Capítulo 1
Marco teórico

1.1 Ingeniería de software

La ingeniería de software es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software, desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste después de que se utiliza (Sommerville, 2005:6).

Actualmente, existen varias metodologías para abordar un proyecto de software que han sido generadas a partir de mejoras continuas a lo largo de los años para que el proceso de desarrollo de software se adapte a las necesidades actuales.

La finalidad de apegarse a una metodología o proceso de desarrollo de software es determinar el orden de las etapas involucradas en el desarrollo y evolución del software y establecer los criterios de transición para el progreso de una etapa a otra. Incluye también un criterio de culminación de la etapa actual más los criterios de elección y entrada para la siguiente etapa.

Las metodologías de desarrollo de software son importantes, principalmente porque proporcionan una guía del orden (fases, incrementos, prototipos, validación de tareas, etc.) que un proyecto debe seguir para realizar sus tareas más importantes. Muchos proyectos de software han enfrentado dificultades porque abordan varias fases del desarrollo y evolución de la manera equivocada.

1.2 Evolución de las metodologías de software

A continuación se muestra una breve descripción de algunas metodologías de software existentes que forman parte de la evolución de las mismas.

1.2.1 Codificar- Corregir

Esta metodología básicamente consiste de dos pasos:

- Escribir porciones de código
- Resolver los problemas en el código.

Las etapas de diseño, pruebas y mantenimiento eran consideradas después. A lo largo de su utilización se identificaron varios problemas que dificultaban un desarrollo fluido y exitoso. En realidad, era prácticamente inevitable tener errores en el producto desarrollado que debían ser corregidos antes de su entrega. Algunos ejemplos se enuncian a continuación:

- Después de varias correcciones el código perdía su estructura
- Las expectativas del cliente no se satisfacían, por lo que se rechazaba el producto y se aplicaba mayor esfuerzo del necesario
- Los costos de corregir el código eran muy altos, por la poca preparación de las pruebas

1.2.2 Modelo en cascada

Para resolver los problemas que se presentaban en desarrollos de sistemas en los años 50´s se dio reconocimiento a los problemas y se desarrolló un modelo de etapas. La idea detrás de este modelo estipulaba que el software debía ser desarrollado en varias etapas sucesivas (plan operacional, especificaciones operacionales, especificaciones de código, codificación, parámetros de pruebas, pruebas de ensamblado y evaluación del sistema). Finalmente, en los años 70´s se tenía ya establecido el modelo de cascada que ofrecía las siguientes mejoras:

- Reconocimiento de los ciclos de retroalimentación entre las etapas, y una guía para limitar los mismos entre etapas sucesivas para minimizar trabajo duplicado o innecesario y costoso involucrado en la retroalimentación en diversas etapas.

- Mejor organización al momento de desarrollar un producto de software, pues la definición de sus etapas ayuda a generar planes de trabajo más específicos.

- Resulta sencillo para los usuarios comprender cada una de las fases del modelo, lo cual mejora la comunicación con el equipo de desarrollo del sistema.

El modelo en cascada se convirtió en la base de varios estándares de software en el sector público y privado. Algunas de sus dificultades iniciales han sido atendidas extendiendo el modelo a enfoques ligeramente mas flexibles en cuanto a las transiciones de las etapas que define. Sin embargo, ciertas dificultades no se han podido resolver mediante este modelo. La principal dificultad ha sido su énfasis en documentos completamente elaborados para las etapas iniciales de requerimientos y diseño. Para ciertos tipos de software tales como compiladores o sistemas operativos este modelo funciona muy bien. No obstante, para otras clases de software, particularmente para aplicaciones que ofrecen constante interacción con el usuario final, esta rigidez es poco viable ya que los requerimientos del producto tienden a cambiar constantemente.

1.2.3 Modelo evolutivo

Las etapas del modelo evolutivo consisten en expandir incrementos de un producto operacional de software, con las directivas de evolución siendo determinadas por la experiencia operacional.

El modelo evolutivo está idealmente relacionado a una aplicación de lenguaje de cuarta generación y se apega a situaciones donde el usuario menciona 'no puedo decirte qué es lo que quiero, pero lo sabré cuando lo vea'. Provee una ágil capacidad operacional inicial y establece las bases para determinar mejoras subsecuentes al producto. Pero este modelo también sufre ciertas dificultades. Generalmente es difícil distinguirlo contra el modelo de codificar y corregir en él que la falta de planeación fue la motivación inicial del surgimiento del modelo en cascada. Regularmente se basa en la suposición poco realista de que el sistema será lo suficientemente flexible para agregar rutas evolutivas no planeadas. Este supuesto es inválido en las siguientes condiciones:

- Circunstancias en las que varias aplicaciones evolutivas independientes deben ser al final integradas estrechamente

- Casos en que correcciones de software temporales se materializan y transforman en restricciones que ya no se pueden modificar en el proceso evolutivo.

- Cuando el nuevo software está incrementalmente reemplazando a un software existente que carece de organización y modularidad que permita hacer que cada parte del sistema evolucione.

1.2.4 El modelo de transformación

Este modelo supone la existencia de la capacidad de convertir automáticamente una especificación formal de un producto de software en un programa que satisfaga esa especificación. Por tanto, los pasos que se establecen en este modelo son los siguientes:

- Tener una especificación formal derivada del mejor entendimiento del producto deseado.

- Transformación automática de la especificación a código
- Un ciclo iterativo, de ser necesario, para mejorar el desempeño del código resultante mediante la optimización del sistema en transformación.
- Ejercicio del código resultante.
- Un ciclo iterativo independiente para ajustar la especificación basada en la experiencia operativa y optimizar el sistema de software.

1.2.5 El modelo en espiral

El modelo en espiral, originalmente propuesto por Bohem, es un modelo evolutivo de proceso de software que combina la naturaleza iterativa de los prototipos con los aspectos controlados y sistemáticos del modelo en cascada. Principalmente está intencionado para su aplicación en proyectos largos, complicados y costosos. Provee el potencial de un desarrollo rápido de versiones incrementales de software. Al utilizar el modelo en espiral, el software es desarrollado en series de liberaciones incrementales. Durante las iteraciones iniciales, la liberación incremental puede ser un modelo en papel o un prototipo y va creciendo hasta cubrir las funcionalidades requeridas.

Asimismo, el modelo en espiral puede reunir prácticas de modelos anteriores y proveer una guía mediante la combinación de las mejores prácticas de las mismas.

Este modelo ha evolucionado a través de los años, basado en experiencias para perfeccionar el modelo en cascada en casos tales como grandes proyectos de software. La Figura 1.1 representa un diagrama del modelo en espiral. Como se muestra en la figura, la dimensión radial de la espiral representa el costo acumulado para terminar los pasos del desarrollo. La dimensión angular representa el progreso para terminar cada ciclo de la espiral.

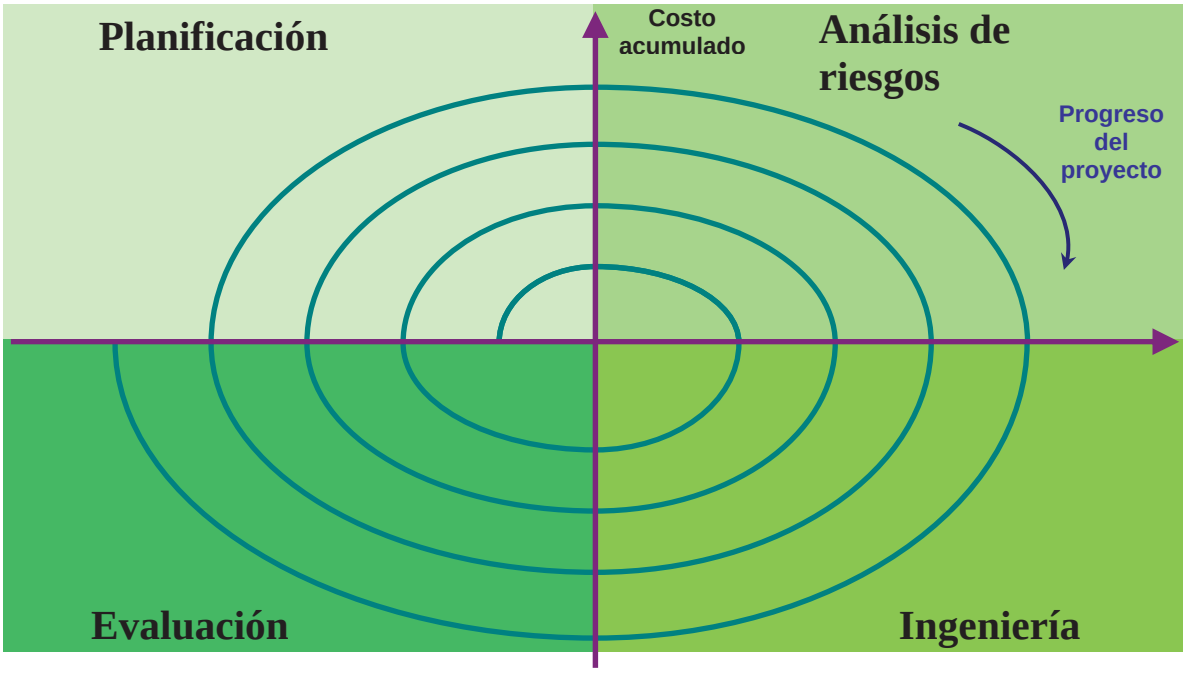

Figura 1.1 Diagrama del modelo en espiral

Cada una de las regiones está compuesta por un conjunto de tareas, las cuales son adaptadas a las características del proyecto en cuestión. Para proyectos pequeños el número de tareas es menor y por lo tanto no requieren demasiada formalidad. En contraste, refiriéndonos a proyectos grandes y críticos, cada región del modelo contiene más tareas que son definidas para lograr un mayor grado de formalidad y detalle.

Conforme el proceso evolutivo inicia, el equipo involucrado en la ingeniería de software se mueve de manera espiral en sentido a las manecillas del reloj, iniciando cerca del centro. El primer circuito de la espiral puede resultar en el desarrollo de la especificación del producto, los circuitos subsecuentes pueden ser usados para desarrollar un prototipo, para entonces tener versiones más sofisticadas del producto de software de manera progresiva. Cada paso a través de la región de planeación para determinar los objetivos, alternativas y restricciones resulta en ajustes en el plan del proyecto, pues las acciones a tomar para atender los riesgos o situaciones particulares repercuten en el plan inicial y es necesario

revisar constantemente el plan del proyecto. Tanto el calendario como el costo son modificados, dependiendo de la retroalimentación obtenida con el cliente. Adicionalmente el líder de proyecto ajusta el número planeado de iteraciones requeridas para que el producto esté terminado.

1.2.6 Un ciclo típico de la espiral

Cada ciclo de la espiral inicia con la identificación de los siguientes elementos:

- Los objetivos de la porción del producto que está siendo elaborada (desempeño, funcionalidad, alcance, etc.)
- Las alternativas que existen para la implementación de esta porción del producto
- Las restricciones que tiene cada alternativa para que sea aplicada (costo, calendario, recursos, etc.)

El siguiente paso consiste en evaluar las alternativas en cuanto a los objetivos y restricciones. Frecuentemente, este proceso identificará áreas de incertidumbre que son fuentes potenciales de riesgo.

Posteriormente, la siguiente etapa consistirá en la formulación de una estrategia efectiva en costo para implementar o desarrollar los objetivos definidos. Asimismo, se tiene que prestar especial atención a las actividades que atiendan las fuentes de riesgo identificadas en el paso anterior. Algunas técnicas que pueden ser utilizadas incluyen simulaciones, prototipos, benchmarking, cuestionarios, etc.

Una vez que se evaluaron los riesgos y desarrollaron los elementos necesarios, el siguiente paso está determinado por los riesgos que no se hayan podido atender en el paso anterior.

Para ello, se analizan y planean las siguientes fases que serán parte de una nueva iteración en la espiral.

Durante las iteraciones posteriores, versiones más completas del sistema son producidas. El modelo en espiral está dividido en un determinado número de actividades del esquema, dependiendo el detalle que se quiera dar a la espiral. Algunas de las actividades más comunes son:

- Planificación: Actividades requeridas para definir recursos, fechas de entrega y otra información relacionada con el proyecto.
- Análisis de riesgo: Actividades necesarias para evaluar riesgos tanto técnicos como administrativos del proyecto
- Ingeniería: Actividades necesarias para construir una o más representaciones de la aplicación
- Evaluación del cliente: Actividades necesarias para obtener la retroalimentación del cliente basada en la evaluación de las representaciones de software creadas e implementadas durante la etapa de ingeniería.

Hay 4 preguntas que surgen a causa del modelo en espiral.

1) ¿Cómo se empieza la espiral?
2) ¿Cómo salir de la espiral cuando es apropiado terminar tempranamente el proyecto?
3) ¿Por qué la espiral termina tan abruptamente?
4) ¿Qué sucede con la mejora del software (o mantenimiento)?

Las respuestas a estas preguntas involucran hacer la observación de que el modelo en espiral es aplicable tanto a desarrollos como a mejoras de software. En cualquiera de los

casos, la espiral es iniciada bajo la hipótesis de que una misión operacional en particular (o grupo de misiones operacionales) pueden ser mejoradas mediante un esfuerzo orientado al software. Entonces, el modelo en espiral involucra probar esta hipótesis: en cualquier momento, si la hipótesis falla la prueba (por ejemplo, si los retrasos causan que un producto de software pierda su periodo útil, o si un producto comercial superior es lanzado), el modelo en espiral es terminado. Por otro lado, termina con la instalación de un producto de software, ya sea nuevo o modificado, y la hipótesis es probada mediante la observación del efecto producido en la misión operacional. Usualmente, la experiencia con la misión operacional resulta en otra hipótesis relacionada a mejoras del software y una nueva espiral de mantenimiento es iniciada para probar la hipótesis. La iniciación, terminación e iteración de las tareas y productos de ciclos previos son implícitamente definidos en el modelo en espiral.

A diferencia de los modelos de proceso clásicos que terminan cuando el software es entregado, el modelo en espiral puede ser adaptado para su aplicación a lo largo del ciclo de vida del producto de software. El centro del modelo en espiral puede ser usado para representar el punto inicial para diferentes tipos de proyecto. Un “proyecto de desarrollo conceptual” inicia cerca del centro de la espiral y continuará hasta que el proyecto conceptual sea completado. Si el concepto debe ser desarrollado en un producto real, el proceso continúa al siguiente ciclo (nuevo punto de entrada para el proyecto de desarrollo del producto) y un “nuevo proyecto de desarrollo” es iniciado. El nuevo producto evolucionará a través de un número de iteraciones alrededor de la espiral, siguiendo la ruta que limita la región que de alguna manera es más clara que el centro. En esencia, la espiral, cuando es caracterizada de esta manera, permanece en operación hasta que el producto es

retirado. Hay veces que el proceso permanece inactivo, pero en el momento en que un cambio es iniciado, el proceso es iniciado en el punto de entrada apropiado (por ejemplo, cuando se necesita una mejora).

El modelo en espiral es un enfoque realista para desarrollar sistemas de gran escala. Dado que el software evoluciona conforme el proceso avanza, el desarrollador y el cliente tienen un mejor entendimiento del proyecto y toman acciones ante riesgos en cada nivel de la evolución del sistema. El modelo en espiral utiliza los prototipos como un mecanismo para reducir riesgos pero, más importante, para permitir al desarrollador aplicar el acercamiento prototipado en cualquier etapa de la evaluación del producto. Mantiene el enfoque sistemático sugerido en el ciclo de vida clásico, pero lo incorpora en un esquema iterativo que refleja de mejor manera el mundo real. El modelo en espiral demanda consideraciones de riesgos técnicos en todas las etapas del proyecto, que al ser aplicadas apropiadamente, deberá reducir riesgos antes de que se vuelvan problemas.

Pero como en otros paradigmas, el modelo en espiral no es una panacea. Puede ser complicado convencer al cliente (particularmente en situaciones de contratos) que el enfoque evolutivo es controlable. Demanda amplia experiencia en la evaluación de riesgos y depende de la misma para lograr el éxito. Finalmente, el modelo no ha sido utilizado tanto como los modelos lineales secuenciales o de prototipos. Tomará varios años antes de que la eficacia de este importante paradigma pueda ser determinada con absoluta certeza.

1.3 Mantenimiento de software

1.3.1 Definición y relevancia del mantenimiento de software

Una vez que el sistema se ha instalado se necesitan acciones para que el funcionamiento del mismo sea como se espera. El mantenimiento involucra tanto procesos de mejora y

optimización, como acciones correctivas para eliminar defectos del sistema. Es una de las consideraciones más relevantes de un proyecto de software, pues una vez que el mismo ha sido entregado se deben contemplar las modificaciones necesarias para corregir errores, mejorar el desempeño u otros atributos o adaptar el producto a nuevas necesidades y demandas del usuario.

Mantenimiento correctivo. Incluso cuando las mejores metodologías de calidad son utilizadas en el desarrollo de software no estamos exentos de tener algún error después de la instalación del sistema. En el mantenimiento correctivo se hacen modificaciones al software para corregir defectos que no fueron identificados durante el desarrollo del sistema y que el funcionamiento sea el que se espera.

Mantenimiento adaptativo. Durante el ciclo de vida del sistema es común que elementos del entorno (sistema operativo, reglas de negocio, hardware, etc.) cambien. El mantenimiento adaptativo se refiere a las actividades necesarias para que el sistema se adapte a esos cambios.

Mantenimiento perfectivo (mejoras). Conforme el sistema es utilizado por los usuarios es probable que se identifiquen nuevas funcionalidades que pueden ser incorporadas al sistema para que los beneficios del producto de software sean mayores. El mantenimiento perfectivo se encarga de extender el sistema a nuevas capacidades de acuerdo a la especificación del cliente.

Mantenimiento preventivo**.** Los productos de software tienden a deteriorarse después de que se hacen varias modificaciones probablemente derivadas por los tipos de mantenimiento anteriormente mencionados. Por lo tanto, se necesitan tomar acciones para reducir riesgos de falla en el futuro. El mantenimiento preventivo se encarga de modificar el sistema para que sea más fácil corregirlo, adaptarlo o mejorarlo.

1.3.2 Los procesos de mantenimiento de software

El mantenimiento de software involucra varias actividades a lo largo de la etapa operativa del producto de software, las cuales se agrupan en 6 procesos descritos a continuación.

1.3.2.1 Implementación del proceso

Para realizar la implementación del proceso de mantenimiento de software, el equipo que brindará el mantenimiento del producto genera el plan y los procedimientos a ser ejecutados durante el proceso de mantenimiento de software. Es importante resaltar que el plan de mantenimiento y el plan de desarrollo de software son generados de manera paralela. Asimismo, se establecen las formas de comunicación que se utilizarán con el usuario cuando el mantenimiento sea requerido. Por otra parte, el equipo de mantenimiento debe contar con toda la documentación asociada al sistema, la cual fungirá como la base de conocimiento técnico y funcional, necesario para brindar un mantenimiento apropiado.

La organización y el equipo de mantenimiento en conjunto definen el concepto de mantenimiento, para así tener un claro alcance del mismo.

Entre las tareas que el equipo de mantenimiento debe realizar se encuentran:

- Desarrollar los planes y procedimientos de mantenimiento
- Establecer procedimientos para peticiones de cambio del sistema

- Establecer procedimientos para reportar problemas
- Implementar la gestión de la configuración del sistema
- Estimar costos de mantenimiento

1.3.2.2 Análisis del problema y modificaciones

Este análisis se orienta a entender el problema, desarrollar una solución y obtener la aprobación para poder desarrollarla. Para ello se llevan a cabo las siguientes actividades:

- Analizar el problema encontrado o modificación solicitada para conocer el impacto, tanto en la organización, como en el sistema existente. En esta actividad se definen características, tales como el tipo de cambio, alcance y criticidad.
- Replicar el problema, ya sea con ayuda del usuario o por cuenta propia del equipo de mantenimiento. Implica contar con una estrategia de pruebas para replicar el problema, tener un ambiente con la misma versión de software y documentar los resultados.
- Generar y documentar opciones para corregir el problema o implementar la modificación solicitada, incluyendo estimaciones y riesgos por cada una de las opciones.
- Obtener la aprobación para la opción elegida por el cliente. De esta manera se tendrá definida la solución específica elegida por el cliente.

1.3.2.3 Implementación de la modificación

Durante este proceso, el equipo de mantenimiento y soporte de la aplicación desarrolla y prueba la corrección o modificación siguiendo la opción aprobada por el cliente apoyándose en la documentación de la misma. Se realizan las siguientes actividades:

- Actualización de los planes y procedimientos de pruebas
- Actualización de la documentación del sistema
- Modificación del código fuente
- Se genera un informe de resultados

1.3.2.4 Revisión y aceptación de la modificación

La revisión y aceptación de la modificación se utiliza para asegurar que las modificaciones se hicieron de forma correcta y cubrieron la necesidad o problema reportados. Entre las actividades para la revisión del mantenimiento se encuentran:

- Verificar que se cumplen los estándares de codificación
- Verificar que sólo se modificaron los componentes necesarios
- Verificar que la integración de los componentes modificados con el resto de los componentes sea correcta
- Comprobar que la documentación fue actualizada
- Ejecutar pruebas y documentar los resultados de las mismas

Una vez que la revisión ha terminado el mantenedor deberá obtener la aprobación para terminar satisfactoriamente el mantenimiento.

1.3.2.5 Migración

El proceso de migración es aplicado cuando se necesita que el sistema sea ejecutado en entornos diferentes, por tanto, el mantenedor necesita determinar las acciones necesarias para llevarla a cabo y generar la documentación relacionada. De esta manera se crea un plan de migración con las siguientes actividades:

- Análisis de requerimientos y definición de la migración

- Identificación de impacto, riesgos y esfuerzo de la migración
- Desarrollo de herramientas necesarias para efectuar la migración
- Conversión de datos y productos de software
- Ejecución de la migración
- Verificación de la migración por medio de pruebas
- Soporte para el entorno previo

1.3.2.6 Retiro del producto

El retiro del producto de software se lleva a cabo una vez que su periodo útil ha finalizado y conlleva a un nuevo análisis con enfoque costo-beneficio para elegir alguna de las siguientes opciones:

- Conservar el software obsoleto
- Cambiar de tecnología a través de un nuevo producto de software
- Desarrollar un nuevo producto de software

Después de tomar la decisión, se genera un plan de retiro con las siguientes actividades:

- Cese total o parcial del soporte del software
- Archivar el software y datos
- Definir la transición al nuevo producto de software
- Determinar el impacto del retiro del software
- Retirar el software

En la figura 1.2 se muestra la representación gráfica de los procesos de mantenimiento de software anteriormente descritos.

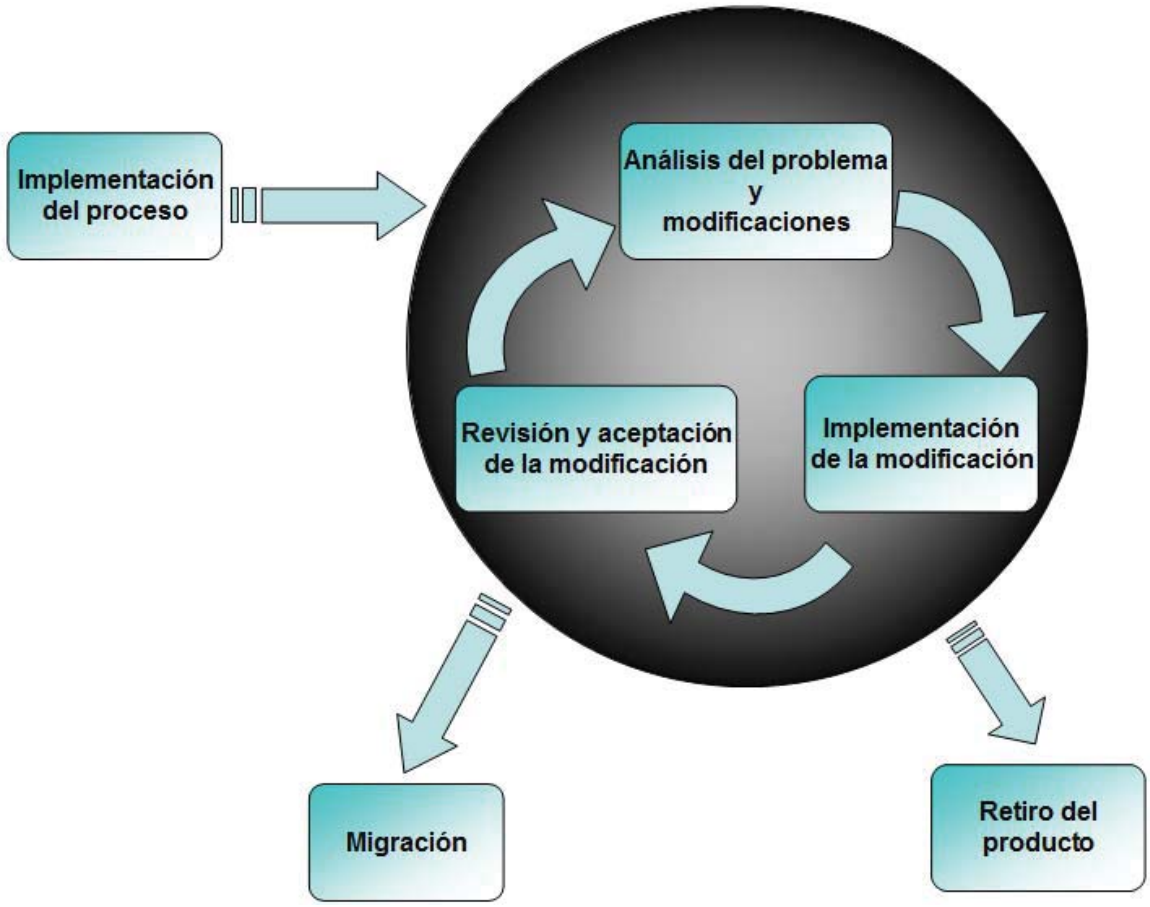

Figura 1.2 Procesos del mantenimiento de software

1.4 UML

Describimos de manera general el término UML (Unified Modeling Language) como un lenguaje utilizado para visualizar, especificar, construir y documentar los artefactos de un sistema de software. Se trata de un lenguaje gráfico orientado a apoyar todas las fases de desarrollo de software, desde la especificación y análisis de requerimientos hasta la construcción e instalación de los mismos.

La idea central detrás de la utilización de UML es capturar los detalles significativos de un sistema, de manera que el problema sea claramente comprendido, se desarrolle una solución al respecto y su implementación se identifique y construya de manera clara.

El lenguaje UML no sólo proporciona una notación para los elementos de construcción del sistema, también permite expresar relaciones complejas entre dichos elementos. Una relación puede ser estática, si se trata de conceptos de herencia entre un par de clases, interfaces implementadas o dependencias con otra clase. Por otro lado, existen relaciones dinámicas cuando describen el comportamiento del sistema, por ejemplo, cuando representan los intercambios de mensajes entre clases o el flujo de control del sistema.

A continuación se enlistan los tipos de diagramas proporcionados por el lenguaje UML.

- Diagrama de casos de uso. Son utilizados para representar gráficamente un comportamiento, mediante una secuencia de interacciones entre los actores y las funciones de un sistema. Debe capturar de manera precisa los requerimientos desde la perspectiva del usuario. En la figura 1.3 se muestra un ejemplo correspondiente a este tipo de diagramas.

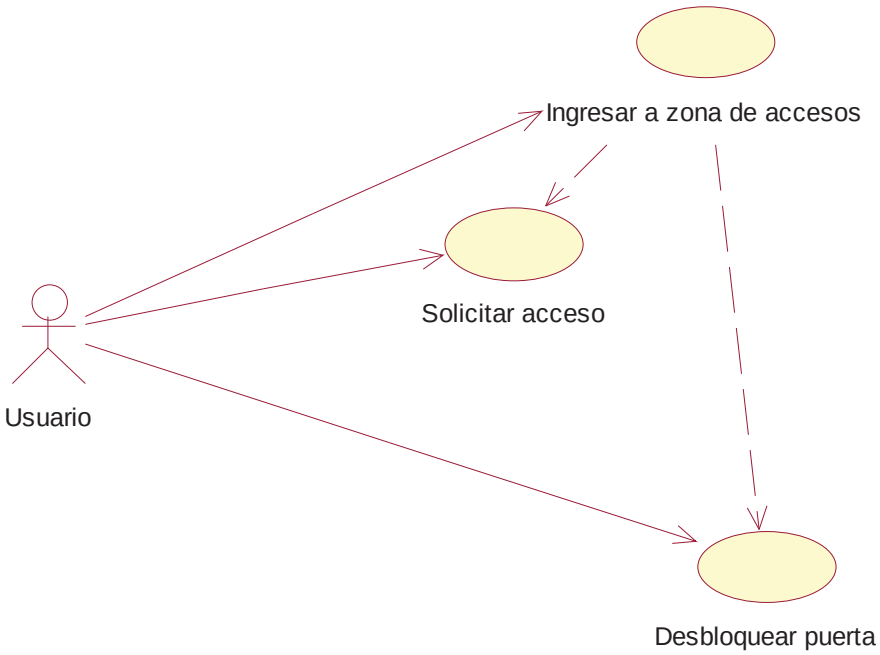

Figura 1.3 Ejemplo de diagrama de caso de uso

- Diagrama de clases. Muestra las relaciones estáticas existentes entre un grupo dado de clases e interfaces del sistema. Las relaciones más comunes representadas con este diagrama son herencia y dependencia de clases, tal y como se muestra en la figura 1.4.

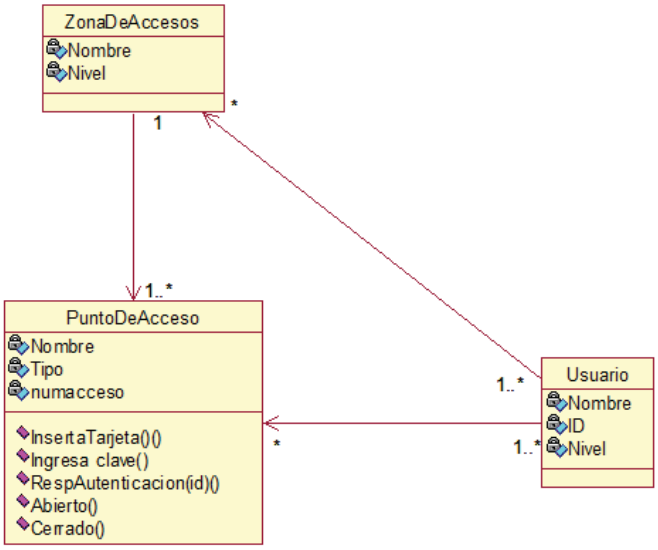

Figura 1.4 Ejemplo de diagrama de clases

- Diagrama de objetos. Brinda una vista instantánea de las relaciones que existen entre las instancias de clase en un determinado momento. Es de utilidad para capturar e ilustrar, de forma estática, relaciones complejas y dinámicas dentro del sistema como se aprecia en la figura 1.5.

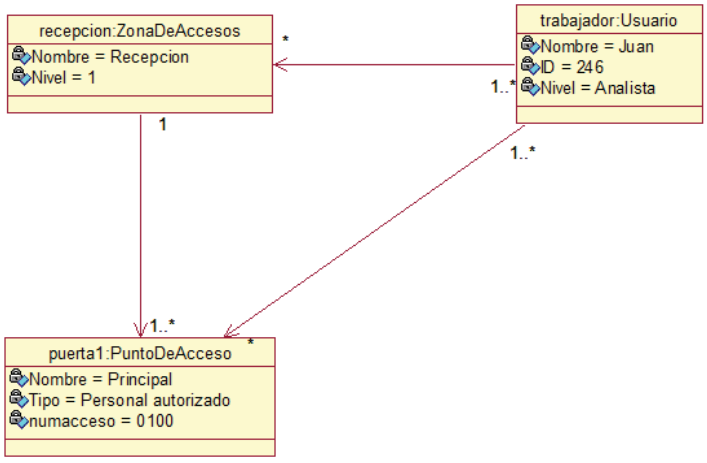

Figura 1.5 Ejemplo de diagrama de objetos

- Diagrama de estados. Excelente para ejemplificar el comportamiento dinámico de un sistema. Son aplicables a sistemas u objetos reactivos u orientados a eventos donde el orden de ejecución es importante como se ilustra en la figura 1.6.

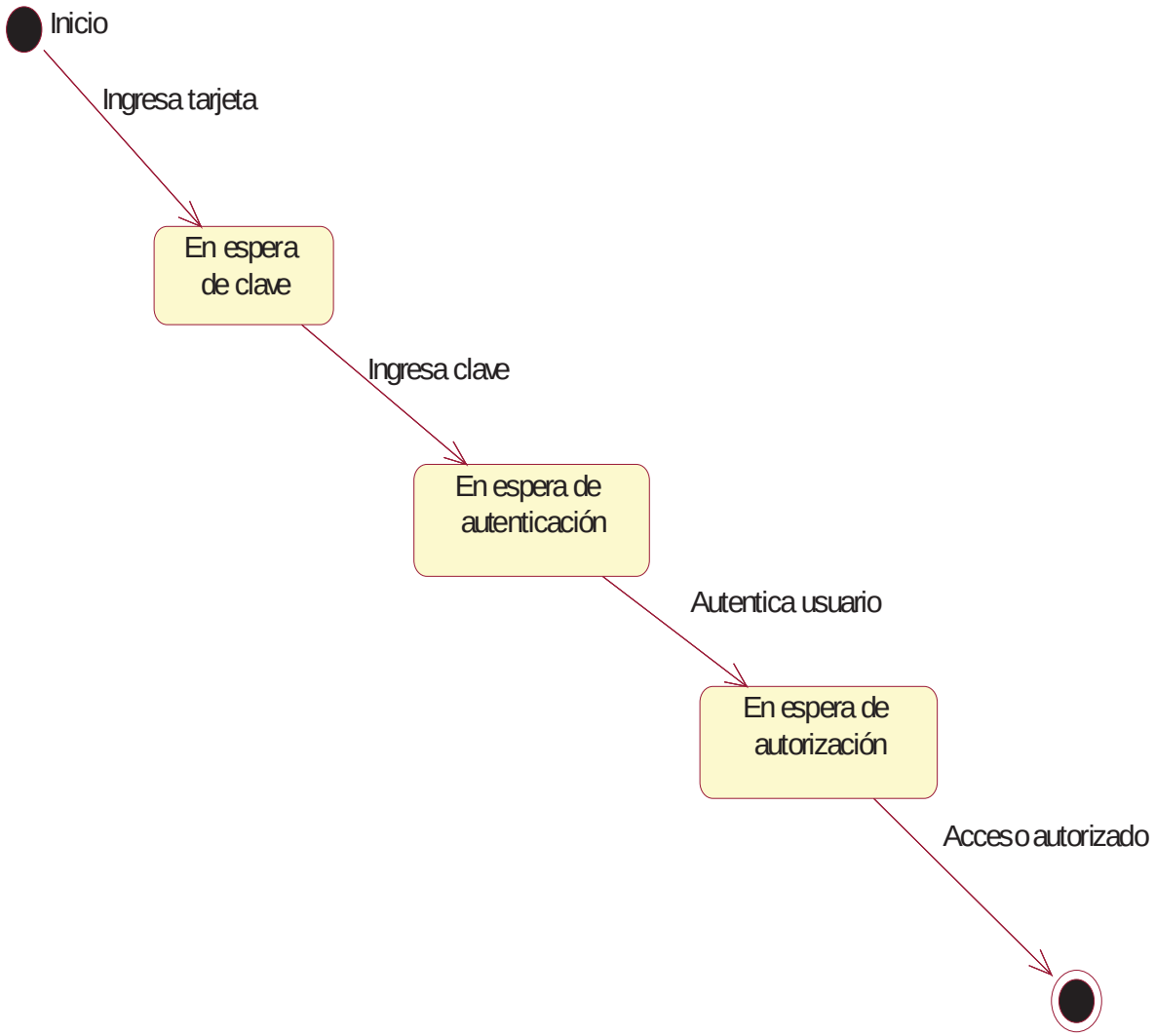

Figura 1.6 Ejemplo de diagrama de estados

- Diagrama de actividades. Son una extensión del diagrama de estados, pero orientado a flujos de trabajo dentro del sistema desde el punto de vista funcional. La figura 1.7 ejemplifica este tipo de diagramas.

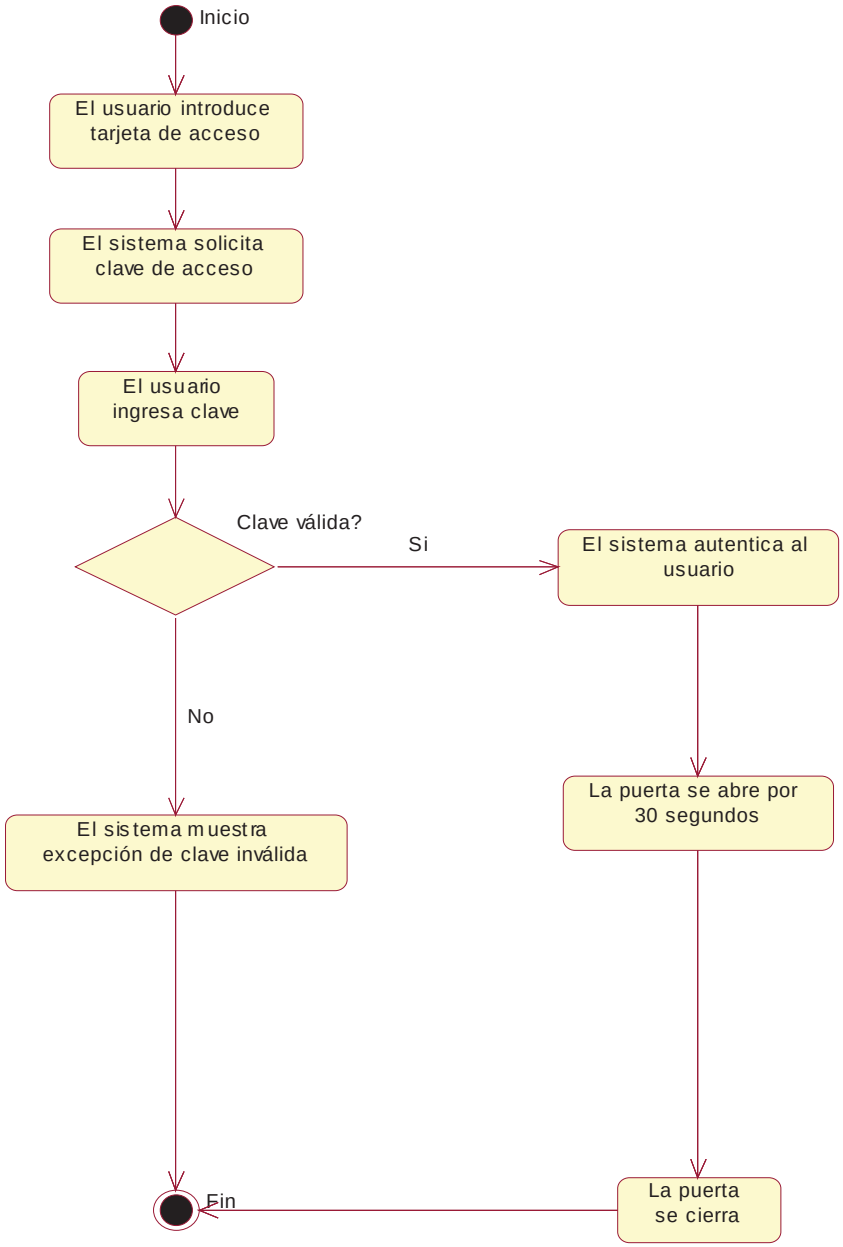

Figura 1.7 Ejemplo de diagrama de actividades

- Diagrama de secuencias. Son utilizados para modelar el intercambio de mensajes entre los objetos del sistema. También clarifican el tiempo relativo de los mensajes. La figura 1.8 ejemplifica un diagrama de secuencias para un sistema de control de accesos.

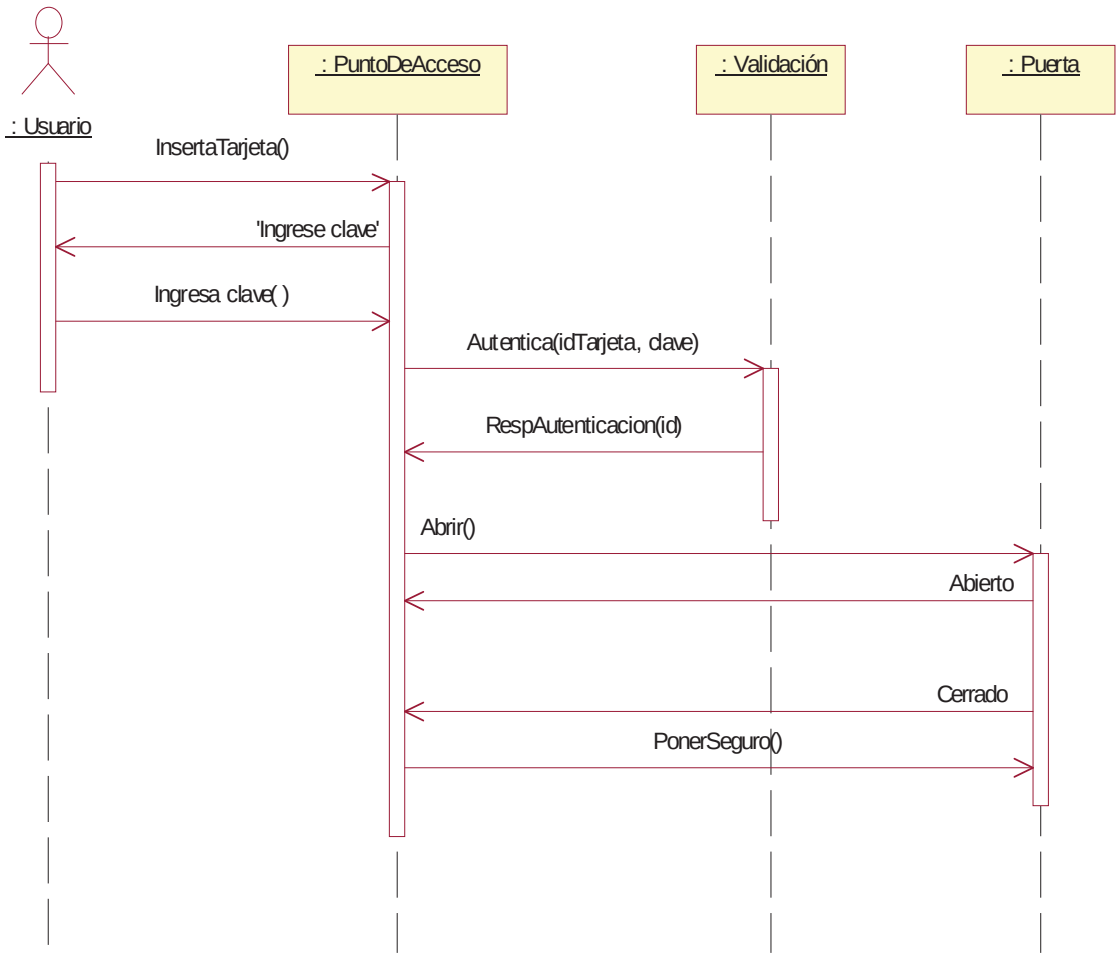

Figura 1.8 Ejemplo de diagrama de secuencias

- Diagrama de colaboración. Registran el intercambio de mensajes en el contexto de las relaciones estructurales generales de los objetos como se muestra en la figura 1.9.

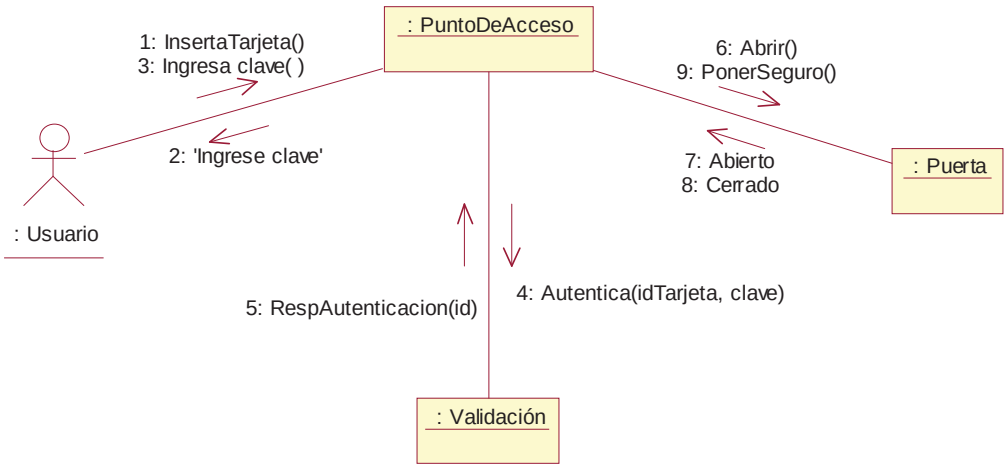

Figura 1.9 Ejemplo de diagrama de colaboración

- Diagrama de componentes. Representan la manifestación física de una parte del sistema, como algún archivo o ejecutable. Típicamente un componente se mapea a una o más clases o subsistemas como se ilustra en la figura 1.10.

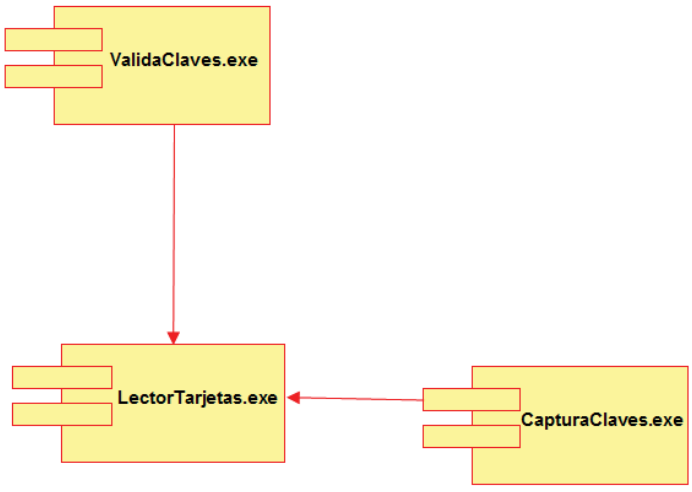

Figura 1.10 Ejemplo de diagrama de componentes

- Diagrama de despliegue. Muestra la arquitectura del sistema desde la perspectiva de los nodos y procesadores, así como también las relaciones entre ellos. Normalmente uno o más componentes se asocian a un nodo de despliegue. Son útiles para modelar y desarrollar arquitecturas de sistemas distribuidos. La figura 1.11 muestra un ejemplo de este tipo de diagramas.

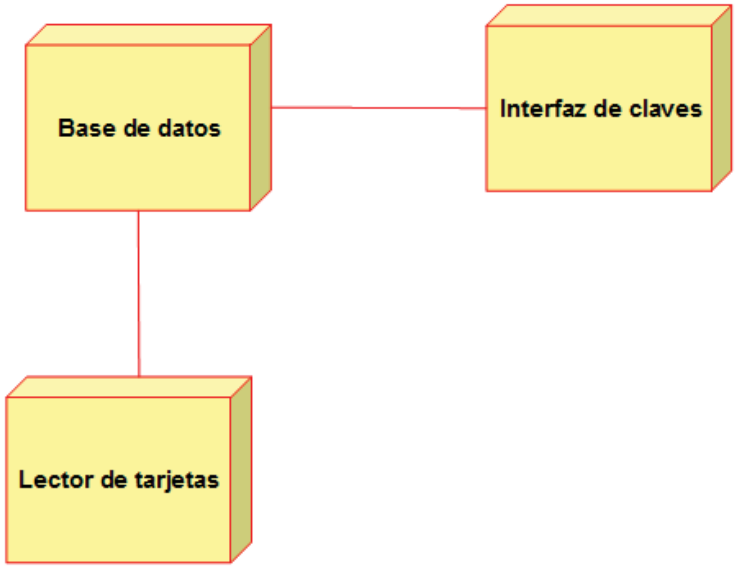

Figura 1.11 Ejemplo de diagrama de despliegue

1.5 ITIL

La Biblioteca de Infraestructura de Tecnologías de Información, abreviada como ITIL, por sus siglas en inglés, proporciona un marco de trabajo para la gestión de servicios de tecnologías de información, mediante una lista de las mejores prácticas adquiridas con el tiempo, relacionadas al control, operación y administración de los recursos disponibles y así, promover un acercamiento constante hacia la gestión de calidad y mejora continua de las tecnologías de información. De esta manera, se atienden necesidades tales como:

- Planeación estratégica para el negocio y las tecnologías de información
- Integración y alineación entre las metas del negocio y las tecnologías de información
- Implementación del concepto de mejora continua
- Medición de la efectividad y eficiencia de las tecnologías de información en la organización
- Optimización de costos
- Utilización de tecnologías de información para aumentar la ventaja competitiva
- Administración del cambio tanto en el negocio como en las tecnologías de información

El objetivo principal de la administración de servicios es asegurar que las tecnologías de información estén alineadas a las necesidades del negocio y soportarlo de manera activa. Es de gran importancia que las tecnologías de información actúen como un agente de cambio para facilitar la transformación del negocio. Para ello, se definen las etapas del marco de trabajo de ITIL, mostradas en la figura 1.12, que van desde la definición y análisis iniciales de los requerimientos del negocio en la estrategia y diseño del servicio, a través de la migración al ambiente de producción dentro de la etapa de transición del servicio, para llegar a la operación diaria y mejoramiento mediante las etapas de operación del servicio y mejora continua.

Figura 1.12 Marco de trabajo de ITIL

En cuanto a la etapa de estrategia del servicio de ITIL, se establece la forma de diseñar, desarrollar e implantar la gestión de la cartera de servicios como un activo estratégico. Posteriormente, en la etapa de diseño del servicio, se define un plan de administración de elementos, tales como capacidad, disponibilidad y continuidad del servicio, proveedores, niveles de respuesta, así como también la seguridad de la información.

Después, en la etapa de transición del servicio, se gestiona la configuración, la adaptación al cambio, pruebas, validación e implantación del servicio, sin descartar el apropiado manejo del conocimiento.

Referente a la etapa de operación del servicio, se proponen guías para obtener eficiencia y efectividad en la entrega y soporte del servicio mediante la gestión de accesos, eventos, incidentes y problemas para cumplir con las expectativas en cada solicitud de servicio.

Finalmente, la etapa de mejora continua del servicio se encarga de identificar posibles áreas de oportunidad para las etapas de diseño, transición y operación del servicio, con el objetivo de proporcionar más valor al cliente, mediante informes y mediciones de variables representativas del comportamiento del servicio.

1.6 Conclusiones del capítulo

Una vez terminada la revisión de diversas metodologías de ingeniería de software, es evidente el cambio que han tenido en el transcurso de los años para adaptarse a las necesidades del mundo actual. De igual manera, se aprecia que la incorporación de gestión de riesgos y enfoques de desarrollo incremental tienen cada vez mayor importancia al momento de generar un producto de software nuevo. A su vez, la participación del cliente en un proyecto de este tipo, es cada vez mayor, pues de esa manera se asegura el cumplimiento de sus expectativas respecto a los requerimientos que ha proporcionado. Asimismo, para facilitar el entendimiento de los requerimientos, herramientas como UML han sido utilizadas de manera exitosa, causando una mejor comunicación entre el cliente y los desarrolladores de software

En lo que respecta al mantenimiento de software, podemos observar que las mejores prácticas y lecciones aprendidas en el pasado han dado lugar a estándares destinados a la apropiada gestión del mantenimiento y soporte de sistemas. Por ello es que librerías como ITIL son comúnmente usadas en la actualidad.