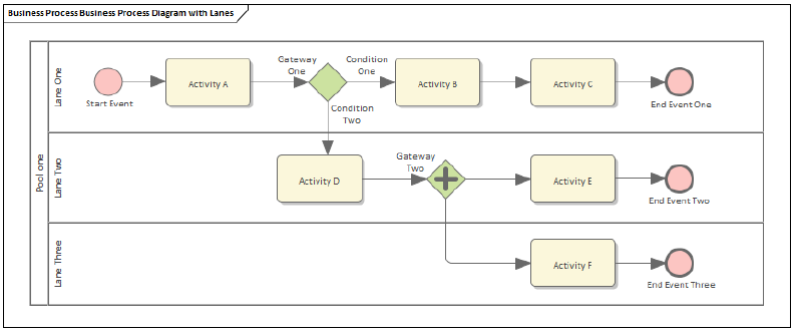


Diagrama de actividades (UML)



Permite especificar las actividades o pasos dentro de un flujo de trabajo, dentro de un proceso, e inclusive, dentro de un caso de uso. Cada rectángulo en el diagrama representa una actividad o paso, las flechas representan el paso de control de una actividad a otra, los rombos representan punto de decisión, y las columnas verticales pueden representar actores, roles o sistemas.

Flujo de actividades en un proceso de negocio (BPMN)



Un modelo similar al anterior es el **flujo de actividades** propuesto por la notación **BPMN** (Business Process Modeling and Notation). Este es un estándar de la OMG (Object Managment Group). Este diagrama está orientado exclusivamente a representar procesos de negocio, procesos organizacionales. Nuevamente aquí los rectángulos son actividades, las flechas representan la secuencia de ejecución y los andariveles (swimlanes) representan roles o áreas organizacionales. Además de ser una herramienta para modelar procesos, la idea detrás de esta narración también es que sirva para generar un proceso que pueda ser ejecutado por herramientas especializadas, es decir, que pueda generar un esquema ejecutable del proceso. La notación es relativamente sencilla y muy fácil de aprender.

Prototipado

Nuevamente, los **prototipos** aparecen como una muy buena herramienta, en este caso no sólo para descubrir requisitos, sino también para analizarlos. La idea con los prototipo de las interfaces de usuario, es explorar cómo podrían ser dichas interfaces, con el propósito fundamental de entender mejor el comportamiento del sistema a construir. Nuevamente, el tema aquí no es discutir aspectos de diseño, sino tratar de entender un poco más acerca de cómo el usuario entiende que utilizará el sistema.

EARS (Easy Approach to Requirements Syntax)

Si lo que estamos haciendo es especificar el lenguaje natural, hay varios lineamientos que se pueden utilizar para minimizar los problemas derivados de la ambigüedad. Uno de los lineamientos se llama EARS. Propone formatos muy específicos para describir requisitos de distinto tipo: Ubicuos, es decir, generales, que aplican a todo el sistema; relacionado con eventos; relacionados con estados; relacionados con comportamiento opcional o indeseado. En la siguiente tabla podemos ver los formatos que propone:

Ubicuo	Algo que el sistema debe hacer incondicionalmente.	N/A	“El sistema deberá cumplir con el estándar IRAM 34882”
Basado en eventos	Algo que el sistema debe hacer en respuesta a un evento disparador	Cuando	“Cuando arribe un mensaje, el sistema deberá emitir un alerta de notificación”
Basado en estados	Algo que se activa mientras se permanece en un estado determinado	Mientras	“Mientras el vehículo esté en movimiento, el sistema mantendr
Opcional	Algo necesario sólo bajo determinadas sircunstancias	Si	“Si se abre la puerta del vehículo con el motor en marcha, el sistema emitirá un alerta de peligro”
Comportamiento indeseado	Una respuesta del sistema a eventos no deseados	Si...entonces	

Criterios de aceptación

Una estrategia muy útil para completar la especificación es identificar criterios de aceptación. La idea detrás de esto es que el simple acto de pensar las condiciones bajo las cuales los requisitos van a ser dados por satisfechos (esos son los criterios de aceptación), ayuda a encontrar problemas en la especificación mucho antes de comenzar con el desarrollo del producto. Por otro lado, en escenario donde se utilizan historias de usuario, los criterios de aceptación, sirven para completar la especificación. Recordemos que las historias de usuarios no son requisitos de software en el sentido estricto de la definición, sino que son más bien recordatorios de que se debe mantener una conversación. Agregarles criterio de aceptación a estos requisitos de usuario, los acercan un poco más a la definición clásica de requisitos de software, o sea que a través de ellos empezamos a detallar un poco más cuál es el comportamiento esperado, qué es lo que pasa en distintos escenarios de utilización, en distintos escenarios o variantes de esa historia de usuario.

[**V** / F] Los criterios de aceptación ayudan a clarificar el alcance de la historia de usuario y permiten derivar casos de prueba.

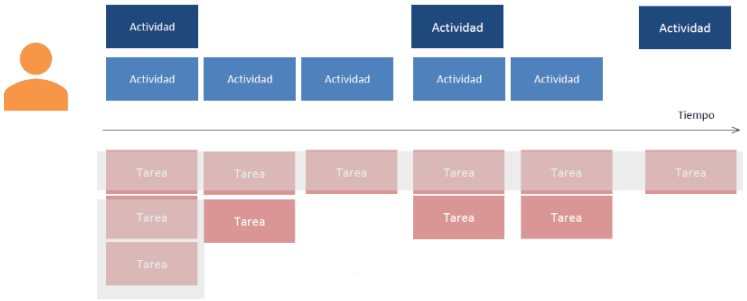
Reglas de negocio

También podemos utilizar reglas de negocio. Todas las organizaciones están sometidas a regulaciones de algún tipo, a leyes, estándares, a políticas. Estos principios de operación y de control son colectivamente conocidos como reglas de negocio. Por ejemplo, el IVA es del 21%, las facturas se deben emitir el último día hábil del mes o los usuarios menores de 18 años deben ser autorizados por sus padres, tutores o encargados. Las reglas de negocio pueden describir entonces cálculos complejos, restricciones, inferencias o simplemente hechos.

Hay 2 técnicas que vamos a mencionar y que vamos a explicar en detalle más adelante:

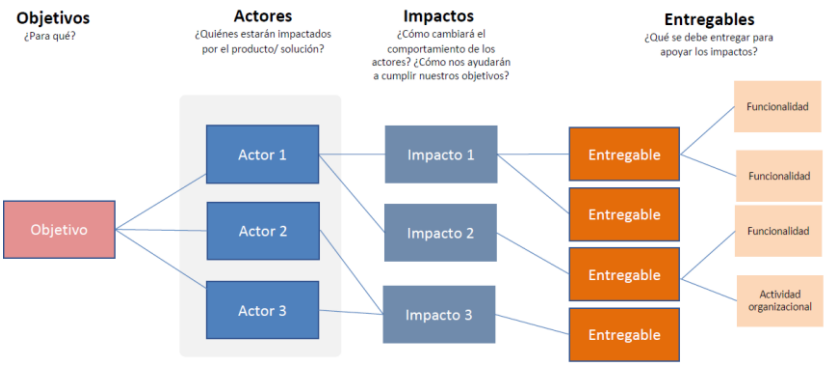
User Story Mapping

Es un modelo que describe las actividades que realizan a lo largo del tiempo los usuarios de una aplicación y cómo esas actividades se descomponen en subactividades y tareas. A partir de las tareas de último nivel podemos identificar historias de usuario. El diagrama se lee de izquierda a derecha.



Impact mapping

Mapa de impacto o impact map, es un diagrama que tiene como propósito alinear los equipos de trabajo con los objetivos de la organización. Este diagrama también se lee de izquierda a derecha. Primero se identifican los objetivos que se buscan satisfacer con la solución, o producto desarrollar. Luego, se identifican los actores que serán impactados por la solución. Y finalmente, qué es lo que se deberá hacer para producir esos impactos en esos actores, es decir, qué entregables tenemos que producir, qué productos, qué servicios, qué funcionalidades. Si seguimos descomponiendo esa estructura, en el último nivel vamos a encontrar realmente, historias de usuario. Es una herramienta muy muy útil para identificar requisitos y, obviamente, para alinear esos requisitos con los objetivos que se persiguen.



Recomendaciones

Para ir finalizando, van algunas recomendaciones:

Generales

- Es importantísimo **definir cuáles son los límites** del sistema que estamos analizando, si no lo hacemos, vamos a estar profundizando en temas que a lo mejor no nos corresponde hacer.
- Hay que estar **preparado para gestionar** los inevitables conflictos que van a surgir. No todo el mundo tendrá la misma visión, no todos los requisitos serán consistentes desde el comienzo.
- Tampoco todos los requisitos serán igualmente importantes, los recursos y el tiempo son limitados, con lo cual es imprescindible que **clasifiquemos y prioricemos**.
- Ya lo hemos mencionado antes al hablar de las actividades de descubrimiento, es importante **mantener la trazabilidad vertical y horizontal**. Necesitamos saber de dónde vienen los requisitos y también necesitamos saber qué artefactos están relacionados con ellos. Por ejemplo, si hay un cambio en un requisito determinado, qué otros requisitos se podrían ver afectados, qué impacto tiene ese cambio en los criterios de aceptación, en los casos de prueba, en las estimaciones, en los planes; qué cambios hay que hacer en el modelo de dominio; cómo afecta el cambio al diseño y al código. Si nos organizamos bien de entrada, las respuestas a todas estas preguntas deberían ser sencillas de elaborar, en caso contrario, estaremos desperdiciando mucho tiempo (que por otro lado es valioso) y que deberíamos distraer de otras actividades.
- También mencionamos con anterioridad la importancia de **evaluar riesgos**. Aquellos requisitos que impliquen un nivel de riesgo por encima de determinado umbral, ya sea por su complejidad o por lo difícil de trasladar al código o por cualquier otra razón, merecen tener una prioridad más alta, y probablemente, merezcan ser analizados con mayor nivel de detalle, probablemente a través de algún prototipo.

Modelado

- Tenemos que definir **cuáles son los modelos que nos conviene** desarrollar de acuerdo al escenario en el que estemos. No todos los modelos son útiles en todas las situaciones. También es importante que los modelos guarden consistencia entre sí. Por ejemplo, los objetos de dominio deberían estar mencionados en algún caso de uso, y los objetos de dominio que mencionemos en los casos de uso o en las historia de usuarios deberían estar incluidos en el modelo de dominio.
- También es importante decidir **qué modelos serán necesario formalizar**. Una gran parte del modelado será informal, como propone Agile Modeling, pero probablemente algunos modelos merezcan tener una versión en algún medio electrónico, que, por otra parte, nos va a permitir compartir.
- Al modelar, tenemos que procurar hacerlo desde **varios puntos de vista**. Para el aspecto objeto de dominio, o datos tenemos el modelo de dominio; para el aspecto funcional, podemos utilizar el modelo del caso de uso o User Story Map; para el aspecto control, que es muy importante en sistemas de tiempo real, podemos emplear diagramas de estado.
- Y, por supuesto, el **prototipado** es una técnica muy adecuada para, como ya dijimos, explorar requisitos poco claros o riesgosos.

Especificación

- Es muy importante **determinar el nivel de formalidad**. Si el equipo de desarrollo es relativamente pequeño y se encuentra físicamente en el mismo lugar, probablemente alcance con historias de usuario y con criterios de aceptación.
- Ahora bien, si el equipo es más grande o hay dispersión geográfica o hay una relación contractual entre la organización que produce el software y la organización que lo está solicitando, probablemente sea necesario **desarrollar estándares para una especificación** un poco más formal.
- En todos los casos es importante **utilizar un lenguaje claro, consistente y conciso**. Esto es independiente de la técnica que utilicemos. Siempre hay que recordar *que un requisito se escribe una vez pero se lee muchas veces.*
- Por eso es muy importante empezar los requisitos en forma **cuantitativa** (siempre que se pueda).
- Utilizar un **glosario o diccionario de datos** para aclarar palabras o expresiones particulares del dominio del problema.

Conclusiones

Para recordar

Recordemos entonces que, **análisis y especificación usualmente se realizan en simultáneo**. Recordemos también que analizar no consiste solamente en desarrollar modelos. El objetivo es **encontrar conflictos, encontrar inconsistencias** y resolverlos. Ya hemos también mencionado la importancia que tiene que definir cómo se van a **organizar y gestionar los requisitos**: ¿Los vamos a poner en un repositorio compartido?, ¿los vamos a pegar en un pizarrón?, ¿los vamos a pegar en una pared? Nuevamente no hay que perder de vista que los modelos ayudan a entender el problema y a definir el sistema deseado pero, hay que analizar **seriamente cuáles son los modelos que nos conviene** construir, para qué, con qué grado de formalidad. En algún entorno muy, muy específico, los modelos de análisis, a través de sucesivas transformaciones, pueden transformarse en modelos de diseño y luego, en código fuente. Esto es lo que pasa en Model Driven Design, por supuesto, no es la generalidad de los casos, por eso es importante determinar qué nos va a convenir desarrollar y qué modelo no nos va a convenir desarrollar. Lo mismo ocurre con las especificaciones. Tenemos que analizar cuidadosamente qué tipo de especificaciones producir, para quienes, para qué.

Y finalmente, lo que tenemos que evitar en todos los casos es la parálisis del análisis. Esa tendencia a sobre analizar y a no poder avanzar. En algún momento la actividad de análisis tiene que darse por concluida para poder seguir con el desarrollo.

[**V** / F] Generalmente los casos de uso y las historias de usuario se van identificando y refinando progresivamente a medida que avanza el proyecto.