Ingeniería en Desarrollo de Software
3er semestre

Programa de la asignatura
Introducción a la ingeniería de software

Unidad 2
Análisis y modelado de requerimientos

Clave:

Ingeniería: TSU:

15142318 / 16142318

Universidad Abierta y a Distancia de México

Índice

Unidad 2. Análisis y modelado de requerimientos

Presentación de la unidad

En la unidad anterior comprendiste los diferentes tipos de procesos de desarrollo de software, por ejemplo, el de cascada, el incremental, el espiral, etc., cada uno con sus propias etapas. Sin embargo, todos tienen en común una etapa destinada para el análisis del proyecto. Como parte de esta etapa se requiere de una serie de actividades que tiene que desempeñar el analista de un proyecto para poder identificar cuáles serán las funciones que el software deberá realizar. Estas actividades se verán más adelante.

El objetivo de esta unidad es aprender los tipos de requerimientos de software que existen, los métodos de recolección para identificarlos, documentarlos y por último la manera de representarlos utilizando estándares de modelado como lo es UML, Lenguaje Unificado de Modelado (por sus siglas en inglés *Unified Modeling Language*).

Los requerimientos son las necesidades o especificaciones que tiene el cliente respecto a un producto. Para fines de esta asignatura, se entiende que ese producto es el **software.** En esta unidad, es importante que se logre la comprensión de cómo obtener una adecuada descripción de las necesidades que el cliente pretende cubrir con las funciones del software, para poder transformarlas en requerimientos claros y bien definidos para el equipo de trabajo y para el mismo cliente. Los conjuntos de requerimientos serán desarrollados y gestionados a través de las etapas del proceso de desarrollo y poco a poco serán transformados en un software que reunirá todas esas características solicitadas. A continuación se mostrara un ejemplo para mejor comprensión:

Cuando un cliente solicita que se le desarrolle un software para llevar el control de las existencias de su almacén de materia prima, ya que la empresa ha crecido y por lo tanto el almacén cada vez tiene una mayor cantidad de materiales, y provoca que la actual forma de trabajo, realizada con hojas de cálculo, resulte cada vez más difícil de actualizar y mantener. Por lo que ha decidido que un software construido para este fin podría ser una solución a esta necesidad.

El ejemplo previo representa las necesidades que se deben detectar y traducir en requerimientos y posteriormente representarlos en un lenguaje de modelado para que sean entendibles para el equipo de desarrollo de software.

Propósito

Al término de esta unidad lograrás:

- Identificar:
    1. Tipos de técnicas de recolección.
    2. Requerimientos de un caso de estudio.
    3. Diagramas del dominio y de interacción

Competencia específica

- Analizar los diagramas del dominio e interacción, para la representación gráfica de los requerimientos de un caso de estudio, tomando en cuenta los estándares del Lenguaje Unificado de Modelado (UML).

2.1. Obtención y especificación de requerimientos

Sofware

Muchos proyectos de desarrollo de software no llegan a terminar en tiempo y forma de acuerdo con lo planeado, debido a que **los requerimientos** no se obtuvieron ni se especificaron completamente, o no fueron muy precisos. Durante la construcción del software se van identificando inconsistencias, por ejemplo, puede haber duplicidad de requerimientos por un mal manejo de nombres que se han registrado varias veces, pero en esencia son los mismos; otro ejemplo sería la inconsistencia en revisiones de lo que se requiere; es decir, a veces con una revisión

puede parecer entendible, pero en el momento de tratar de traducirla para generar algo, no es posible por falta de datos o porque simplemente no se comprende.

Ahora bien, en la obtención de requerimientos, se tiene que las tareas del cliente y del analista serán llegar a la definición adecuada de requerimientos procurando que las necesidades del cliente respecto al software se cumplan y se puedan describir en términos entendibles para el equipo de desarrollo (líder, analista, diseñadores, programadores, encargados de pruebas). Por ello, la función del analista o ingeniero de requerimientos es ser un interrogador-consultor, que será de ayuda al cliente para determinar la completa definición de requerimientos.

De esta manera, se puede decir que los requerimientos son la base del desarrollo de software, porque servirán para entender qué es lo que se va a producir, por lo que éstos deberán describirse de manera clara, concreta, completa y sin ambigüedades (evitar que sean confusos, mal escritos, redundantes). Se debe tener presente que los lectores de los requerimientos serán personas que posiblemente no conocen aspectos técnicos, por ejemplo, el cliente, usuarios, contratistas y, por otra parte, personal altamente especializado como diseñadores, programadores, arquitectos de software, encargados de pruebas y de implantación, el líder, y el mismo analista. Para describirlos, se puede comenzar a clasificarlos como requerimientos del usuario y requerimientos del sistema, tal como se muestraa continuación:

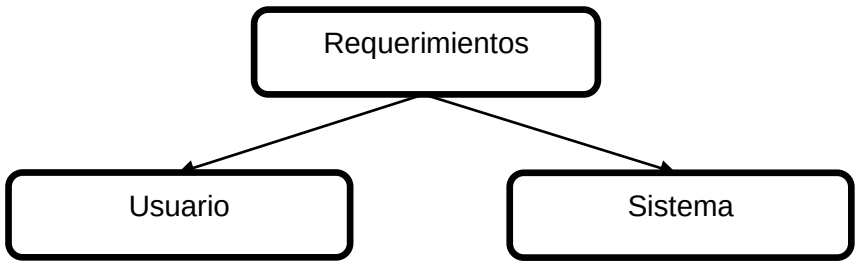


Diagrama de clasificación de requerimientos

En este diagrama se visualiza **la primera** clasificación que observa y comprende lo que Sommerville (2011, p. 83) define, “los requerimientos del usuario son enunciados, en un lenguaje natural junto con diagramas, acerca de qué servicios esperan los usuarios del sistema, y de las restricciones con las cuales éste debe operar”.

Un ejemplo de esto puede ser el siguiente:

Requerimiento del usuario.

> "El usuario puede realizar la consulta de las existencias del almacén de materia prima de manera remota vía internet."

En seguida se definen los principales requerimientos del usuario:

Casos de uso: representaciones gráficas de las funciones de un software. Sus elementos son el caso de uso o función del software, actor que es representado por una figura de un "monigote" y los arcos que son líneas que relacionan al actor y los casos de uso.

Escenarios: son descripciones del usuario de lo que se espera que el software realice. Estas descripciones se realizan como una historieta y son consideradas como requerimientos. Generalmente se utilizan en modelos de desarrollo ágiles.

Prototipos: representaciones tipo borrador de la construcción del software, generalmente se utiliza para mostrar interfaces y su navegación.

La segunda clasificación son los requerimientos del sistema. Observa la siguiente definición de Sommerville (2011, p. 83): "los requerimientos del sistema son descripciones más detalladas de las funciones, los servicios y las restricciones operacionales del sistema de software".

Ejemplo de requerimiento del sistema.

> "Todos los viernes el software generará el reporte denominado "Explosión de materiales" que consiste en mostrar la materia prima que se necesitará para la producción de la siguiente semana. El reporte deberá indicar de color rojo los materiales que deberán comprarse y las cantidades para poder producir y mantener los niveles de stock adecuados."

La manera de representar los requerimientos del sistema es el documento de especificación de requerimientos del software o SRS (*Software Requierement Specification*), que contiene todos los requerimientos descritos con un alto nivel de detalle.

A veces forma parte del contrato entre el comprador del software y el equipo de desarrollo.

De cualquier manera, aun y con todo el cuidado que se haya tenido al detectar los requerimientos, éstos podrán presentar cambios durante el desarrollo del software. Sin embargo, un adecuado control mantendrá el proyecto dentro del margen de lo planeado en tiempo, costo y recursos.

Los requerimientos del sistema del software se clasifican como **funcionales** y **no funcionales**. Esta clasificación sirve para identificar de una mejor manera cuáles son los requerimientos que se refieren a las características para concebir el software como un "todo" y qué funciones el sistema debe realizar de manera específica. En el siguiente tema encontrarás una explicación más amplia de este tipo de requerimientos.

2.1.1. Requerimientos funcionales y no funcionales

Como se ha mencionado en el tema anterior, la redacción de requerimientos puede ser poco exacta, esto puede generar problemas en su interpretación, ocasionando que no se logre el entendimiento del software. Es por ello que hay otra manera de clasificar los requerimientos: una subclasificación de los requerimientos del sistema que los divide en funcionales y no funcionales:

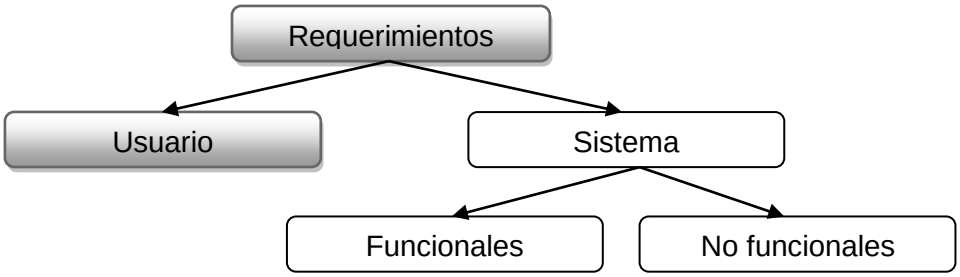


Diagrama de clasificación de requerimientos y de requerimientos del sistema

Para entender de forma clara, observa la siguiente definición que Sommerville (2011, p. 84) da sobre los requerimientos funcionales: "son enunciados acerca de servicios que el sistema debe proveer, de cómo debería reaccionar el sistema a entradas particulares y de

cómo debería comportarse el sistema en situaciones específicas. En algunos casos, los requerimientos funcionales también explican lo que no debe hacer el sistema".

Un ejemplo de requerimiento funcional sería:

> "El software debe enviar una alerta cuando los niveles de stock o reservas han comenzado a utilizarse y debe sugerir generar la orden de compra."

Un requerimiento no funcional podría estar conformado por varios requerimientos funcionales, que generalmente son restricciones en presupuesto, políticas de la empresa, imposiciones del gobierno, vínculos con algún otro sistema de información o atributos de calidad del sistema. Estos requerimientos se refieren al software como una sola entidad y no como un conjunto de elementos. Por ejemplo:

> "El software debe estar accesible para realizar las consultas en tiempo real vía internet las 24 horas del día los 365 días del año."
>
> "El software debe restringir el acceso a personas ajenas a la organización."

Es importante comentar que al redactar la descripción de cada requerimiento no funcional, éste debe cuantificarse de alguna manera, porque de lo contrario dicha descripción quedará imprecisa provocando una mala interpretación llena de subjetividad. Algunas expresiones comunes son las que se encuentran en la siguiente tabla que se presentan a continuación, con dos columnas; la primera se llama "Descripción incorrecta", la cual contiene palabras o frases que normalmente son usadas en la descripción de algún requerimiento no funcional, pero no debe ser así; la segunda columna, llamada "Descripción correcta", ofrece ejemplos para quitar la imprecisión del requerimiento, haciendo éste más cuantificable, medible y por lo tanto alcanzable.

Descripción incorrecta	**Descripción correcta**
Que el software sea **rápido (veloz, en tiempo real, lento)**	El tiempo de despliegue de las páginas estáticas será dado en un rango de tiempo de **<=5 segundos.** Otros elementos pueden ser **transacciones**, **tiempos de respuesta** del usuario, **eventos**, todo sobre la unidad de tiempo.

Será un software **pequeño (grande, mediano)**	Se espera que el archivo que se genere no exceda los **10,000 KB** para que pueda ser descargado fácilmente por el usuario desde internet. Bits, Kb, B, KB, TB, GB, etc.
Que sea **fácil de usar (sencillo, amigable, lógico)**	El software contará con un módulo de **ayuda** de cada uno de los procesos y ventanas. También se puede disminuir la ambigüedad del requerimiento indicando que el personal será capacitado por un cierto periodo de tiempo estimado para que pueda aprender a utilizarlo.
La información que genere debe ser **fiable (veraz, oportuna)**	El software deberá realizar procesos de eliminación de registros incompletos cuando estos no hayan sido completados cuando el software se haya apagado por fallos eléctricos. Conservando de esta manera, solo los registros completos. Otras consideraciones son la probabilidad de que el sistema no esté disponible (si se hacen respaldos o migraciones a otro servidor).
Que el software sea **robusto (potente)**	Cuando el software detecta algún error de autentificación, por ejemplo la pérdida de variables de sesión o cookies, lanzará una página indicando al usuario que vuelva a introducir su número y contraseña. Otros casos podrían ser indicar si se requiere el reinicio después de una falla e incluso podría indicar en qué porcentaje los datos están corruptos por un error o falla.
El software debe ser **portable (portátil, transportable, movible)**	Se espera que el software pueda ser visible en los navegadores iExplorer v. 9, Chrome v. 18, Mozilla 6. La idea es que describa exactamente en qué otros sistemas o equipos debe funcionar el software.

Tabla para describir requerimientos no funcionales

Observa el siguiente ejemplo que demuestra un requerimiento no funcional y que también se encuentra mal definido:

> “El software debe ser fácil de usar para el personal del almacén.”

Como pudiste ver es incorrecto porque no es posible cuantificar o medir su cumplimiento, ahora se presentara el mismo requerimiento no funcional descrito de forma correcta:

> “El personal del almacén tomará una capacitación de 3 horas en el uso del software, además el software contará con videos explicando la funcionalidad del mismo como apoyo extra.”

Los requerimientos son la base del desarrollo de un sistema de software, es por ello que se hace un especial énfasis en su obtención, especificación, clasificación y buena redacción. Pero no siempre se cuenta con la información necesaria a la mano, para ello habrá que utilizar algunas técnicas de recolección para identificar y priorizar los requerimientos. Este tema se abordará en el siguiente punto.

2.1.2. Técnicas de recolección, identificación y priorización de requerimientos

Obtener la información de la descripción de un software no es una tarea sencilla. Sería deseable que todos los clientes tuvieran la definición, clara, precisa y detallada de lo que esperan que una empresa de desarrollo de software construya para la solución o mejora de sus procesos. Sin embargo, esto no siempre es así. A veces, se necesita utilizar una o varias estrategias para llegar obtener la información necesaria que ayude a determinar detalladamente dichas características. En este tema se analizarán algunas técnicas que se utilizan para este fin.

El proceso de desarrollo de requerimientos según Sommerville (2011, p. 102) consta de cuatro fases, las cuales deberán realizarse iterativamente hasta determinar que ya se tienen completamente los requerimientos que representan la funcionalidad del software y que el cliente espera:

1. **Descubrimiento de requerimientos.** Esta etapa incluye actividades para interactuar con los involucrados del sistema para identificar los requerimientos y toda la documentación relacionada: diagramas y descripción de casos de uso, escenarios, SRS, diagramas de dominio e interacción, etc. Mismos que se explican con mayor detalle en los siguientes temas.

2. **Clasificación y organización de requerimientos.** Consiste en organizar el listado de requerimientos agrupándolos de acuerdo con su contenido lógico. Es decir, en el orden en que se realizan los procedimientos en la realidad y en un orden funcional, que vaya de acuerdo a la secuencia en que se ha diseñado el software para su operación. Se deberán identificar requerimientos que se relacionen y puedan formar módulos o divisiones del software.

3. **Priorización y negociación de requerimientos.** Consiste en identificar a los requerimientos que tienen mayor importancia o urgencia de ser desarrollado respecto a los demás, este es un proceso de negociación entre los diversos proveedores de requerimientos de la organización.
4. **Especificación de requerimientos**. Este proceso consiste en ir documentando los requerimientos con todo el detalle posible, por ejemplo la descripción de los casos de uso es una muestra de cómo se puede describir detalladamente un requerimiento. Un documento más completo y a la vez más complejo es el SRS, que integra todos los diagramas y su descripción, que se hayan realizado para describir cada requerimiento. Cada vez que se repite un ciclo de este proceso el detalle aumenta y a esto se le llama refinamiento de requerimientos.

Como ya se mencionó estos cuatro pasos se repiten y la comprensión de los requerimientos por parte del analista mejora en cada iteración. La comprensión de los requerimientos por parte de los involucrados es un proceso difícil, principalmente porque el cliente pocas veces tienen bien definido lo que necesitan de un sistema de cómputo y pueden hacer peticiones inalcanzables en tiempo, costo o recursos. Además, utilizan sus propios términos, también suelen decir un mismo requerimiento de muchas maneras haciendo entender que se tratan de cosas diferentes.

Otra dificultad para el equipo de desarrollo es cuando, a la mitad del proceso de creación del software, se realizan cambios en la organización por nuevos participantes que no se habían involucrado desde el inicio del proyecto, lo cual puede afectar en tener que volver a realizar otras descripciones en los requerimientos o incluso insertando nuevos requerimientos. Llevar una gestión adecuada del cambio puede ayudar para atenuar el problema.

A continuación se describirán algunas técnicas para obtener información de la descripción de un software: observación, cuestionario, entrevista y encuestas:

La observación

Partiendo de la idea de que todo software es un modelo de un modelo del usuario, se debe identificar cómo es que el usuario opera su modelo para entonces poder simular esto con un software. Es por ello que la observación es una técnica sencilla, pero poderosa, debido a que permite ser testigos de cómo operan las cosas en una organización.

Para aplicar la técnica de la observación, el analista deberá estar presente en el lugar donde se ejecuta el proceso que se desea observar, analizando la funcionalidad de la operación que se quiere construir en un software. Su principal ventaja es que se observan todos los detalles del funcionamiento real de la empresa (que a veces no es como se encuentra descrito en los manuales de proceso) y además se detecta el ambiente en el que se instalará el proyecto

Entrevistas

Las entrevistas son importantes, porque permiten estar en contacto con el cliente y se pueden combinar con otras técnicas de recolección de información, como son la observación o escenarios, entre otras. Las entrevistas se pueden realizar de dos maneras: cerradas y abiertas

1. Cerradas: los participantes responden a un conjunto de preguntas. No se pueden agregar más preguntas.

2. Abiertas: **no hay** un conjunto de preguntas predefinidas solo se abarca una problemática específica, las preguntas van surgiendo de acuerdo con lo que el entrevistador quiere indagar o descubrir.

En la práctica puede realizarse la combinación de ambos tipos de entrevistas. El analista prepara un conjunto de preguntas, que incluso, puede enviar previamente al cliente. Durante la entrevista, estas servirán de guía. El analista podrá realizar nuevas preguntas para obtener mayor detalle; basarse en las respuestas dadas o para indagar un nuevo tema que no haya contemplado en la entrevista cerrada. El analista podrá solicitar al cliente ejemplos o documentos de la organización para completar las respuestas.

Escenarios

Se utilizan para comprender como funcionaría el sistema de software en un escenario real. Los analistas utilizarán esta información como requerimientos del sistema. Los escenarios consisten en ejemplos sobre las descripciones de las sesiones de cada interacción, comienzan con el bosquejo de una iteración y se va detallando en cada una. Esta técnica se aplica generalmente en la programación extrema. Cada escenario debe incluir la descripción de:

1. **El evento que dispara el proceso**, corresponde con la explicación de la actividad que genera el uso del proceso que posteriormente será realizado en el software.
2. **Flujo normal,** se refiere a la secuencia de actividades que se realizan para operar el proceso que se pretende automatizar por medio del software.
3. **Qué puede salir mal,** describir aquellas actividades que no están incluidas en el flujo normal, pero que si llegasen a ocurrir alterarían la secuencia del proceso o causarían algún error.
4. **Otras actividades,** actividades que pueden o deben ejecutarse al mismo tiempo, es decir, actividades paralelas.
5. **Estado final del sistema,** describir el estado del software cuando las actividades del proceso llegan al fin.

A continuación se verán los puntos anteriores con un ejemplo de un escenario:

Lo que dispara el proceso:

Todos los lunes desde las 8:00 am llegan los proveedores con diferentes materias primas, este proceso es el resultado de las compras que se generaron durante el viernes de la semana anterior. Otros tipos de entradas es la materia prima de importación, ésta puede llegar en cualquier momento del día y cualquier día de la semana.

Flujo normal:

El almacenista revisa que el proveedor haya llegado con la orden de compra y factura. Posteriormente, hace la inspección física revisando que se haya surtido lo que se indicó en la orden de compra, tanto en cantidad como en las características del producto. Por último, revisa que la factura contenga exactamente lo que se pidió y que esté correctamente escrita. Cuando termina procede a dar la entrada física al almacén y genera el registro de entrada en el sistema. Realizando una búsqueda con las primeras tres palabras del artículo para ubicar su código y capturar la cantidad que está recibiendo.

Qué puede salir mal

Si el artículo no ha sido registrado previamente, no podrá localizarlo para registrar entradas. Por lo cual deberá registrarlo como un nuevo artículo obteniendo del sistema la clave que le corresponde de acuerdo con las reglas del negocio plasmadas en el documento “Registro de artículos”.

Otras actividades

Mientras se realizan estas actividades, las consultas y generación de reportes del sistema del almacén podrán estarse realizando sin ningún problema.

Estado final del sistema

Los registros de entrada al almacén deberán estar en la base de datos. Se deberán actualizar los campos de existencias y costos promedios del registro de cada artículo que ha tenido un registro de entrada.

Ejemplo de escenario: entrada de materia prima al almacén

Cuestionario

Esta técnica resulta útil cuando es necesario captar un gran volumen de información en poco tiempo. También cuando la separación geográfica impide que la información sea captada por otras técnicas. Para que un cuestionario sea útil, éste debe ser conciso y estar enfocado en pocos aspectos del sistema. Su redacción debe estar bien definida, debe ser precisa y clara. Se pueden combinar preguntas abiertas (puede redactar libremente una respuesta) y cerradas (la respuesta queda limitada a una cantidad de opciones); además, se puede incluir una descripción para explicar el objetivo del cuestionario y favorecer que el usuario conteste lo más objetivamente posible. Por último, puede contener una frase de agradecimiento (Piattini *et al*. 2004, p. 184).

A continuación presta atención al siguiente tema que describe el documento de requerimientos.

2.1.3. Documento de requerimientos

Existe mucha documentación que se va generando con los requerimientos, pero el llamado SRS (especificación de requerimientos de software) es un documento formal para detallar lo que debe implementar el equipo de desarrollo, ya que incluye a los requerimientos de usuario y del sistema.

Este documento SRS es útil por las siguientes razones:

- Es una herramienta de comunicación y de integración de las especificaciones del software entre el equipo de desarrollo.
- Puede ser utilizado como la base del contrato entre el comprador y la empresa que desarrolla el software.
- Se puede utilizar con el fin de estimar tiempos y costos del proyecto, y por lo tanto para poder generar el plan del proyecto.
- Es útil para describir las etapas de análisis, siendo uno de los principales entregables de estas etapas.
- Es útil para controlar los cambios que vayan surgiendo en el desarrollo del software.
- Ayuda a evaluar el producto final.

El SRS se desarrolla para varios lectores por ejemplo, clientes, administradores, desarrolladores, testers, etc. El contenido de un documento SRS de acuerdo al estándar de la IEEE (*Institute of Electrical and Electronics Engineers*) es un estándar genérico y se adapta a usos específicos. Como se muestra en la siguiente tabla.

CAPÍTULO	DESCRIPCIÓN
Prefacio	Describir a quién va dirigido, la historia de versiones con el resumen de cambios realizados en cada versión.
Introducción	Describe brevemente la necesidad que se cubrirá con el sistema, las funciones que tendrá y cómo funcionará con otros sistemas. Y cómo se ajusta el sistema a los objetivos de la organización.
Glosario	Define los términos técnicos que se emplean en el documento.
Definición de requerimientos del usuario	Aquí se presentan los servicios que ofrecen al usuario. También, en esta sección se describen los requerimientos no funcionales del sistema. Se hace uso del lenguaje natural o diagramas que entiendan los lectores.
Arquitectura del sistema	Se muestra una descripción de alto nivel de la arquitectura del sistema, mostrando su funcionalidad a través de módulos del sistema.
Especificación de requerimientos del sistema	Se muestran con detalle los requerimientos funcionales y no funcionales, e incluso, las interfaces a otros sistemas.
Modelos del sistema	Son gráficos que muestran relaciones entre componentes del sistema y su entorno. Por ejemplo gráficos de objetos, modelos de flujos de datos o modelos semánticos.
Evolución del sistema	Describe los posibles cambios futuros para el sistema como parte de una continuidad, ya sea ocasionada por el hardware, por el usuario o por la organización.
Apéndices	Información detallada y específica que se relaciona con la aplicación a desarrollar.
Índice	Pueden incluirse varios índices, el de contenido, de diagramas, de funciones, etc.

Tabla. Estructura de un documento de requerimientos. Tomada de Sommerville (2011, p. 93).

Como se puede observar, el documento SRS tiene algunas ventajas; sin embargo, de acuerdo con el enfoque de las metodologías ágiles se considera que no se debe llenar un documento tan extenso, porque al momento en que se inicia con su elaboración comienza a hacerse obsoleto, así que este documento se desperdicia en gran medida, por ejemplo, en el enfoque que presenta la programación extrema se recopila de manera incremental requerimientos del usuario y se escriben en tarjetas como historias de usuario. De esta manera, el usuario da prioridad a los requerimientos para su implementación en el siguiente incremento del sistema.

Ya sea que decida o no utilizar un documento tan específico como lo es un SRS o que se decidan por las metodologías ágiles cuya especificación es mínima, en ambas metodologías deberá existir la validación de los requerimientos para asegurar que están correctos. En el siguiente tema se explicará con mayor detalle el proceso de validación, pero antes realiza la primera actividad para la unidad.

2.1.4. Validación de requerimientos

La validación de requerimientos consiste en revisar que éstos definan adecuadamente el sistema que necesita el cliente. Este proceso es importante porque los errores en requerimientos pueden conducir a grandes costos al tener que rehacerlos, por esto es necesario que se realicen varios tipos de comprobaciones:

1. **Comprobación de validez.** Los requerimientos adicionales o derivados realmente se deben relacionar con lo que el cliente necesita.
2. **Comprobaciones de consistencia.** Los requerimientos en el documento no deben estar en conflicto, no deben ser contradictorios o tener descripciones diferentes de la misma función del sistema.
3. **Comprobaciones de totalidad**. El documento de requerimientos debe contener todas las peticiones del usuario y sus restricciones.
4. **Comprobaciones de realismo.** Revisar que los requerimientos sean factibles y alcanzables, tomando en cuenta la tecnología a emplear, presupuesto y conocimiento necesario para su desarrollo.

5. **Verificabilidad**. Los requerimientos deben ser entendibles para el cliente y el desarrollador, así como verificables.

Para realizar la verificación se utilizan varias técnicas, algunas de ellas son:

1. **Revisiones de requerimientos**. Personal del equipo de desarrollo revisa que no haya errores e inconsistencias.
2. **Creación de prototipos.** Se construye un modelo ejecutable del sistema para que el cliente pueda comprobar si cubre con las funcionalidades que solicitó.
3. **Generación de casos de prueba.** Al realizar casos de prueba, sí resulta difícil realizarla, o no se puede comprobar, esto indica que el requerimiento está mal definido o presenta alguna inconsistencia.

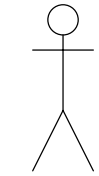
Ya que los requerimientos son más confiables y correctos, es posible comenzar a modelarlos con los diagramas UML. En los siguientes temas se verán diferentes diagramas de modelado.

2.1.5. Casos de uso

Traducir las peticiones del cliente en un lenguaje más técnico y formal, resulta ser una actividad a veces complicada, por lo que convendría apoyarse de alguna herramienta que pueda ser fácil de usar y comprender. Esto se cubre con los **casos de uso** que son diagramas del lenguaje unificado de modelado (UML) y también se consideran como una técnica de recolección de requerimientos. Se describe en un diagrama todas las iteraciones posibles entre los usuarios y el sistema.

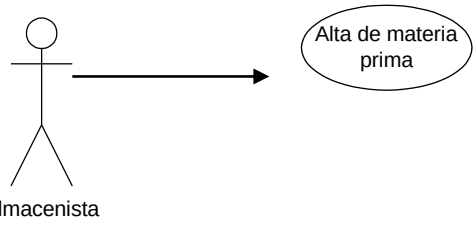
Son muy útiles para representar las funciones que el sistema debe tener y sus elementos son los siguientes:

- Actor. Se refiere a cualquier elemento que interactúa con el sistema. Éste puede ser un usuario, otro sistema o algún hardware. Se representa con un “monigote**”** con su nombre en la parte inferior. Dicho nombre debe comenzar con mayúscula.

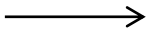


- Caso de uso. Son las funciones o comportamientos del sistema. Se representan con una elipse que contiene la descripción de la función en lenguaje natural o estructurado. Algunas veces si el nombre es muy largo es conveniente colocarlo debajo de la elipse.

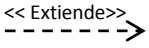
- Las líneas entre los actores y los casos de uso se denominan arcos de comunicación o relaciones.



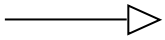
Hay diferentes tipos de relaciones:



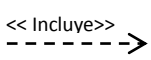
Asociación. Interacción entre el actor y el caso de uso.



Extiende. Comportamiento adicional de un caso de uso base. También se puede poner la palabra <<Utiliza>>



Generalización. Hereda las características de un caso de uso a otros generando una relación de especialización.



Incluye. Contiene el comportamiento de un caso de uso dentro de otro.

Los diagramas de caso de uso deben tener una descripción en la que se detallen las especificaciones de su comportamiento en flujo de actividades de secuencia normal y excepciones. Para lo cual se puede utilizar una tabla como la siguiente:

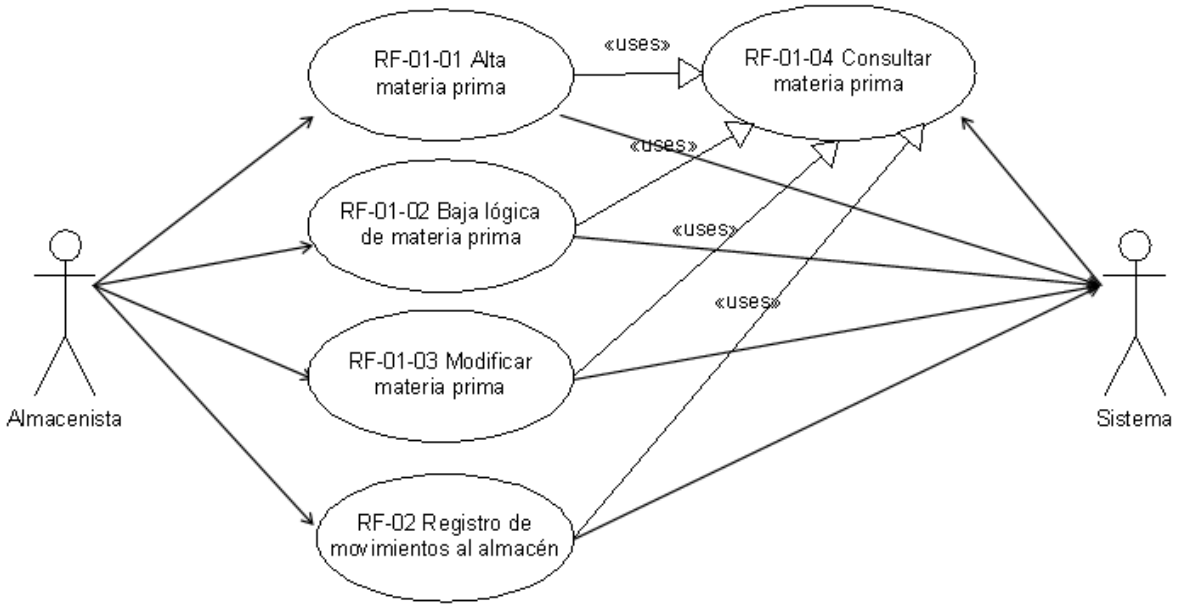
[Id requerimiento]	Nombre del requerimiento
Descripción	Lo que el sistema debe permitir con [actores] en un [tiempo] la [funcionalidad]
Requerimientos asociados	[id requerimiento] [Nombre del requerimiento] [id requerimiento] [Nombre del requerimiento] …
Secuencia normal	[# paso] [Acción] [# paso] Si [situación que produce alternativa] realizar [acción] …
Excepciones	[# paso] En el caso de que [situación que provoca la excepción] el sistema deberá [acción] …
Frecuencia	Cantidad de veces en que se lleva a cabo
Prioridad	Con respecto a otros requerimientos [Alta, media, baja]
Observaciones	Otras especificaciones necesarias

Tabla. Descripción de los casos de uso. Adaptada de Sommerville (2011, p. 125).

Ejemplo

RF-01	Registro de materia prima al almacén
Descripción	El sistema debe permitir al almacenista cada vez que exista una materia prima nueva registrarla como un nuevo material al almacén.
Requerimientos asociados	RF-01-01 Alta materia prima RF-01-02 Baja lógica de materia prima RF-01-03 Modificar materia prima RF-01-04 Consultar materia prima RF-02 Registro de movimientos al almacén

Secuencia normal	1. Almacenista consulta una materia prima en el sistema 2. Si existe Selecciona en el sistema la materia prima Si no existe Da de alta la materia prima asignándole un código 3. Registra movimiento de entrada al almacén.
Excepciones	1. En caso de que haya encontrado un error en la captura de la materia prima, el sistema debe tener la opción de modificar los datos. 2. En caso de tener materia prima descontinuada, el sistema debe permitir cambiar el estado a “Baja”, para que se muestren las consultas únicamente con materiales vigentes.
Frecuencia	El almacén recibe las compras todos los lunes en un horario de 8 am a 2 pm, sin embargo puede existir alguna compra no programada algún otro día de la semana.
Prioridad	Alta
Observaciones	Cuando se inserta una nueva materia prima el código que se le asigna también lo lleva en barras para facilitar la lectoescritura en el sistema.



Los diagramas de caso de uso pueden ser útiles para descubrir requerimientos en los procesos de los usuarios, pero también son de gran ayuda para describir a los requerimientos funcionales del software. Hay otros diagramas que también sirven para mostrar con otra vista a los requerimientos. Estos son los que a continuación se verán en el modelado del dominio y de la interacción.

2.2. Aplicación de modelo del dominio y de la interacción

Un modelo es la representación de un concepto, objeto o sistema. Se toma como referencia para producir algo igual. Enfocando esto al desarrollo de software, se puede decir que se necesita identificar los modelos del usuario de la forma en que realiza sus procesos y convertirlos en modelos con gráficos o símbolos que representen este proceso y que puedan ser entendibles para el equipo de desarrollo de software (analistas, diseñadores, programadores, responsables de pruebas, etc.).

El modelado de sistemas es una disciplina útil para representar la visión que se tiene del software que se va a construir o la descripción técnica del software terminado. El modelado de sistema consiste en la construcción de diferentes tipos de diagramas y el estándar que más se utiliza es UML, que cuenta con aproximadamente trece diagramas para describir diferentes perspectivas del software. En este tema se verán dos **categorías** de diagramas: de dominio del sistema y los de interacción.

Modelos del dominio del sistema:
Los diagramas de dominio se utilizan para representar aspectos de la realidad del contexto del software. No describen al software, más bien muestran su entorno, incluso incluyen a otros sistemas automatizados y el tipo de asociación que pueden tener.

Los modelos de dominio se pueden representar por medio de diagramas de clases donde puedes delimitar la frontera de un software y mostrar su entorno, que puede incluir o no, a otros sistemas automatizados y las relaciones que pueden tener. Un modelo de dominio consta de un conjunto de diagramas de clases con sus atributos, pero omitiendo la definición de sus operaciones. Los diagramas de clases se verán en el siguiente tema.

Modelos de interacción:

Los sistemas tienen interacciones de algún tipo, por ejemplo, con el usuario que implican entradas y salidas del mismo. También hay interacciones entre el sistema a desarrollar y otros sistemas o interacciones entre los componentes del sistema. En el modelado de caso de uso y los de secuencia muestran interacciones con diferentes niveles de detalle y es posible utilizarlos juntos. El diagrama de colaboración es otro ejemplo de este tipo de modelos de interacción. A continuación se explicará cómo puede modelarse la estructura de un sistema por medio de los diagramas de clases.

2.2.1. Diagramas de clases

Una manera de modelar la estructura de un sistema es utilizar un diagrama de clases. Los diagramas de clases ayudan a desarrollar un sistema con un diseño orientado a objetos. Una clase es considerada como una definición general de un tipo de objeto del sistema y está representada por un rectángulo con nombre de la clase (esto en su representación sencilla). La línea que une las clases es una asociación que indica un vínculo entre dichas clases. Los objetos representan cualquier cosa del mundo real, por ejemplo, un material, una factura, un almacenista, etc. Por parte del sistema se agregan objetos que le dan funcionalidad al sistema (Sommerville, 2011, p. 129).

Por ejemplo, la siguiente figura es un diagrama de clase simple que muestra dos clases: Materia_prima y Movimientos_E_S (movimientos de entrada y salida) con una asociación entre ellos. Además, muestra cuántos objetos intervienen en la asociación, en este caso es 1:* lo que significa que 1 materia prima puede tener muchos movimientos de entrada y salida. Esto es denominado multiplicidad.

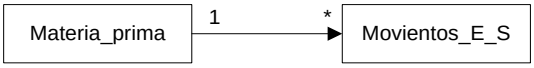


Diagrama de clases y asociación

Multiplicidad. La multiplicidad determina cuántos objetos de cada tipo intervienen en la relación, existen los siguientes.

1	Uno y solo uno

0..1	Cero o uno
X..Y	Desde X hasta Y
*	Muchos
0..*	Cero o muchos
1..*	Uno o muchos

- Cada línea de relación tiene dos multiplicidades (una para cada extremo de la relación).
- Para especificar hay que indicar la multiplicidad mínima y máxima (mínima...máxima).
- Cuando a multiplicidad es 0, la relación es opcional.
- Cuando es 1 indica que la relación es obligatoria.

Para especificar con mayor detalle las clases, se puede agregar las características de éstas. Por ejemplo, un objeto artículo tendrá atributos como un identificador, la descripción, existencias y costo unitario, sus operaciones podrían ser Alta, Baja, Consultar y Modificar principalmente. En UML los atributos y operaciones se muestran en un rectángulo con tres secciones. De esta manera se tiene que:

1. El nombre de la clase-objeto aparece en la parte superior.
2. Los atributos aparecen en la parte media y de manera opcional tienen el tipo de dato de cada atributo.
3. Las operaciones o comportamientos (llamadas métodos en algunos lenguajes de programación como Java) están en la sección inferior del rectángulo.

Algunas veces se encontrarán objetos que pertenecen a un mismo tipo y por lo tanto, pueden compartir atributos o comportamientos que les son comunes. Estos pueden ser asociados por medio de la relación llamada herencia.

Herencia (especialización/generalización). Indica que una clase hereda los métodos y atributos de la clase base, por lo cual una clase derivada además de tener sus propios métodos y atributos, podrá acceder a las características y atributos visibles de su clase base (public y protected).

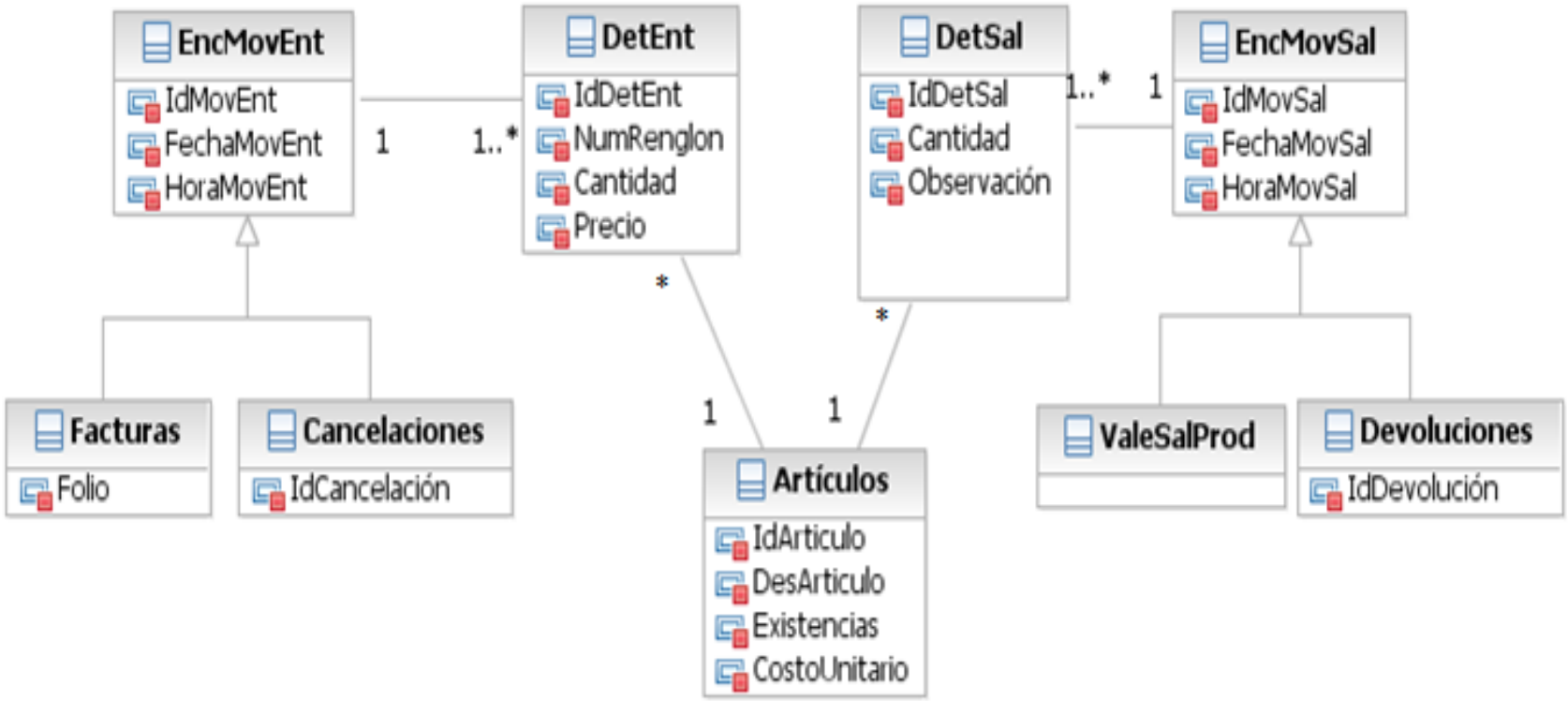


Diagrama de clases con propiedades y generalización

En seguida se muestran dos conceptos necesarios para comprender el diagrama que se encuentra posterior a éstos:

Composición. La composición es una relación en donde el tiempo de vida del objeto está condicionado por el tiempo de vida del que lo incluye. El objeto base depende del objeto que lo compone o sea es parte de un todo.

Agregación. La agregación es una relación donde el tiempo de vida del objeto incluido no depende del que lo incluye. El objeto base utiliza al objeto incluido para su funcionamiento.

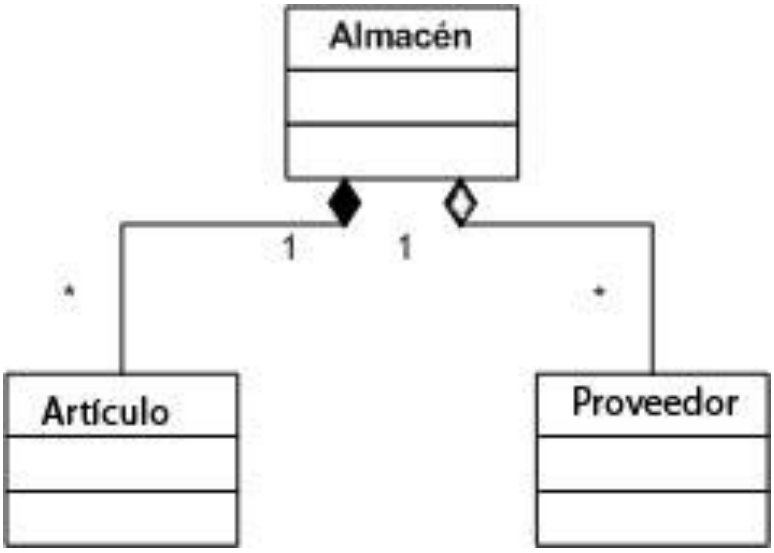


Diagrama de clases con composición y agregación

El anterior diagrama demuestra lo siguiente:

- Un Almacén se compone de 1 a muchos objetos tipo Artículo y se asocia con 0 o muchos objetos tipo Proveedor.
- Cuando se destruye la clase Almacén también es destruida la clase Artículo. En cambio no es afectada la clase Proveedor.
- La composición se identifica con un rombo relleno.
- La agregación se identifica con un rombo transparente.
- Los diagramas de clases son los más comunes dentro del modelado de sistemas orientados a objetos. Pero sirven para representar una vista estática del sistema.

Si se quiere demostrar cómo se da la comunicación e interacción dentro del sistema se tendrá que acudir a otro tipo de diagramas, por ejemplo, el diagrama de secuencia, que se verá a continuación.

2.2.2. Diagramas de secuencia

El diagrama de secuencia en UML es un diagrama de interacción donde destaca el envío de mensajes entre un conjunto de objetos. Los objetos pueden ser instancias con el nombre del objeto o bien, objetos anónimos. Estos diagramas se utilizan para describir la vista dinámica de un sistema. En otras palabras el diagrama de secuencia muestra las funcionalidades que se representan en un caso de uso. Los objetos y actores se muestran en la parte superior del diagrama, de estos se desprende una línea punteada vertical. Las interacciones entre objetos se indican con flechas dirigidas. El rectángulo sobre las líneas punteadas representa el ciclo de vida del objeto. La lectura es de arriba hacia abajo. Los mensajes llevan unas flechas que indican las llamadas a objetos, parámetros y valores que regresan. Las condiciones o alternativas se expresan dentro de corchetes.

Observa el siguiente diagrama de secuencia:

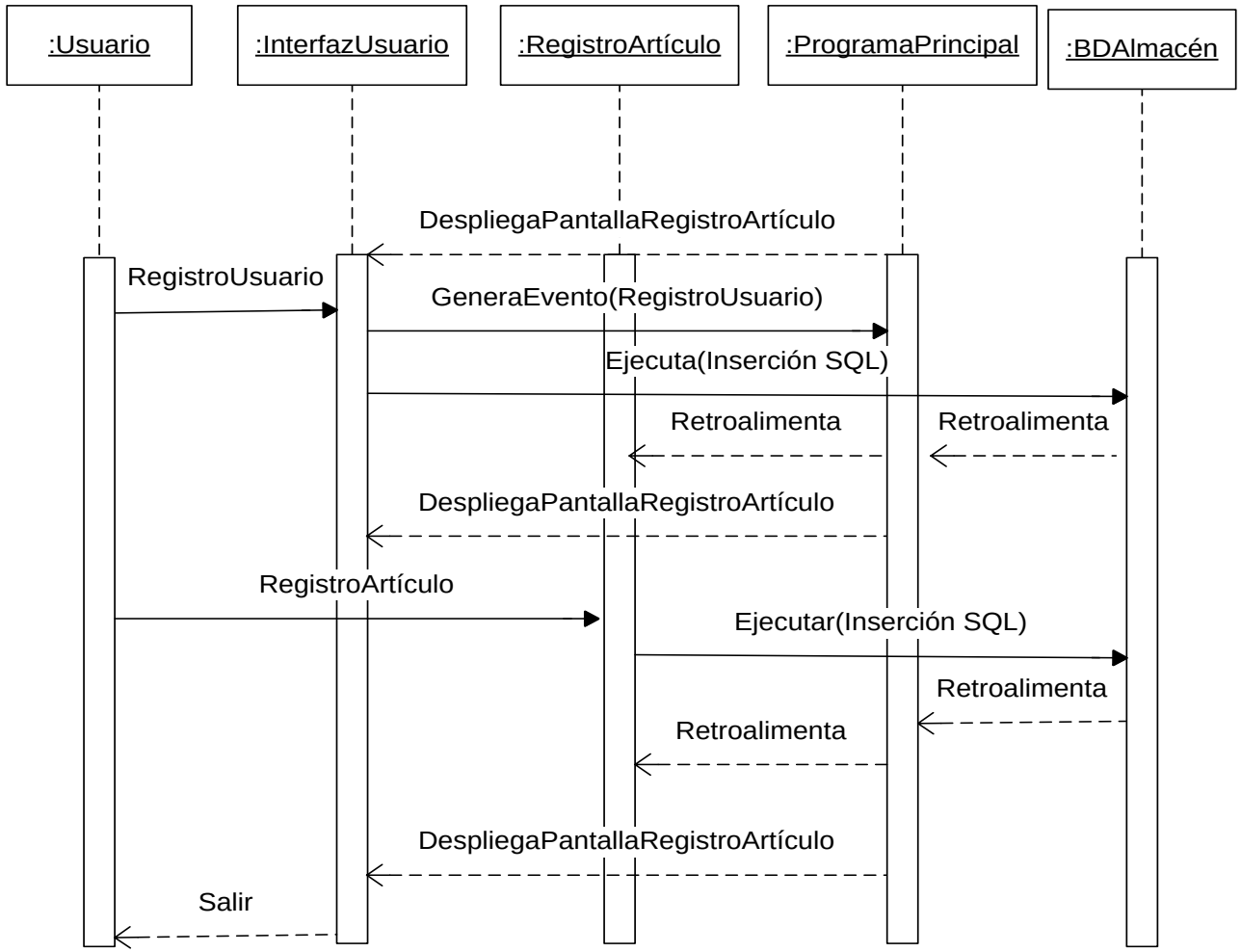


Diagrama de secuencia de alta de artículo del almacén de materia prima. Tomada de Booch, Rumbaugh, y Jacobson (1999, p. 83).

El diagrama de secuencia de la imagen anterior despliega una serie de mensajes entre los objetos usuario, InterfazUsuario, RegistroArtículo, ProgramaPrincipal, BDAlmacén. Comienza con el DespliguePantallaRegistroArtículo, cuando el usuario entra por primer ocasión debe registrarse en la InterfazUsuario, lo cual dispara el evento GeneraEvento(RegistroUsuario) en el ProgramaPrincipal, a su vez, ejecuta la inserción SQL en la BDAlmacén. Cuando realiza la consulta, el resultado de esta es regresado al ProgramaPrincipal y la InterfazUsuario.

Retorna el DespliegePantallaRegistroArtículo, el usuario introduce datos en el RegistroArtículo y se ejecuta la InserciónSQL en la BDAlmacén. La BDAlmacén, Retroalimenta al ProgramaPrincipal y al RegistroArticulo.

El diagrama de secuencia es un tipo de diagrama de interacción es decir, de comportamiento, porque demuestran los aspectos dinámicos del sistema. Otro tipo de diagrama con estas características es el diagrama de colaboración que se verá en el siguiente tema.

2.2.3. Diagramas de colaboración

Los diagramas de colaboración se utilizan para explicar una estructura entre los objetos que se envían mensajes. Los objetos generalmente son instancias con o sin nombres (anónimos), y se utilizan para describir una de las vistas del sistema. Estos diagramas son útiles para entender cuál es la mejor manera de desarrollar un sistema, permite a los diseñadores de software asegurarse de que el software que se desarrolla cubrirá con los requerimientos cuando sea implementado (Booch, Rumbaugh y Jacobson, 1999, p. 84).

A continuación se presenta un ejemplo de un diagrama de colaboración donde se describe el caso de uso del registro de un pedido. Desde el objeto **Ventana de registro de pedidos** se emite el mensaje **prepara()** para el objeto **Pedido**, este a su vez envía el mensaje **prepara()** condicionado para cada elemento del pedido**.** Por ejemplo, PVC pegamento es un ejemplo de una instancia del **elemento del pedido**, del cual se revisará si hay existencia y si hay existencia se aplica **descuenta()** y se prepara el **Artículo para**

entrega, si **no** se debe revisar si se **necesitaReordenar()**, en este caso se aplica el **Reorden de artículo**.

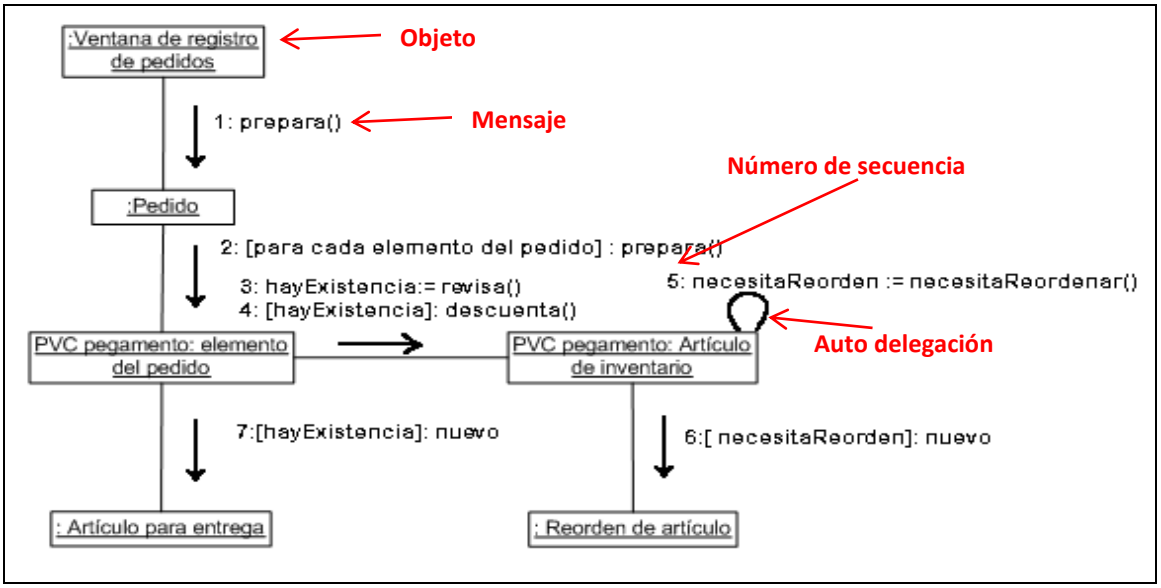


Diagrama de colaboración

En el diagrama anterior se pueden apreciar los componentes. Los rectángulos con el texto subrayado representan instancias de objetos que pueden tener nombre o no (anónimos). Si carecen de nombre se dejan los dos puntos [:]. Tienen líneas de transición que llevan un mensaje con una flecha que indica la dirección del mensaje. Los mensajes van numerados marcando el orden de la secuencia, esta numeración puede ser entera o decimal. Los objetos pueden enviarse mensajes ellos mismos, lo cual es llamado autodelegación y los mensajes pueden contener una condición.

Los aspectos dinámicos del software se representan por medio del flujo de mensajes principalmente como en la figura anterior. Otro tipo de diagramas de interacción son los diagramas de estado que se explican en el siguiente tema.

2.2.4. Diagramas de estado

Los diagramas de estados se aplican para mostrar una vista dinámica de un sistema. Son útiles para modelar el comportamiento de una interfaz o una clase. Se aplican ampliamente para modelar sistemas reactivos Booch, Rumbaugh y Jacobson (1999, p. 85).

Gráficamente un estado se representa por un rectángulo ovalado. Las transiciones mediante flechas con el nombre del evento respectivo. Para indicar el estado inicial se utiliza un círculo relleno. Y para el estado final se utiliza un círculo hueco que incluye un círculo relleno más pequeño en el interior. Las transiciones pueden llevar condiciones que limitan al estado hasta que la condición sea verdadera y estas condiciones van entre corchetes.

A continuación lee el fragmento de la descripción de un caso de uso “Devoluciones en el almacén” y en seguida observa cómo es representado en un diagrama de estado.

Descripción	Se registra la entrada al almacén si es que no hubo algún error. Cuando los artículos están dentro, se pueden detectar defectos en el mismo almacén o en la producción. El almacenista genera los documentos necesarios para devolver dicho artículo al proveedor.
Precondiciones	1. El proveedor debe presentarse con la factura.
Pos condiciones	1. Se registra la entrada en el sistema. 2. Se registra la devolución y número de nota de crédito en el sistema. 3. Se registra la cancelación en el sistema. 4. Se hace el cálculo inverso de los costos promedios por cada devolución.

Flujo Normal	1. El proveedor presenta los artículos solicitados con su factura respectiva. 2. El almacenista revisa que los artículos no tengan defectos (E1). 3. El almacenista registra la entrada de los artículos y los acomoda. 4. Personal del almacén detectan defecto en el artículo (E2). 5. Usuarios de la empresa detectan defecto en los artículos (E3). 6. El proveedor recibe el material devuelto y genera una nota de crédito.
Flujos Alternativos	A1: Se debe generar el cálculo inverso de los costos promedios por cada devolución.
Excepciones	E1: el almacenista explica al proveedor el defecto y rechaza el artículo. E2: el almacenista hace devolución y solicita nota de crédito. E3: el almacenista hace una cancelación (entrada), una devolución (salida) y solicita nota de crédito al proveedor.

Tabla de un fragmento de la descripción del caso de uso “Devoluciones en almacén”

En seguida observa cómo se representa en un diagrama de estado:

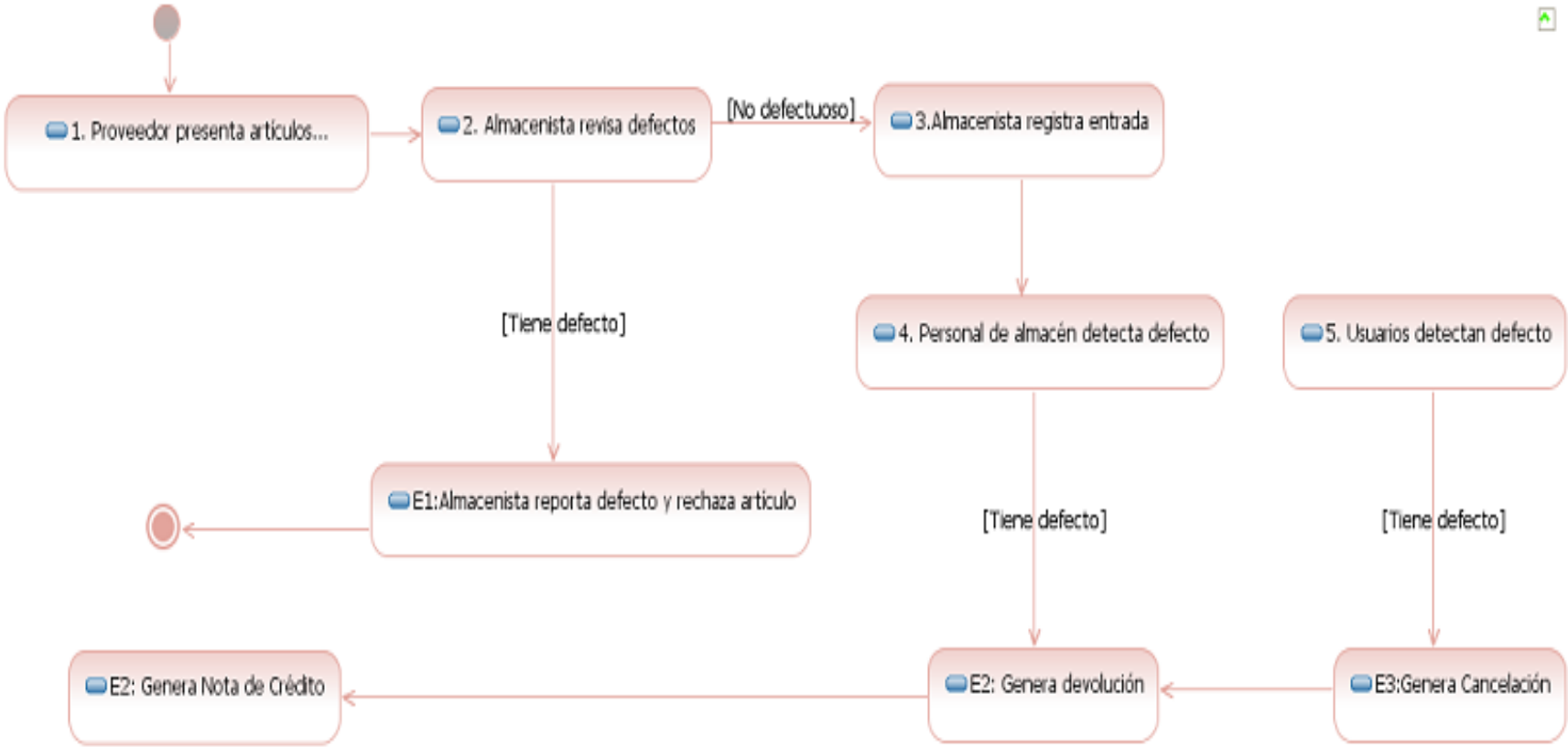


Diagrama de estados de devolución de un artículo al almacén

Todos los diagramas son útiles para representar diferentes vistas del modelo del software que se quiere construir, para poder lograrlo será necesario partir de una buena base: los requerimientos y como se ha visto durante esta unidad se deben definir adecuadamente.

Cierre de la unidad

Aprender a construir diagramas de modelado y saber interpretarlos forma parte de las habilidades que todo desarrollador de software debe tener, no importa el rol que desempeñe dentro de un equipo de de desarrollo de software. El líder, analista, diseñador, programador, encargado de pruebas, documentador, etc. deben entender este tipo de lenguaje que facilita la rápida comprensión de los requerimientos del software y su exposición desde diferentes vistas.

Todo este conocimiento te servirá para saber interpretar software documentado por terceros y también para aprender a documentar tus propios proyectos. Si formas parte de un equipo de desarrollo de software, el modelado es una parte importante de la traducción e interpretación de los requerimientos, por lo tanto, para una óptima comunicación entre los miembros de un equipo serán necesarios estos conocimientos.

Para saber más

Para complementar más acerca del tema de requerimientos. La siguiente propuesta es un estudio de la ingeniería de requerimientos que abarca retos futuros como lo son la escala y la agilidad.

- Cheng, B.H.C. y Atlee, J. M. (2007). *Research the Requeriments Process* (2ª ed.). Michigan: Michigan State University. Recuperado de http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4221627

Para buscar más ejemplos y otra forma de describir los diagramas vistos en clase, o bien, para conocer los otros diagramas que no se abarcaron en el programa de la asignatura puedes consultar el siguiente manual de UML.

- Programación C.E.C.yT. “Juan de Dios Bátiz Paredes” – IPN. (n.d.). *Desarrollo orientado a objetos con UML*. http://es.scribd.com/doc/2458870/Desarrollo-Orientado-a-Objetos-con-UML-librobookespanolspanish

Fuentes de consulta

- Booch, G., Rumbaugh J. y Jacobson, I. (1999). *El lenguaje unificado de modelado*. Madrid: Addison Wesley.
- Piattini, M., Calvo M., Cervera J. y Fernández, L. (2004). *Análisis y diseño de aplicaciones informáticas de gestión. Una perspectiva de ingeniería de software.* México: Alfaomega/Ra-Ma.
- Sommerville, I. (2011). *Ingeniería de software*. México: Pearson Educación.