UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

TESIS

DISEÑO E IMPLEMENTACIÓN DE APLICACIÓN PARA PRÉSTAMO DE LIBROS DE LA BIBLIOTECA CENTRAL EN DISPOSITIVOS MÓVILES ANDROID

QUE PARA OBTENER EL TITULO DE

INGENIERA EN COMPUTACIÓN
INGENIERO ELÉCTRICO Y ELECTRÓNICO

PRESENTAN:

ANGÉLICA MÉNDEZ VEGA
JOSÉ AGUILERA LÓPEZ

DIRECTOR:
M. C. ALEJANDRO VELÁZQUEZ MENA

CIUDAD UNIVERSITARIA
21/ABRIL/2015

AGRADECIMIENTOS

ANGÉLICA

Quiero agradecer a mi familia por su amor, apoyo y enseñanza incondicional.

A José, mi persona favorita, quien ha estado conmigo en los buenos y malos momentos.

A mis amigos por compartir su vida, sueños y alegrías.

A la UNAM, a la Biblioteca Central y a la facultad de Ingeniería. Alma mater.

Agradezco de corazón a todos mis seres queridos que con su ejemplo me han ayudado a ser una mejor persona día con día.

JOSÉ

A mi mamá y a mi papá por haberme apoyado durante mis estudios y durante toda mi vida. Y a mi hermano Israel que se ha convertido en un gran amigo.

A Angie por permitirme ser parte de su vida a pesar de todos mis defectos y hacerme feliz con su compañía.

A la UNAM, en especial a la Facultad de Ingeniería, representadas por los miembros del jurado.

CONTENIDO

Índice de Ilustraciones

Índice de Tablas

INTRODUCCIÓN

Primero, se revisarán los temas importantes de la ingeniería de software que sirven de fundamento para el ciclo de vida de las aplicaciones de software. Posteriormente, se revisará la plataforma objetivo de la aplicación: los dispositivos móviles y particularmente el sistema operativo Android. Finalmente, se presentarán los procesos de ingeniería de software enfocados en la obtención de la aplicación lista para su entrega a las autoridades correspondientes de la Biblioteca Central.

Con la aplicación móvil para la plataforma Android, cuyo objetivo es la Biblioteca Central de la UNAM, y que sustenta la presente tesis, será posible que cualquier usuario, previamente registrado en la Biblioteca Central, pueda tomar prestado el material bibliográfico disponible sin apoyo del personal de la biblioteca. El auto-préstamo se realizará siguiendo siempre las reglas de seguridad establecidas para la autorización del mismo. También se ha contemplado el servicio de renovación de préstamo de material a través de la misma aplicación.

Las pruebas que se han realizado de la aplicación, previas a su entrega a la Biblioteca Central, han sido ejecutadas realizando peticiones a un servidor de pruebas perteneciente a dicha institución; el servidor de pruebas empleado hasta el momento aún no consulta el catálogo completo de la biblioteca ni el directorio de usuarios existentes. Sin embargo, la apertura de los servicios, requeridos por la aplicación para poder ser utilizada por la comunidad, queda completamente fuera del alcance de la presente tesis y será responsabilidad de la Biblioteca Central la liberación de la aplicación.

CAPÍTULO 1 - INGENIERÍA DE SOFTWARE

En el presente capítulo se expondrán temas de ingeniería de software que, en pocas palabras, describen la base de conocimiento para el ciclo de vida ideal de cualquier sistema de software. Se partirá de las definiciones generales de cada concepto y se explicarán los puntos más importantes al respecto.

1.1 Introducción a la ingeniería de software

La definición de ingeniería de software del "Software and Systems Engineering Vocabulary" (SEVOCAB), traducida, es: "La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería de software"[1].

Un detalle importante a resaltar de la definición anterior es que se están considerando, de manera generalizada, las tres fases que componen el ciclo de vida completo de un software, es decir:

- Desarrollo.
- Operación.
- Mantenimiento.

En el ambiente laboral es común encontrar que dichas fases sean ejecutadas por distintos equipos o empresas de software.

El objetivo fundamental de la ingeniería de software es el desarrollo de sistemas de software de alta calidad que cumplan, en la medida de lo posible, con las fechas de entrega y con el costo proyectado; sin embargo, es una tarea complicada porque con el avance de la tecnología los sistemas de software se han vuelto más complejos. Además, posterior a la entrega del software, es necesario encargarse del soporte y operación del mismo porque, seguramente, se encontrarán errores y se requerirá añadir nuevas características.

1.2 Requerimientos de software

Los requerimientos de software expresan las necesidades y condiciones existentes en un sistema de software para resolver un problema del mundo real; se puede pensar en los requerimientos como el resultado de la comunicación entre los usuarios, el cliente y los desarrolladores del software.

Fundamentalmente, se explicarán las definiciones necesarias y los conceptos de requerimientos de sistema y requerimientos de usuario. Se plantearán las definiciones de requerimientos funcionales y no funcionales. Y, finalmente, se explicará cómo organizar los requerimientos obtenidos.

1.2.1 Requerimientos funcionales y no funcionales

Los requerimientos funcionales son declaraciones de los servicios que el sistema debe proveer, de cómo reaccionará el sistema a entradas particulares y cómo reaccionará ante ciertas situaciones. Es posible incluir las características que no incluye el sistema en caso de que sea necesario aclarar algún punto. También se puede definir a un requerimiento funcional como aquel al que se puede ejecutar un conjunto de pruebas para validar su comportamiento ante el cliente y los usuarios finales[2].

Los requerimientos no-funcionales son los requisitos que deben cumplir los servicios y funciones del sistema, están relacionados con propiedades que definen la calidad y comportamiento del sistema en su totalidad. Estos requerimientos se pueden descomponer en varios grupos:

- *Requerimientos de producto*: detallan el comportamiento del producto, por ejemplo la velocidad de respuesta, cantidad de recursos de hardware necesarios (procesador, memoria), etc. Estos requerimientos pueden ser definidos por métricas como velocidad, tamaño, capacidad y porcentajes.
- *Requerimientos de la organización*: se derivan de las políticas y procedimientos en la organización del cliente y de los desarrolladores. Algunos ejemplos son los estándares para el producto, el lenguaje de programación a emplear, fechas de entrega para el producto y la documentación.
- *Requerimientos externos*: se consideran los requerimientos derivados de factores externos al sistema y a su proceso de desarrollo. Se incluyen los requerimientos de interoperabilidad con otros sistemas dentro o fuera de la misma organización, requerimientos para cumplir normativas establecidas en la ley de una jurisdicción en particular, requerimientos de ética para asegurar que el sistema será aceptable para los usuarios y el público en general.

1.2.2 Requerimientos de dominio

Estos requerimientos se derivan del dominio en el que se va a implantar el sistema de software. Incluyen en su redacción terminología y conceptos especializados en el dominio de la aplicación. A partir de ellos se pueden generar requerimientos funcionales e inclusive condicionar algunos requerimientos funcionales previos.

Los requerimientos de dominio surgen de las condiciones del cliente, por ejemplo de la arquitectura de los sistemas existentes con los que se va a interactuar y la legislación bajo la cual se rige el negocio.

1.2.3 Requerimientos de usuario

Los requerimientos de usuario son enunciados, descritos en lenguaje natural y/o diagramas, de los servicios que el sistema debe proveer y bajo qué condiciones; son los requerimientos para el cliente del sistema y los usuarios finales. Deben describir los requerimientos funcionales y no-funcionales para que sean entendibles por los usuarios del sistema sin un amplio conocimiento técnico.

1.2.4 Requerimientos de sistema

Los requerimientos de sistema son versiones de los requerimientos de usuario que los ingenieros de software pueden utilizar como punto de partida para el diseño del sistema. Los requerimientos de sistema presentan las funciones de sistema, servicios y las restricciones operativas a detalle. Deben ser una completa y concisa especificación del sistema que se va a desarrollar[3].

Usualmente, es posible descomponer un requerimiento de usuario en varios requerimientos de sistema para añadir detalle, explicar las funciones y servicios que debe proveer el sistema.

El lenguaje natural no es recomendable para estos requerimientos debido a que la interpretación del mismo dependerá de quienes lo hayan redactado y, posteriormente, de quienes lean el documento. Los requerimientos de sistema pueden ser escritos utilizando notaciones especializadas, por ejemplo:

- *Lenguaje natural estructurado*: depende de la definición de formularios y plantillas para expresar los requerimientos.
- *Lenguajes de descripción de diseño*: se utiliza un lenguaje parecido a un lenguaje de programación pero con características para especificar un modelo operacional del sistema.
- *Notación gráfica*: un lenguaje grafico acompañado de texto para definir los requerimientos funcionales del sistema. Actualmente se utilizan los diagramas de casos de uso y los diagramas de secuencia.
- *Especificaciones matemáticas*: son notaciones basadas en conceptos matemáticos como maquinas de estado finito o conjuntos.

1.3 Procesos de requerimientos de ingeniería

El objetivo del proceso de requerimientos de ingeniería es crear y mantener un documento de requerimientos de sistema. El proceso en general se divide en cuatro subprocesos. El estudio de factibilidad evalúa la utilidad del sistema para el negocio; el levantamiento y análisis se encarga del descubrimiento de requerimientos; en la especificación se concentran los requerimientos en un formato estandarizado; y durante la validación de requerimientos se corrobora que dichos requerimientos definen el sistema que el cliente quiere[4]. El proceso completo se muestra en la ilustración 1-1.

Los subprocesos descritos anteriormente están enfocados en la definición, documentación y aprobación de requerimientos para el sistema, sin embargo, e invariablemente, los requerimientos sobre el sistema cambian constantemente. Las modificaciones pueden ser de hardware, software e inclusive en la organización. El proceso encargado de la administración de cambios en los requerimientos es la administración de requerimientos.

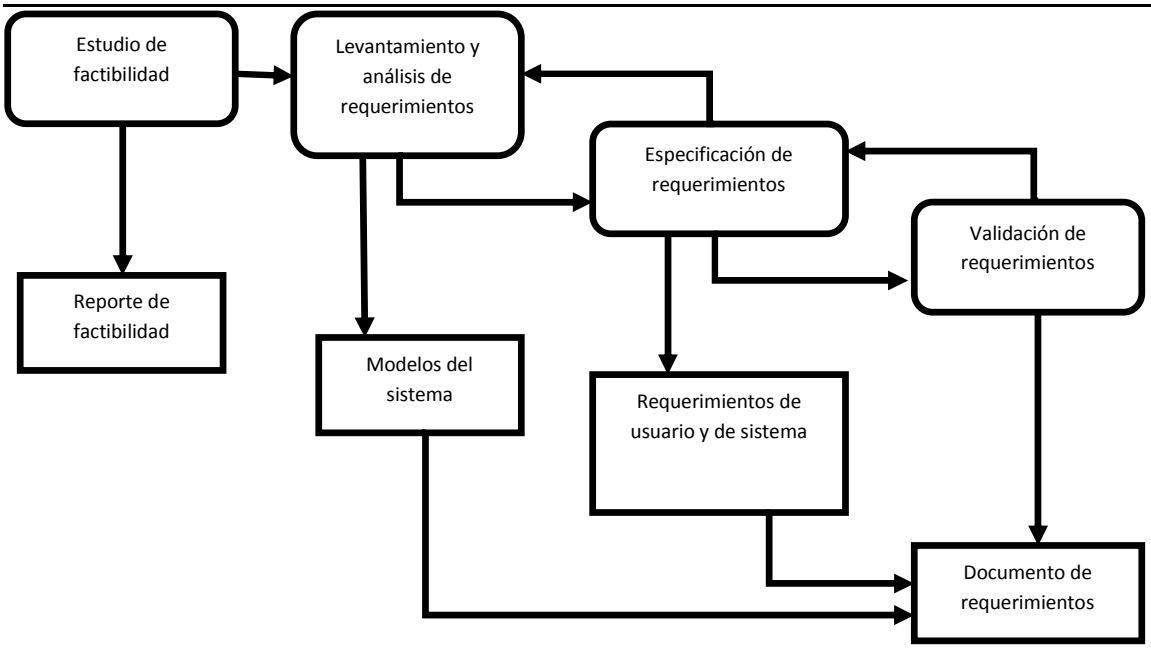


Ilustración 1-1 - Proceso de ingeniería de requerimientos

1.3.1 Estudio de factibilidad

Se puede iniciar un estudio de factibilidad partiendo de un conjunto de requerimientos de negocio, una descripción o esquema del sistema y saber como el sistema apoyara al negocio. El estudio de factibilidad debe ser algo breve y objetivo que pueda dar respuesta a las siguientes cuestiones:

- ¿El sistema contribuye a los objetivos principales de la organización?
- ¿El sistema puede ser implementado con la tecnología disponible y, cumpliendo las condiciones de costo y fechas de entrega?
- ¿El sistema puede ser integrado con otros sistemas que se encuentran actualmente en operación?

La cuestión de si el sistema contribuye o no a los objetivos de negocio es crítica. Si un sistema no cumple estos objetivos, no tiene ningún valor real que aportar al negocio. Si bien esto puede parecer obvio, muchas organizaciones desarrollan sistemas que no contribuyen a sus objetivos de negocio, ya que no tienen una clara declaración de dichos objetivos, porque no pueden definir los requisitos de negocio para el sistema o porque otros factores políticos o de organización influyen en la adquisición del sistema.

Llevar a cabo un estudio de viabilidad implica la recolección de información, evaluar dicha información y redactar informes al respecto. La fase de evaluación de la información identifica la información que se requiere para responder a las tres preguntas que se platearon arriba. Una vez que la información ha sido identificada, se debe hablar con las fuentes de información para obtener las respuestas a esas preguntas. Algunos ejemplos de posibles preguntas son:

- ¿Cómo puede prosperar la organización si éste sistema no se llevará a cabo?
- ¿Cuáles son los problemas con los procesos actuales y cómo ayudaría un nuevo sistema a aliviar esos problemas?
- ¿Qué contribución directa aportaría el sistema a los requerimientos y objetivos de negocio?
- ¿La información del sistema podría ser transferida hacia y desde otros sistemas de la organización?
- ¿El sistema requiere tecnología que no ha sido implementada previamente en la organización?

En un estudio de viabilidad es necesario consultar las fuentes de información, tales como los gerentes de los departamentos en los que se utilizará el sistema, los ingenieros de software que están familiarizados con el tipo de sistema que se propone, expertos en tecnología y los usuarios finales del sistema.

1.3.2 Levantamiento y análisis de requerimientos

En esta actividad, los ingenieros de software trabajan con los clientes y los usuarios finales del sistema para averiguar sobre el dominio de aplicación, los servicios que el sistema debe proporcionar, el rendimiento requerido del sistema, las limitaciones de hardware, y así sucesivamente. El término de *las partes interesadas*, *stakeholders* en inglés, se utiliza para referirse a cualquier persona o grupo que se verán afectados por el sistema, directa o indirectamente.

Las actividades del proceso de levantamiento y análisis de requerimientos son:

- *Descubrimiento de requerimientos*. Este es el proceso de interacción con las partes interesadas en el sistema para recolectar sus necesidades del sistema. Durante esta actividad también se descubren documentos y requisitos de dominio de las partes interesadas.
- *Clasificación y organización de requerimientos*. En esta actividad se toman los requerimientos sin clasificar y se agrupan en grupos de requerimientos relacionados.
- *Priorización y negociación de los requerimientos*. Inevitablemente, donde existen múltiples interesados involucrados, los requisitos entrarán en conflicto. Esta actividad tiene que ver con la negociación de los conflictos existentes en los requerimientos.
- *Documentación de requerimientos*. Los requerimientos se documentan para ser analizados posteriormente.

Las actividades de levantamiento y análisis de requerimientos se repiten continuamente como se ve en la ilustración 1-2.

Ilustración 1-2 - Proceso de levantamiento y análisis de requerimientos

1.3.3 Validación de requerimientos

La validación de requerimientos tiene que ver con demostrar que los requerimientos realmente definen el sistema que el cliente quiere. La validación de requerimientos es importante porque los errores en un documento de requerimientos pueden conducir a altos costos de reproceso cuando son descubiertos durante el desarrollo o inclusive cuando el sistema se encuentra en servicio.

Durante el proceso de validación de los requisitos, deben llevarse a cabo controles en el documento de requerimientos. Estos controles incluyen:

- *Validez*. Un usuario puede pensar que se necesita un sistema para llevar a cabo ciertas tareas. Sin embargo, con mayor reflexión y análisis se pueden identificar funciones adicionales que sean requeridas.
- *Consistencia*. Comprueba que los requerimientos definidos en el documento no entren en conflicto. Es decir, no debe haber definiciones o condiciones contradictorias en el funcionamiento del sistema.
- *Integridad*. Comprueba que el documento de requerimientos incluya todos los requerimientos y condiciones que definen todas las funciones del sistema.
- *Realismo*: Con el conocimiento de la tecnología existente, los requisitos deben ser evaluados para verificar que en realidad podrán ser implementadas. Estos controles también deben tener en cuenta el presupuesto y el calendario para el desarrollo del sistema.

- *Verificabilidad*. Para reducir la posibilidad de conflictos entre el cliente y el proveedor, los requisitos del sistema siempre deben estar escritos para que sean verificables. Esto significa que se debe escribir un conjunto de pruebas que puedan demostrar que el sistema reúne todos los requerimientos especificados.

1.3.4 Administración de requerimientos

La administración de requerimientos es el proceso de comprender y controlar cambios en los requerimientos del sistema. Es necesario hacer un seguimiento de los requerimientos para detectar dependencias con otras partes del sistema y así poder evaluar el impacto de cambios en los requerimientos. Se debe establecer un proceso formal para hacer propuestas de cambios y posteriormente se vinculen con los requisitos del sistema. El proceso de administración de requerimientos debe comenzar tan pronto como un borrador del documento de requerimientos esté disponible.

La evolución de los requerimientos, una vez que el sistema se encuentra en servicio, es inevitable. Se necesita dar seguimiento a los requerimientos individuales y mantener claras las dependencias entre ellos para poder medir el impacto de la modificación en algún requerimiento. Ilustración 1-3.

Conforme se desarrollan los requerimientos se obtiene un mejor conocimiento de las necesidades de los usuarios, quienes pueden proponer cambios en los requerimientos y finalmente se obtiene una serie de requerimientos que han sido analizados por el equipo de desarrollo y los propios usuarios, como se muestra en la imagen siguiente.

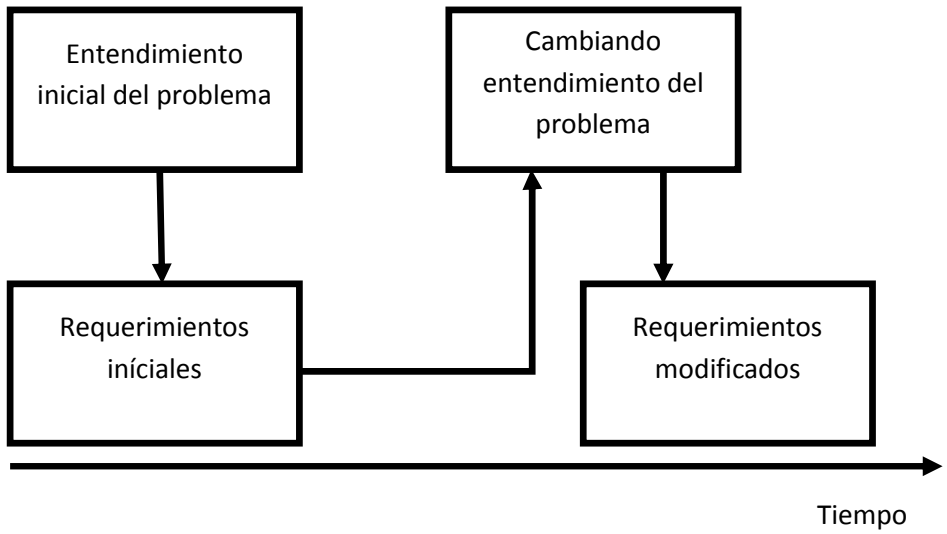


Ilustración 1-3 - Evolución de los requerimientos

Desde la perspectiva de la evolución del software, los requerimientos caen en dos clases:

1. *Requerimientos perdurables*. Estos son requerimientos relativamente estables que se derivan del núcleo de las actividades de la organización y que se relaciona directamente con el dominio de aplicación del sistema.

2. *Requerimientos volátiles*. Estos son los requerimientos que son propensos a modificarse durante el proceso de desarrollo del sistema e inclusive ya que el sistema se encuentra en operación.

1.4 Diseño

El diseño se define como "el proceso de definición de la arquitectura, componentes, interfaz y otras características de un sistema o componente"[5]. Visto como un proceso, el diseño de software es la actividad del ciclo de vida del software en la que el documento de requerimientos se analiza con el fin de producir una descripción de la estructura interna del software que servirá de base para su construcción. El resultado de la etapa de diseño produce la arquitectura de software, es decir, como se encuentra organizado el sistema de software en componentes particulares y las interfaces entre dichos componentes. También describe los componentes del sistema de software con tal detalle que es posible su construcción.

El diseño de software juega un papel importante en desarrollo de software: durante el diseño del software, los ingenieros de software producen varios modelos que forman una especie de borrador de la solución a implementar. Se pueden analizar y evaluar estos modelos para determinar si será posible cumplir con los diversos requerimientos del sistema.

También es posible examinar y evaluar soluciones alternativas y cambios. Por último, se pueden utilizar los modelos resultantes para planificar las actividades posteriores al desarrollo, como la verificación y la validación del sistema, además de su uso como puntos de arranque de la construcción del sistema y las pruebas.

1.4.1 Diseño de la arquitectura

Los grandes sistemas siempre se descomponen en sub-sistemas que proporcionan un conjunto de servicios relacionados. El proceso inicial de identificar estos subsistemas y el establecimiento de un marco de trabajo para el control y la comunicación de los subsistemas se define como diseño de arquitectura. El resultado de este proceso de diseño es la descripción de la arquitectura del software.

A continuación se enlistan ventajas del diseño y documentación de la arquitectura de un sistema de software:

- *Comunicación con los interesados.* La arquitectura es una presentación de alto nivel del sistema que puede ser utilizado como una base para la discusión por una gama de diferentes grupos de interés.
- *Análisis del sistema*. Hacer la arquitectura del sistema en una etapa temprana en el desarrollo del sistema requiere un poco de análisis. Las decisiones de diseño arquitectónico tienen un profundo efecto en si el sistema puede cumplir con los requisitos críticos tales como el rendimiento, la fiabilidad y facilidad de mantenimiento.
- *Alcance a gran escala*. El modelo de la arquitectura del sistema es una compacta y manejable descripción de cómo un sistema está organizado y de cómo interactúan sus componentes. La arquitectura del sistema es a menudo la misma para otros sistemas con

exigencias y requerimientos similares, así que pueden soportar la reutilización de software a gran escala.

La arquitectura del sistema afecta al rendimiento, robustez, distributividad y mantenimiento de un sistema. El estilo y la estructura particular elegido para una aplicación, puede depender de requerimientos no funcionales del sistema:

- *Rendimiento*. Si el rendimiento es un requisito crítico, la arquitectura debe ser diseñada para localizar las operaciones críticas dentro de un pequeño número de subsistemas, con tan poca comunicación como sea posible entre estos subsistemas.
- *Seguridad*. Si la seguridad es un requisito fundamental, se debe usar una estructura en capas para la arquitectura, con los activos más importantes protegidos en la capa más interna.
- *Disponibilidad*. Si la disponibilidad es un requisito crítico, la arquitectura debe ser diseñada para incluir componentes redundantes, de modo que sea posible sustituir y actualizar los componentes sin detener el sistema.
- *Mantenibilidad*. Si la mantenibilidad es un requisito fundamental, la arquitectura del sistema debe diseñarse utilizando componentes independientes que pueden ser fácilmente modificados. Los productores de datos deben estar separados de los consumidores y las estructuras con datos compartidos deben ser evitados.

Obviamente existe un potencial conflicto entre algunas de estas arquitecturas. Por ejemplo, usando componentes no tan detallados mejora el rendimiento y, por el contrario, con el uso de componentes muy detallados se mejora la mantenibilidad. Si ambos son requisitos importantes en el sistema, entonces se debe encontrar alguna solución que comprometa ambos requerimientos arquitectónicos. Esto a veces se puede lograr mediante el uso de diferentes estilos arquitectónicos para diferentes partes del sistema.

Hay una superposición significativa entre los procesos de requerimientos de ingeniería y el diseño arquitectónico. Idealmente, una especificación de sistema no debe incluir ninguna información de diseño. En la práctica, esto es poco realista excepto para sistemas muy pequeños.

Un diseño de sub-sistema es la descomposición de un sistema en componentes con un nivel de detalle no muy grande, cada uno de los cuales puede ser un sistema sustancial independiente del resto.

1.4.2 Modelos arquitectónicos del sistema

El resultado del proceso de diseño arquitectónico es el documento de diseño arquitectónico. Este puede incluir una serie de representaciones gráficas del sistema con el texto descriptivo asociado. Debe describir cómo el sistema está estructurado en subsistemas, el enfoque adoptado y la forma en que cada sub-sistema se estructura en módulos. Los modelos gráficos del sistema presentan diferentes perspectivas sobre la arquitectura. Los modelos arquitectónicos que se pueden desarrollar pueden incluir:

1. *Modelo estructural estático* que muestra los subsistemas o componentes que son desarrollados como unidades separadas.
2. *Modelo de proceso dinámico* que muestra cómo el sistema está organizado en procesos en tiempo de ejecución.
3. *Modelo de interfaz* que define los servicios ofrecidos por cada sub-sistema a través de su interfaz pública.
4. *Modelos de relación* que muestra las relaciones, como el flujo de datos, entre los sub-sistemas.
5. *Modelo de distribución* que muestra cómo los sub-sistemas pueden distribuirse a través de diferentes computadoras.

Modelos organizacionales del sistema

La organización de un sistema refleja la estrategia básica que se utiliza para estructurar un sistema. Se tienen que tomar decisiones sobre el modelo organizacional del sistema en general en las etapas tempranas del proceso de diseño arquitectónico. La organización del sistema puede ser directamente reflejada en la estructura de los sub-sistemas. Sin embargo, a menudo es el caso que algún modelo de un sub-sistema incluye más detalles que el modelo organizacional global, y no siempre hay un mapeo sencillo desde los subsistemas hacia la estructura organizacional.

En esta sección, se discuten tres estilos organizacionales que son muy utilizados. Estos son el estilo de repositorio de datos compartidos, el estilo de servicios y servidores compartidos y el estilo en capas en el que el sistema está organizado como la unión de capas funcionales. Estos estilos se pueden utilizar juntos o por separado. Por ejemplo, un sistema puede organizarse en torno a un repositorio de datos compartidos, pero se pueden construir capas alrededor de este para presentar una visión más abstracta de los datos.

Modelo de repositorio

Los subsistemas que componen un sistema deben intercambiar información de manera que puedan trabajar juntos de manera efectiva. Hay dos maneras fundamentales en que esto se puede hacer.

1. Todos los datos compartidos se mantienen en una base de datos central que se puede acceder por todos los sub-sistemas. Un modelo de sistema basado en una base de datos compartida a veces se llama modelo de repositorio.
2. Cada subsistema mantiene su propia base de datos. Los datos se intercambian con otros subsistemas enviándose mensajes entre ellos.

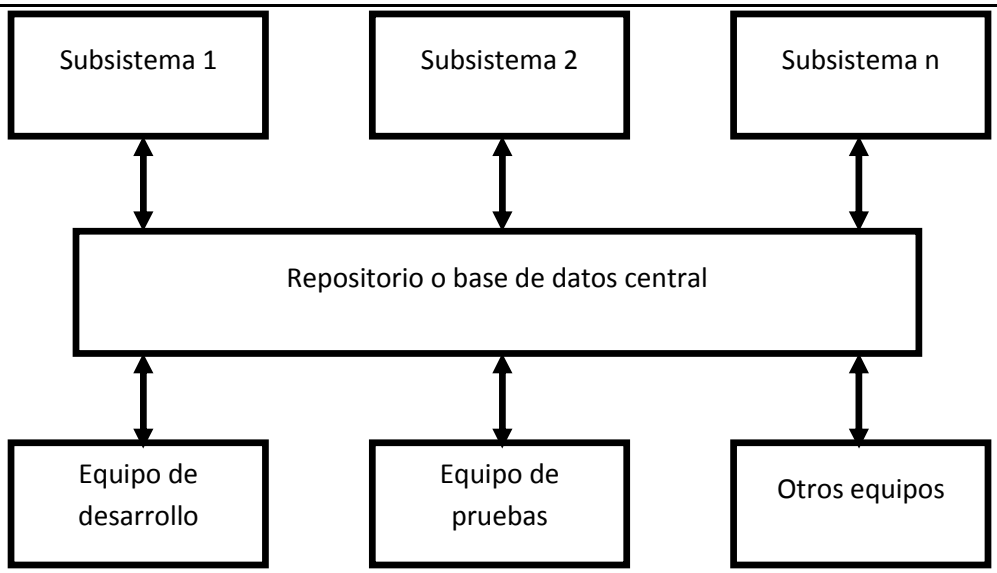


Ilustración 1-4 - Arquitectura: modelo de repositorio

La mayoría de los sistemas que utilizan grandes cantidades de datos se organizan en torno a una base de datos o repositorio compartido, como se muestra en la ilustración 1-4. Por tanto, este modelo es adecuado para aplicaciones donde los datos son generados por un sub-sistema y utilizados por otros.

Modelo cliente-servidor

El modelo de arquitectura cliente-servidor es un modelo de sistema en el que el sistema está organizado como un conjunto de servicios y servidores asociados, con clientes que acceden y utilizan los servicios. Los componentes principales de este modelo son (ver ilustración 1-5):

1. *Un conjunto de servidores* que ofrecen servicios a otros subsistemas. Algunos ejemplos de servidores son los servidores de impresión que ofrecen servicios de impresión, servidores de archivos que ofrecen servicios de gestión de archivos y servidores de compilación, que ofrecen servicios de compilación para lenguajes de programación.
2. *Un conjunto de clientes* que llaman a los servicios ofrecidos por los servidores. Estos son normalmente subsistemas completos. Puede haber varias instancias de un programa cliente ejecutándose concurrentemente.
3. *Una red* que permite a los clientes acceder a estos servicios. Esto no es estrictamente necesario ya que tanto los clientes y los servidores podrían ejecutarse en una sola máquina. En la práctica, sin embargo, la mayoría de los sistemas cliente-servidor se implementan como sistemas distribuidos.

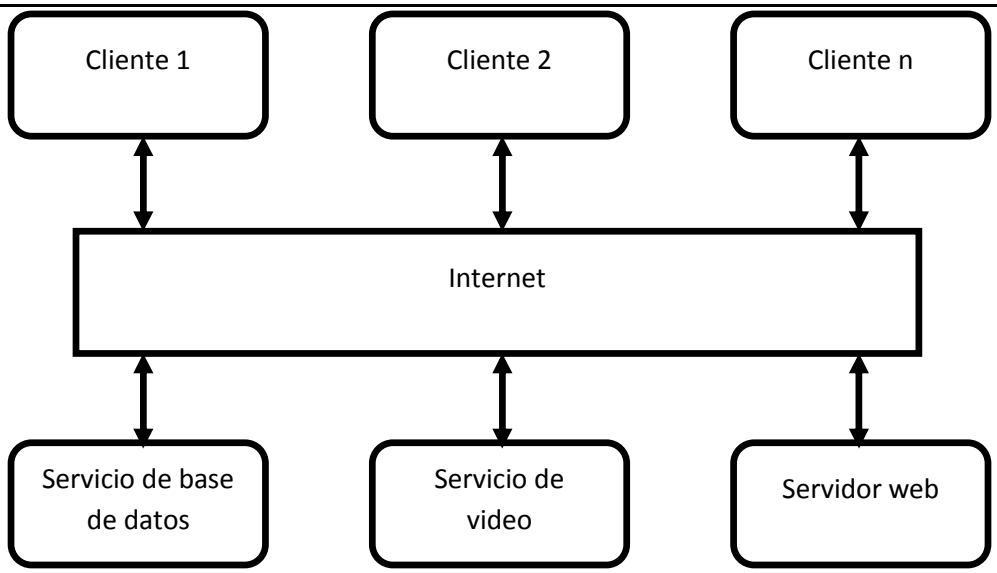


Ilustración 1-5 - Arquitectura: modelo cliente - servidor

Los clientes requieren conocer los nombres de los servidores disponibles y los servicios que proporcionan. Sin embargo, los servidores no necesitan saber la identidad de los clientes o cuántos clientes existen. Los clientes acceden a los servicios proporcionados por un servidor a través de llamadas a procedimientos remotos utilizando protocolos de petición-respuesta, como el protocolo http utilizado en internet. Esencialmente, un cliente realiza una petición a un servidor y espera hasta que recibe una respuesta.

Modelo en capas

El modelo en capas de una arquitectura (a veces llamado modelo de máquina abstracta) organiza un sistema en capas, cada una de las cuales provee un conjunto de servicios. Se puede pensar en cada capa como una maquina abstracta en la que su lenguaje está definido por los servicios que proporciona. Ese lenguaje es usado para implementar el siguiente nivel de la máquina abstracta.

Un ejemplo de esto es el modelo de interconexión de sistemas abiertos: modelo OSI. Ilustración 1-6.

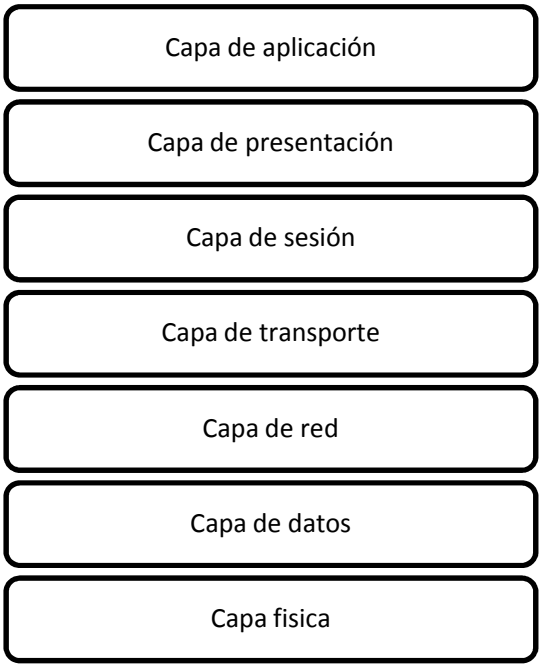


Ilustración 1-6 - Ejemplo de modelo en capas con el modelo OSI

El enfoque por capas soporta el desarrollo incremental de los sistemas. Cada que una capa se desarrolla, algunos de los servicios prestados por esa capa pueden ponerse a disposición de los usuarios. Esta arquitectura también es cambiante y portátil. Mientras su interfaz permanezca sin cambios, una capa puede ser sustituida por otra capa equivalente. Por otra parte, cuando la interfaz de una capa cambia o se añaden nuevas prestaciones a una capa, sólo las capas adyacentes se ven afectadas.

Una desventaja del enfoque por capas es que los sistemas estructurados de esta manera pueden ser complicados. Las capas interiores pueden proporcionar servicios básicos, tales como la gestión de archivos, que se requieren en todos los niveles. Los servicios requeridos por un usuario en el nivel superior podrían atravesar múltiples capas adyacentes para conseguir el acceso a los servicios que son proporcionados varios niveles por debajo del nivel superior. Ello implica que la capa exterior en el sistema no sólo depende de su predecesor inmediato, si no de todas las capas debajo de ella.

El rendimiento también puede ser un problema debido a los múltiples niveles de interpretación de comandos que a veces se requieren. Si hay muchas capas, una solicitud de servicio que parte de una capa superior puede requerir ser interpretada varias veces en diferentes capas antes de que pueda ser procesada. Para evitar estos problemas, las aplicaciones deberían comunicarse directamente con capas internas en lugar de utilizar los servicios proporcionados por las capas adyacentes.

1.4.3 Estilos de descomposición

Después de que se ha elegido la organización general del sistema, se necesita tomar una decisión para el enfoque que será utilizado en la descomposición de los sub-sistemas en módulos. No hay una distinción rígida entre la organización del sistema y la descomposición modular. Sin embargo, los componentes de los módulos suelen ser más pequeños que los sub-sistemas, lo cual permite utilizar estilos de descomposición alternativos.

No existe una distinción clara entre sub-sistemas y módulos, pero es útil pensar en una distinción entre ellos así:

- *Un sub-sistema* es un sistema por sus propias características y cuyo funcionamiento no depende de los servicios prestados por otros sub-sistemas. Los sub-sistemas se componen de módulos y tienen definidas interfaces que se utilizan para la comunicación con otros sub-sistemas.
- *Un módulo* es, normalmente, un componente del sistema que proporciona uno o más servicios a otros módulos y hace uso de los servicios prestados por otros módulos. No es considerado normalmente como un sistema independiente. Los módulos se componen generalmente de otros componentes del sistema más simples.

Hay dos estrategias principales que se pueden utilizar cuando se descompone un subsistema en módulos:

- *Descomposición orientada a objetos*, donde se descompone un sub-sistema en un conjunto de objetos que se comunican entre sí.
- *Descomposición en canalización de funciones*, donde se descompone un sub-sistema en módulos funcionales que aceptan datos de entrada y los transforman en datos de salida.

En el enfoque orientado a objetos, los módulos son objetos con estado privado y operaciones definidas para ese estado. En el modelo de la canalización, los módulos son transformaciones funcionales. En ambos casos, los módulos pueden implementarse como componentes secuenciales o como procesos.

Inicialmente, se debe evitar hacer planes prematuros de añadir concurrencia en un sistema. La ventaja de evitar el diseño de un sistema concurrente es que los programas secuenciales son más fáciles de diseñar, implementar, verificar y probar que los sistemas paralelos. Las dependencias de tiempo entre los procesos son difíciles de formalizar, controlar y verificar. Lo mejor es descomponer los sistemas en módulos, a continuación, decidir si durante la ejecución estos deben ejecutarse secuencialmente o en paralelo.

Descomposición orientada a objetos

Un modelo de arquitectura a orientada a objetos, ordena el sistema en un conjunto de objetos débilmente acoplados con interfaces bien definidas. Los objetos pueden llamar a los servicios ofrecidos por otros objetos.

Una descomposición orientada a objetos se refiere a clases de objetos, sus atributos y sus operaciones. Cuando se implementa, los objetos se crean a partir de clases y algún modelo de control se utiliza para coordinar las operaciones del objeto. Una clase puede hacer uso de otras clases que tienen otros estados y operaciones.

Las ventajas del enfoque orientado a objetos son bien conocidas. Dado que los objetos están débilmente acoplados, la implementación de los objetos puede ser modificada sin afectar a otros objetos. Los objetos son a menudo representaciones de entidades del mundo real por lo que la estructura del sistema es fácilmente comprensible. Debido a que estas entidades del mundo real son utilizadas en diferentes sistemas, los objetos pueden ser reutilizados.

Sin embargo, el enfoque orientado a objetos tiene desventajas. Para utilizar los servicios, los objetos deben hacer referencia explícitamente al nombre y la interfaz de otros objetos. Si una modificación en una interfaz es requerida para satisfacer los cambios propuestos para el sistema, el efecto del cambio se debe evaluar en todos los usuarios del objeto. Mientras que los objetos pueden mapear limpiamente a pequeñas entidades del mundo real, las entidades más complejas a veces son difíciles de representar como objetos.

Descomposición en canalización orientada a funciones

En una canalización orientada a funciones o modelo de flujo de datos, las transformaciones funcionales procesan sus entradas y producen salidas. Los datos fluyen y se transforman a medida que avanzan a través de la secuencia. Cada etapa de procesamiento se implementa como una transformación. Los datos de entrada fluyen a través de estas transformaciones hasta que se convierten a la salida.

Las transformaciones se pueden ejecutar secuencialmente o en paralelo. Los datos pueden ser procesados por cada transformación uno a la vez o en un solo lote.

Cuando las transformaciones se representan como procesos separados, este modelo es a veces llamado el estilo tubería y estilo filtro debido a la terminología utilizada en Unix. El sistema Unix ofrece tubos, *pipes*, que actúan como conductos de datos y un conjunto de comandos que actúan como transformaciones funcionales. Los sistemas que se ajusten a este modelo pueden ser implementados mediante la combinación de comandos de Unix utilizando tuberías y las facilidades de control del Shell de Unix. El termino filtro se utiliza porque una transformación filtra los datos que puede procesar desde su flujo de datos de entrada.

Las variantes de este modelo de canalización han estado en uso desde que las computadoras fueron utilizadas por primera vez para el procesamiento automático de datos. Cuando las transformaciones son secuenciales con datos procesados en lotes, este modelo arquitectónico es un modelo secuencial por lotes. Como ejemplo, se trata de una arquitectura común para los sistemas de procesamiento de datos tales como sistemas de facturación, sistemas de procesamiento de datos que suelen generar muchos informes de salida que se derivan de cálculos simples a partir de un gran número de registros de entrada.

Nótese la diferencia entre este estilo y su equivalente orientado a objetos discutido previamente. El modelo de objetos es más abstracto, ya que no incluye información sobre la secuencia de las operaciones.

Las ventajas de esta arquitectura son:

1. Permite la reutilización de las transformaciones.
2. Es intuitivo ya que mucha gente piensa en su trabajo en términos de procesamiento de entrada y salida.
3. La evolución del sistema mediante la adición de nuevas transformaciones suele ser sencillo.
4. Es simple de implementar ya sea como un sistema concurrente o secuencial.

El principal problema de este estilo es que tiene que existir un formato común para la transferencia de datos que pueda ser reconocido por todas las transformaciones. En Unix, el formato estándar es simplemente una secuencia de caracteres. Cada transformación debe convertir los datos a su entrada y nuevamente a su salida para cumplir con el formato acordado. Esto aumenta la sobrecarga del sistema y puede significar que es imposible integrar transformaciones que utilizan formatos de datos incompatibles.

Los sistemas interactivos son difíciles de escribir usando el modelo canalización debido a la necesidad de un flujo de datos a procesar. Mientras los sistemas de entradas y salidas sencillas se pueden modelar fácilmente de esta manera, las interfaces gráficas de usuario tienen más complejidad en control y formatos de entradas y salidas, basados en eventos tales como clics del ratón o selecciones de menú. Es difícil traducir esto en una forma compatible con el modelo de canalización.

1.4.4 Estilos de control

Los modelos para la estructuración de un sistema tienen que ver con cómo se descompone un sistema en subsistemas. Para funcionar como un sistema, subsistemas deben ser controladas de manera que sus servicios se presten en el lugar adecuado en el momento adecuado. Los modelos estructurales no incluyen, y no deben incluir, información de control. También, el arquitecto debe organizar los sub-sistemas de acuerdo a algún modelo de control que complemente el modelo de estructura que se utiliza. Los modelos de control a nivel de arquitectura tienen que ver con el flujo de control entre los sub-sistemas.

Hay dos estilos de control genéricos que se utilizan en los sistemas de software:

1. *Control centralizado*. Un sub-sistema centralizado tiene la responsabilidad general de control, arranque y paro de otros subsistemas. También puede delegar el control a otro subsistema, pero queda en espera de obtener nuevamente la responsabilidad del control.

2. *Control basado en eventos*. Cada sub-sistema puede responder a eventos generados externamente. Estos eventos podrían venir de otros subsistemas o del entorno del sistema.

Los estilos de control se utilizan en conjunción con los estilos estructurales. Los estilos estructurales que se han discutido pueden ser implementados utilizando el estilo de control centralizado o el estilo basada en el control basado en eventos.

Control centralizado

En un modelo de control centralizado, un sub-sistema se designa como el controlador del sistema y tiene la responsabilidad de gestionar la ejecución de otros subsistemas. Los modelos de control centralizados pueden clasificarse en dos clases, dependiendo de si los sub-sistemas controlados se ejecutan secuencialmente o en paralelo.

1. *Modelo de llamada-retorno*. Es el modelo familiar de subrutinas de arriba hacia abajo (top-down) donde el control comienza en la cima de la jerarquía de subrutinas y, a través de llamadas a subrutinas, se llega a los niveles inferiores del árbol. El modelo subrutina sólo se aplica a sistemas secuenciales.
2. *El modelo de administrador*. Este modelo es aplicable a sistemas concurrentes. Un componente de sistema es designado como un administrador del sistema y controla la puesta en marcha, la detención y la coordinación de otros procesos del sistema. Un proceso es un sub-sistema o módulo que puede ejecutarse en paralelo con otros procesos. Otra forma de este modelo puede aplicarse en los sistemas secuenciales donde una rutina de gestión llama a otros subsistemas en función de los valores de algunas variables de estado. Esto es generalmente implementado como una sentencia *switch-case* con algún lenguaje de programación.

El modelo de llamada-retorno se puede utilizar a nivel de módulo para controlar las funciones u objetos. Las subrutinas en un lenguaje de programación que son llamadas por otras subrutinas son naturalmente funcionales. Sin embargo, en muchos sistemas orientados a objetos, las operaciones en los objetos (métodos) se implementan como procedimientos o funciones. Por ejemplo, cuando un objeto Java solicita un servicio de otro objeto, lo hace llamando a un método asociado.

La naturaleza rígida y limitada de este modelo es a la vez una fortaleza y una debilidad. Es una fortaleza porque es relativamente simple para analizar flujos de control y definir cómo responderá el sistema a entradas particulares. Una debilidad es que las excepciones a la operación normal son difíciles de manejar. Se puede ver en la siguiente ilustración 1-7 un ejemplo de este modelo.

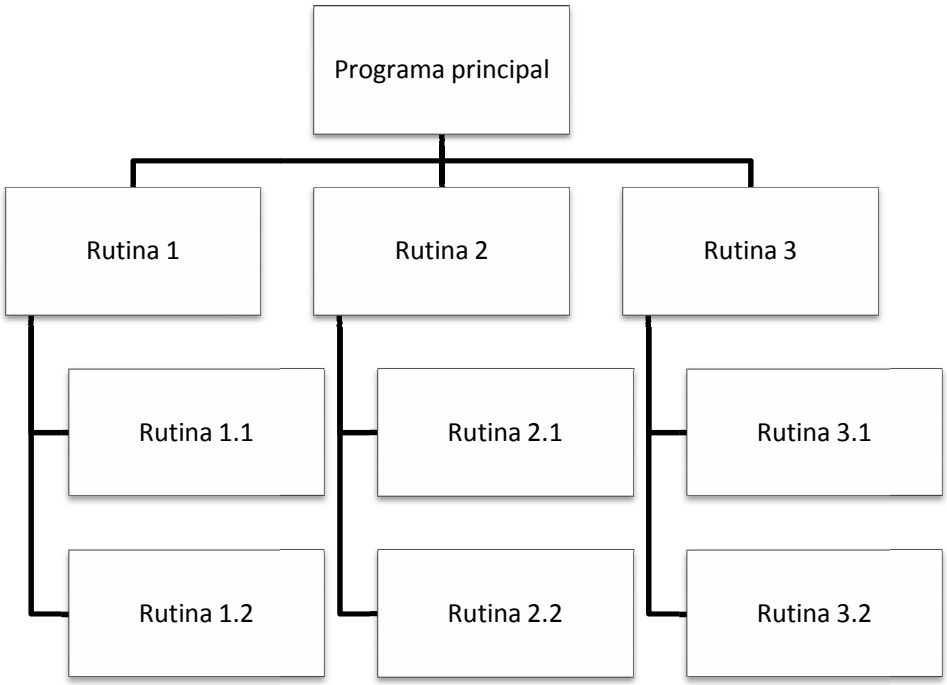


Ilustración 1-7 - Control centralizado con el modelo de llamada y retorno

El modelo administrador se utiliza a menudo en sistemas sencillos de tiempo real que no tienen limitaciones de tiempo muy ajustados. El controlador central o programa principal gestiona la ejecución de un conjunto de procesos asociados con sensores y actuadores.

El proceso controlador del sistema decide cuándo se deben iniciar o detener ciertos procesos en función de variables de estado del sistema. También comprueba si otros procesos han producido información para procesar o si puede enviar información para procesamiento. El controlador normalmente ejecuta sus rutinas en un bucle, sondeando sensores y a otros procesos para detectar eventos o cambios de estado. Por esta razón, este modelo a veces se denomina modelo de evento-bucle. En la ilustración 1-8 siguiente se puede observar un ejemplo de este modelo.

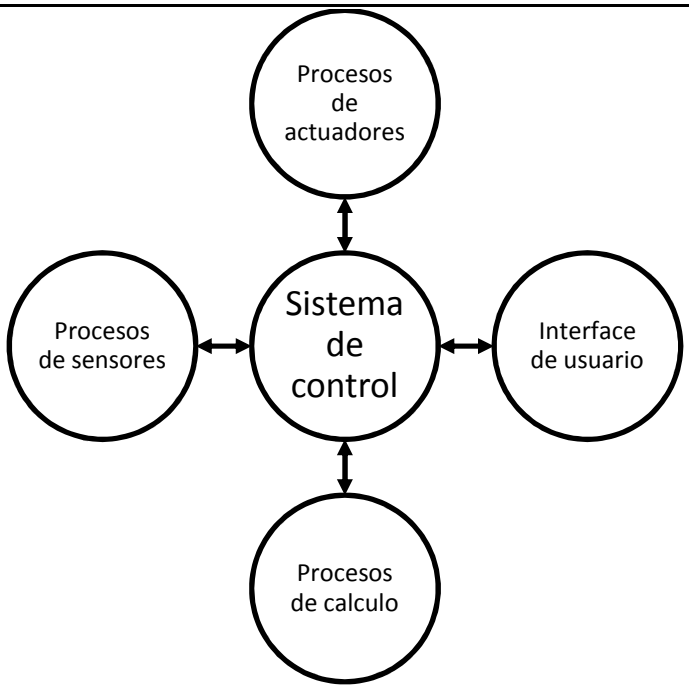


Ilustración 1-8 - Proceso centralizado basado en el modelo administrador

Control basado en eventos

En los modelos de control centralizado, las decisiones de control se determinan generalmente por los valores de algunas variables de estado del sistema. Por el contrario, los modelos de control basados en eventos son impulsados por eventos generados externamente. En este contexto el término evento no sólo significa una señal binaria, puede ser una señal que adopta una serie de valores o un comando de entrada desde un menú.

Hay muchos tipos de sistemas basados en eventos. Estos incluyen editores de texto donde la interfaz de usuario contiene los comandos de edición, sistemas de producción basados en reglas como los sistemas de inteligencia artificial en donde una condición verdadera produce que se active o desactive alguna acción en respuesta y sistemas basados en objetos donde el cambio de valor de un atributo en un objeto provoca algunas acciones.

En esta sección, se discuten dos modelos de control basados en eventos:

1. *Modelos de difusión*. En estos modelos, un evento se difunde a todos los sub-sistemas. Cualquier sub-sistema que ha sido programado para manejar ese evento puede responder al mismo. Ilustración 1-9.
2. *Modelos basados en interrupciones*. Estos modelos se utilizan exclusivamente en sistemas de tiempo real donde las interrupciones externas son detectadas por un manejador de interrupciones y a continuación se pasan a algún otro componente para su procesamiento.

Los modelos de difusión son efectivos en la integración de sub-sistemas de computadoras distribuidas en una red. Los modelos dirigidos por interrupciones son utilizados en sistemas de tiempo real con estrictos requerimientos de tiempo de respuesta.

Todos los eventos podrían ser difundidos a todos los sub-sistemas, pero esto impone una gran cantidad de sobrecarga de procesamiento. Más a menudo, el controlador de eventos y mensajes mantiene un registro de los subsistemas y los eventos de interés para ellos. Los subsistemas generan eventos indicando, tal vez, que algunos datos están disponibles para su procesamiento. El controlador detecta dichos eventos, consulta el registro de eventos y pasa el evento a aquellos sub-sistemas que hayan declarado un interés. En sistemas más simples, tales como los sistemas basados en PC impulsados por eventos en la interfaz de usuario, existen sub-sistemas explícitamente encargados de detectar eventos como los producidos por el ratón, el teclado, y así sucesivamente, y traducir estos eventos en comandos más específicos.

El controlador de eventos también generalmente soporta la comunicación de punto a punto. Un sub-sistema puede enviar explícitamente un mensaje a otro sub-sistema.

La ventaja de este enfoque es que la evolución de un sistema basado en difusión es relativamente simple. Un nuevo sub-sistema para manejar clases particulares de eventos puede ser integrado al registrar sus eventos en el controlador de eventos. Cualquier sub-sistema puede activar cualquier otro sub-sistema sin conocer su nombre o ubicación. Los subsistemas pueden ser en la práctica implementados en máquinas distribuidas. Esta distribución es transparente entre todos los sub-sistemas.

La desventaja de este modelo es que los subsistemas no conocen si o cuando sus eventos serán atendidos. Cuando un sub-sistema genera un evento no sabe que otros subsistemas tienen registrado un interés en ese evento. Es muy posible que diferentes subsistemas estén registrados para atender los mismos eventos, esto puede causar problemas cuando el controlador de eventos pone los resultados a disposición del elemento que generó el evento.

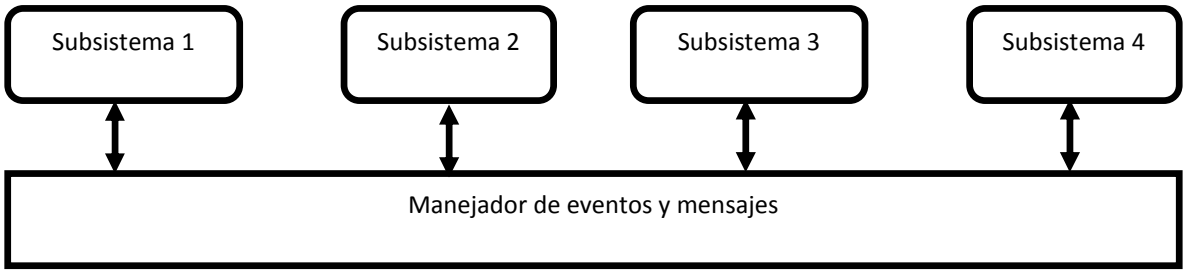


Ilustración 1-9 - Control basado en difusión

Los sistemas de tiempo real que requieren atender rápidamente eventos generados externamente deben ser sistemas basados en eventos. Por ejemplo, si un sistema de tiempo real se utiliza para controlar los sistemas de seguridad en un coche, se debe detectar un posible accidente y, quizás, inflar las bolsas de aire antes de que la cabeza del conductor se golpee en el volante. Para poder proporcionar esta rápida respuesta a los eventos, se debe usar el modelo de control por interrupciones.

Este modelo se utiliza sobre todo en sistemas de tiempo real, donde una respuesta inmediata a algún evento es necesaria. Se puede combinar con el modelo de gestión centralizado. El gestor

central maneja el funcionamiento normal del sistema delegando la atención de emergencias al modelo de control basado en interrupciones

La ventaja de este enfoque es que permite respuestas muy rápidas a los eventos a implementar. Sus desventajas son que es complejo de programar y difícil para validar su funcionamiento. Puede ser imposible de replicar los patrones de tiempo de interrupción durante la fase de pruebas de sistema. Puede ser difícil modificar los sistemas desarrollados utilizando este modelo si el número de interrupciones está limitado por el hardware. Una vez alcanzado este límite, no hay otra forma en que los eventos puedan ser manejados. A veces se puede superar esta limitación por mapeo de varios tipos de eventos en una sola interrupción. Entonces el controlador decide que evento es el que se ha producido. Sin embargo, el mapeo de interrupciones puede ser poco práctico si un sistema requiere una respuesta rápida a las interrupciones individuales.

1.4.5 Arquitectura de sistemas distribuidos

Prácticamente todos los grandes sistemas basados en computadoras son sistemas distribuidos. Un sistema distribuido es un sistema en el que el procesamiento de la información se distribuye en varias computadoras en lugar de limitarse a una sola máquina. Obviamente, la ingeniería de sistemas distribuidos tiene mucho en común con la ingeniería de cualquier otro software, pero hay cuestiones específicas que han de tenerse en cuenta en el diseño de este tipo de sistemas. Se identifican las siguientes ventajas de utilizar un enfoque de sistema distribuido para el desarrollo de sistemas:

1. *Compartir recursos*. Un sistema distribuido permite compartir recursos de hardware y software, tales como discos, impresoras, archivos y compiladores, que están asociados a equipos de una red.
2. *Apertura*. Los sistemas distribuidos son sistemas normalmente abiertos, lo que significa que están diseñados en torno a protocolos estándar que permiten combinar el sistema con los equipos y el software de diferentes proveedores.
3. *Concurrencia*. En un sistema distribuido, varios procesos pueden funcionar al mismo tiempo en equipos independientes de la red. Estos procesos pueden, aunque no necesariamente, comunicarse entre sí durante su funcionamiento normal.
4. *Escalabilidad*. En principio al menos, los sistemas distribuidos son escalables en el sentido de que las capacidades del sistema se pueden incrementar mediante la adición de nuevos recursos para hacer frente a las nuevas demandas al sistema. En la práctica, la red que une las computadoras en el sistema puede limitar la escalabilidad del sistema. Si muchas nuevas computadoras se añaden, entonces la capacidad de la red puede ser inadecuada.
5. *Tolerancia a fallos*. La disponibilidad de varias computadoras y el potencial de replicar información implica que los sistemas distribuidos pueden ser tolerantes a algunos fallos de software y hardware. En la mayoría de los sistemas distribuidos, un servicio básico se puede proporcionar si se producen averías. La pérdida completa de servicio sólo tiende a ocurrir cuando hay un fallo en la red.

Las ventajas anteriormente mencionadas implican que muchos de los antiguos sistemas de cómputo centralizado a gran escala, desarrollados en décadas previas, han sido reemplazados por sistemas distribuidos. Sin embargo, comparados con sistemas que se ejecutan solo en un procesador o en un grupo de procesadores, los sistemas distribuidos presentan un cierto número de desventajas:

1. *Complejidad*. Los sistemas distribuidos son más complejos que los sistemas centralizados. Esto hace que sea más difícil entender sus propiedades emergentes y probar estos sistemas. Por ejemplo, en lugar de que el rendimiento del sistema sea dependiente de la velocidad de ejecución de un procesador, más bien dependerá del ancho de banda de la red la velocidad de los procesadores distribuidos en dicha red. Mover recursos de una parte del sistema a otra puede afectar radicalmente el rendimiento del sistema.
2. *Seguridad*. El sistema se puede acceder desde varias computadoras diferentes y el tráfico en la red puede ser objeto de escuchas. Esto hace que sea más difícil asegurar que se mantiene la integridad de los datos en el sistema y que los servicios del sistema no son degradados por ataques de denegación de servicio.
3. *Manejabilidad*. Los equipos de un sistema pueden ser de diferentes tipos y pueden ejecutar diferentes versiones de sistemas operativos. Los fallos en una máquina pueden propagarse a otras máquinas con consecuencias inesperadas. Esto significa que se requiere un esfuerzo mayor para gestionar y mantener el sistema en funcionamiento.
4. *Imprevisibilidad*. Como todos los usuarios de internet saben, los sistemas distribuidos son impredecibles en su respuesta. La respuesta depende de la carga global en el sistema, su organización y la carga de la red. Como todos estos parámetros pueden cambiar durante un corto período de tiempo, el tiempo necesario para responder a una petición de usuario puede variar drásticamente de una solicitud a otra.

El desafío de la etapa de diseño es ser capaz de diseñar el software y hardware para proporcionar las características deseables del sistema distribuido y, al mismo tiempo, minimizar la probabilidad de problemas que son inherentes en este tipo de sistemas. Para ello, es necesario comprender las ventajas y desventajas de las diferentes arquitecturas de sistemas distribuidos. Se cubren aquí dos tipos genéricos de la arquitectura de sistemas distribuidos:

1. *Arquitecturas cliente-servidor*. En este enfoque, los sistemas pueden ser considerados como un conjunto de servicios que se proporcionan a los clientes que hacen uso de estos servicios. Los servidores y los clientes reciben un trato diferente en estos sistemas.
2. *Arquitecturas de objetos distribuidos*. En este caso, no hay distinción entre servidores y clientes, y el sistema puede ser pensado como un conjunto de objetos que interactúan entre sí y cuya ubicación es irrelevante. No hay distinción entre un proveedor de servicios y un usuario de estos servicios.

Ambas arquitecturas, cliente-servidor y objetos distribuidos, son ampliamente utilizados en la industria, pero la distribución de las aplicaciones se encuentra generalmente dentro de una sola organización. Por tanto, la distribución soportada es intra-organizacional. También se discutirán

otros dos tipos de arquitectura distribuida que son más adecuadas para la distribución entre organizaciones: arquitecturas de sistemas peer-to-peer (p2p) y arquitecturas orientadas a servicios (SOA). Los sistemas peer-to-peer en su mayoría se han utilizado para sistemas personales, pero están comenzando a ser utilizados para aplicaciones de negocios. La arquitectura orientada a servicios es un estilo basado en la operación conjunta y coordinada de servicios, recientemente se ha convertido en una de las opciones más fuertes para la integración de servicios distribuidos dentro de una o en varias organizaciones.

Los componentes de un sistema distribuido pueden implementarse en diferentes lenguajes de programación y pueden ejecutarse en diferentes tipos de procesadores. Los modelos de datos, la representación de la información y los protocolos de comunicación pueden ser todos diferentes. Por tanto, un sistema distribuido requiere un software que puede manejar estas diversas partes, y asegurar que puedan comunicarse e intercambiar datos. El termino *middleware* es usado para referirse a dicho software, se encuentra en medio de los diferentes componentes distribuidos de un sistema. Middleware es software de propósito general que es usualmente adquirido como tal en vez de ser escrito especialmente por desarrolladores de aplicaciones.

1.4.6 Arquitecturas cliente servidor

En una arquitectura cliente-servidor, una aplicación se modela como un conjunto de servicios que son proporcionados por servidores y un conjunto de clientes que utilizan estos servicios. Los clientes necesitan ser conscientes de los servicios disponibles, pero usualmente no conocen de la existencia de los otros clientes. Los servidores y los clientes tienen procesos separados.

La arquitectura cliente-servidor más simple se denomina arquitectura cliente-servidor de dos niveles, es cuando una aplicación se organiza como un servidor (o varios servidores idénticos) y un conjunto de clientes (Ilustración 1-10). Esta arquitectura simple puede ser de dos tipos:

1. *Modelo cliente-ligero*. En un modelo de cliente ligero, todo el procesamiento de aplicaciones y gestión de datos se lleva a cabo en el servidor. El cliente es responsable simplemente de ejecutar el software de presentación.
2. *Modelo cliente-pesado*. En este modelo, el servidor sólo es responsable de la gestión de datos. El software en el cliente implementa la lógica de la aplicación y la interacción con el usuario del sistema.

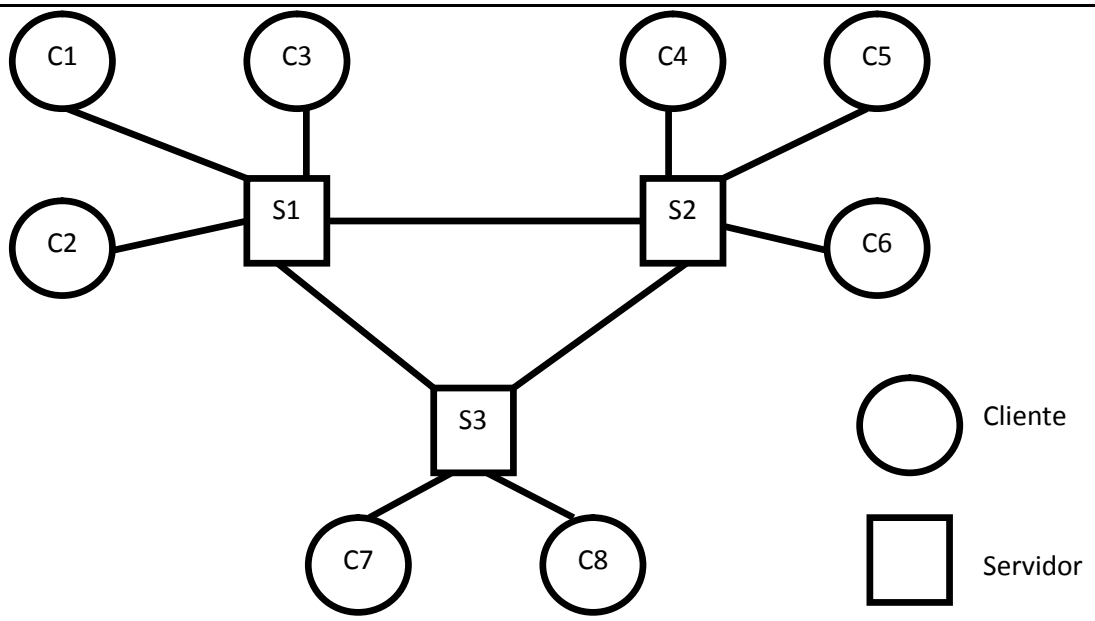


Ilustración 1-10 - Arquitectura cliente - servidor

Puede haber problemas con la escalabilidad y el rendimiento si se elige el modelo de cliente ligero, o problemas de gestión del sistema si se utiliza el modelo de cliente pesado. Para evitar estos problemas, un enfoque alternativo consiste en utilizar una arquitectura cliente-servidor de tres niveles. En esta arquitectura, la presentación, el procesamiento de la aplicación y la gestión de los datos son procesos separados lógicamente además de que se ejecutan en diferentes procesadores.

En algunos casos, es conveniente ampliar el modelo cliente-servidor de tres niveles a la alternativa de varios niveles donde se agregan servidores adicionales al sistema. Los sistemas multicapa se pueden utilizar en aplicaciones que necesitan acceder y utilizar información de diferentes bases de datos. En este caso, un servidor de integración está posicionado entre el servidor de la aplicación y los servidores de bases de datos. El servidor de integración recoge los datos distribuidos y los presenta a la aplicación como si se tratara de una sola base de datos.

1.4.7 Arquitecturas de objetos distribuidos

En el modelo cliente-servidor de un sistema distribuido, los clientes y los servidores son diferentes. Los clientes reciben los servicios de los servidores y no de otros clientes; los servidores pueden actuar como clientes al recibir servicios de otros servidores, pero no solicitan los servicios de clientes; los clientes deben conocer los servicios que son ofrecidos por los servidores y saber cómo comunicarse con estos servidores. Este modelo funciona bien para muchos tipos de aplicaciones.

Un enfoque más general para el diseño de sistemas distribuidos es eliminar la distinción entre cliente y servidor, y diseñar la arquitectura del sistema como una arquitectura distribuida de objetos. En una arquitectura de objetos distribuidos, los componentes fundamentales del sistema son objetos que proporcionan una interfaz para un conjunto de servicios que proporcionan. Otros

objetos llaman a estos servicios sin distinción lógica entre un cliente (receptor de un servicio) y un servidor (proveedor de un servicio).

Los objetos pueden estar distribuidos a través de varias computadoras en una red y comunicarse a través de middleware. Este middleware se denomina agente de petición de objeto. Su rol es el de proveer una interfaz entre objetos. Provee un conjunto de servicios que permiten a los objetos comunicarse y, ser agregados y removidos del sistema. Ilustración 1-11.

Las ventajas del modelo de objetos distribuidos son:

1. Permite que el diseñador del sistema retrasar las decisiones sobre dónde y cómo deben ser proporcionados los servicios. Los objetos proveedores de servicios pueden ejecutarse en cualquier nodo de la red.
2. Es una arquitectura de sistema muy abierta que permite a agregar nuevos recursos cada que es requerido. Existen estándares de comunicación que permiten a los objetos escritos en diferentes lenguajes de programación comunicarse y proveer servicios entre ellos.
3. El sistema es flexible y escalable. Diferentes instancias del sistema con el mismo servicio prestado por diferentes objetos o por objetos replicados se pueden crear para hacer frente a diferentes cargas del sistema. Los nuevos objetos se pueden añadir en el sistema mientras la carga aumenta sin interrumpir a otros objetos del sistema.

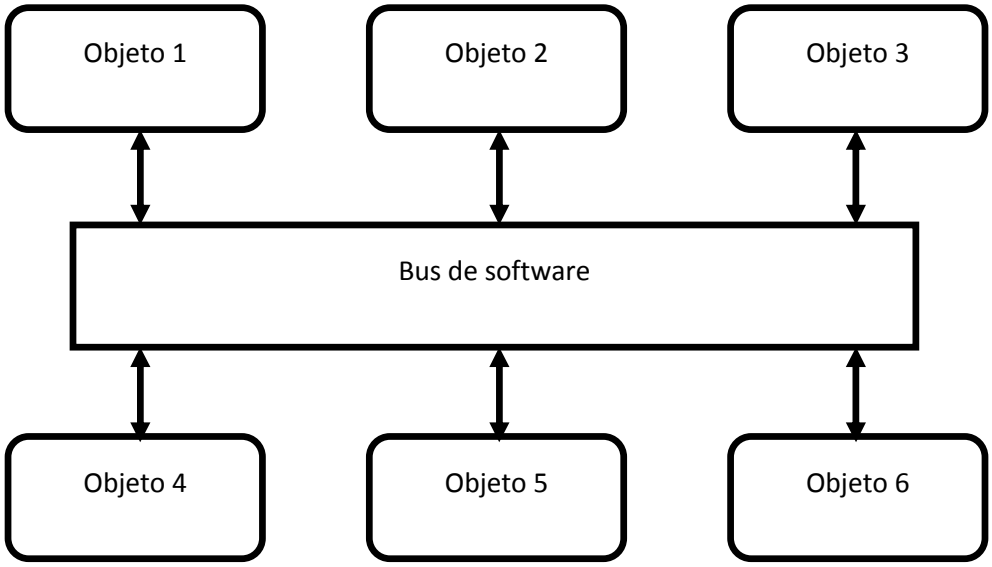


Ilustración 1-11 - Arquitectura de objetos distribuidos

Como se ha indicado previamente, la implementación de una arquitectura de objetos distribuidos requiere un middleware especializado (*Object Request Brokers*) para manejar las comunicaciones entre los objetos distribuidos. En principio, los objetos en el sistema pueden ser implementados utilizando diferentes lenguajes de programación, los objetos pueden ejecutarse en diferentes plataformas y sus nombres no tienen que ser conocidos por todos los demás objetos en el sistema.

El middleware que los une, por tanto, se tiene que encargar de hacer el trabajo para asegurar comunicaciones transparentes entre los objetos.

El middleware para soportar la computación distribuida de objetos es requerido en dos niveles:

- *En el nivel lógico de comunicación*, el middleware proporciona la funcionalidad que permite a los objetos en distintas computadoras el intercambio de datos y el control de información. Estándares tales como CORBA y COM han sido desarrollados para facilitar las comunicaciones entre objetos en diferentes plataformas.
- *A nivel de componentes*, el middleware proporciona una base para el desarrollo de componentes compatibles. Estándares tales como *EJB*, *CORBA* o *Active X* proporcionan una base para la implementación de componentes con métodos estándar que se pueden consultar y ser utilizados por otros componentes.

1.4.8 Arquitecturas peer-to-peer

Los sistemas peer-to-peer (P2P) son sistemas descentralizados, donde los cálculos pueden ser llevados a cabo por cualquier nodo de la red y, al menos en principio, no se realizan distinciones entre clientes y servidores. En aplicaciones peer-to-peer, el sistema global está diseñado para aprovechar el poder de cómputo y almacenamiento disponible a través de una potencialmente grande red de computadoras. Las normas y protocolos que permiten las comunicaciones a través de los nodos están integrados en la propia aplicación y cada nodo debe ejecutar una copia de dicha aplicación. Las tecnologías peer-to-peer han sido ampliamente utilizadas en sistemas personales, por ejemplo sistemas para compartir archivos y de comunicaciones a través de internet. Ilustración 1-12.

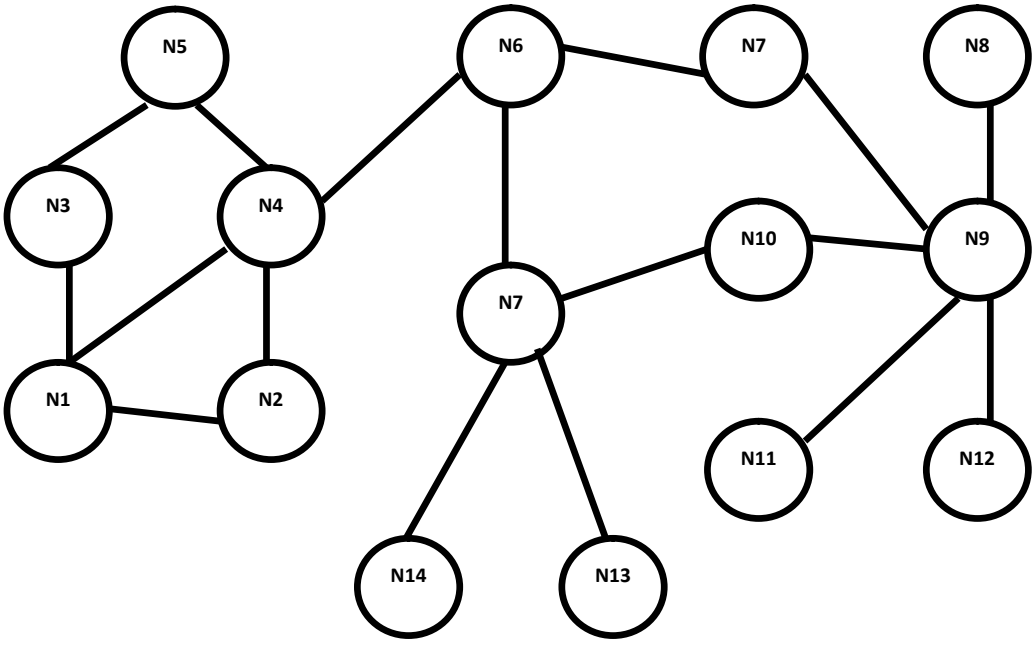


Ilustración 1-12 - Arquitectura peer to peer

En principio, en los sistemas peer-to-peer cada nodo de la red podría ser consciente de todos los demás nodos, podrían hacer conexiones e intercambiar datos entre sí. En la práctica, por supuesto, esto es imposible, por lo que los nodos se organizan en localidades con algunos nodos que actúan como puentes hacia otras localidades de nodos. Es una arquitectura descentralizada.

Esta arquitectura descentralizada tiene ventajas obvias que se encuentran en que es altamente redundante, por lo que es tolerante a fallos y tolerante a la desconexión de nodos en la red. Un modelo de arquitectura p2p alternativa es una arquitectura semi-centralizada donde, dentro de la red, uno o más nodos actúan como servidores para facilitar las comunicaciones entre los nodos.

1.4.9 Arquitectura orientada a servicios (SOA)

La arquitectura orientada a servicios, SOA, surge a partir de la necesidad de desarrollar sistemas que no tengan una alta dependencia de una infraestructura tecnológica o de algún otro sistema. Realizar cambios en un sistema para adaptar una nueva funcionalidad, realizar modificaciones de negocio o cambiar componentes tiene un elevado costo. SOA es un estilo de arquitectura de software basado en la operación conjunta y coordinada de servicios, los cuales son la base de este modelo de arquitectura.

Un servicio es la encapsulación de una funcionalidad de negocio bien definida, diseñado de tal manera que pueda ser reutilizado con facilidad. Un servicio, a su vez, puede estar formado por otros servicios, trabajar de forma distribuida abstrayendo la complejidad asociada a las comunicaciones, incluso puede estar conformado por una o más operaciones de más bajo nivel.

SOA tiene como objetivo el desarrollo de sistemas que tengan la facilidad y la rapidez para soportar cambios tecnológicos o de lógica de negocio, incluso ser capaces de integrarse con sistemas externos o legados, estas características dan como ventaja el ahorro de tiempo y reducción de costos.

Los servicios cuentan con una interfaz en la que describen los parámetros de entrada y de salida, por lo que la implementación queda oculta. La comunicación entre ellos se debe realizar a través del intercambio de mensajes estandarizados con el fin de disminuir el acoplamiento entre los mismos.

En muchas ocasiones cuando se habla de SOA, por error suele relacionarse inmediatamente con servicios web;, si bien esa tecnología brinda un gran soporte para el desarrollo de servicios distribuidos con una implementación final oculta, no son estrictamente necesarios para desarrollar una arquitectura orientada a servicio. Es posible hacer SOA con alguna otra tecnología de comunicación remota o incluso dentro del mismo sistema se puede tener servicios que cumplen con los principios de diseño de la arquitectura. Los principios básicos de una arquitectura orientada a servicio son:

- *Los servicios son reutilizables*. Los sistemas tienen una lógica tal que pueden ser utilizados por múltiples servicios y/o sistemas, independientemente de características como el sistema operativo, lenguaje de programación o arquitectura de hardware.

- *Los servicios comparten un contrato formal*. Un contrato de servicio es la especificación de servicio en la cual se debe mencionar su ubicación, como ser consumido, reglas y características de los métodos que ofrece.
- *Los servicios están débilmente acoplados*. Implica tener poca dependencia entre servicios, de modo que si un servicio sufre un cambio o es sustituido por otro, los servicios dependientes no se vean afectados y presenten los mismos resultados.
- *Los servicios son compuestos*. En ocasiones un servicio puede estar compuesto por otros. De modo que este conjunto de servicios pueden encargarse de resolver una tarea más compleja.

1.4.10 Diseño orientado a objetos

Un sistema orientado a objetos se compone de objetos que interactúan entre ellos, que mantienen su propio estado local y proveen cierta funcionalidad. La representación del estado es privada y no se puede acceder directamente desde el exterior del objeto. El proceso de diseño orientado a objetos implica el diseño de clases de objetos y las relaciones entre estas clases. Estas clases definen los objetos en el sistema y sus interacciones. Cuando el diseño se realiza como un programa en ejecución, los objetos se crean dinámicamente a partir de las definiciones de clases[6]. El proceso completo de desarrollo orientado a objetos esta divido en tres partes:

- *El análisis orientado a objetos* se refiere a la elaboración de un modelo orientado a objetos perteneciente al dominio de la aplicación. Los objetos en ese modelo coincidan con las entidades y operaciones asociadas con el problema que hay que resolver.
- *El diseño orientado a objetos* tiene que ver con el desarrollo de un modelo orientado a objetos de un sistema de software para implementar los requisitos identificados. Los objetos están relacionados a la solución para el problema.
- *La programación orientada a objetos* se refiere a la realización de un software usando un lenguaje de programación orientado a objetos, como Java. Un lenguaje de programación orientado a objetos proporciona constructores para definir clases de objetos y en tiempo de ejecución crea objetos de estas clases.

Los sistemas orientados a objetos son más fáciles de modificar que los sistemas desarrollados utilizando otras alternativas porque los objetos son independientes. Pueden ser tratados y modificados como entidades individuales. Cambiar la implementación de un objeto o añadir nuevas funcionalidades no debe afectar a otros objetos del sistema. Ya que los objetos están relacionados con cosas, algunas veces existe un mapeo claro entre entidades del mundo real y sus objetos equivalentes en el sistema.

Los objetos son, potencialmente, componentes reutilizables porque son encapsulamientos independientes de estados y de operaciones. Los diseños pueden ser desarrollados usando objetos que se han creado en diseños anteriores. Esto reduce los costos del trabajo de diseño, programación y validación. También puede dar lugar a la utilización de objetos de diseño estándar y reducir los riesgos involucrados en el desarrollo de software.

Un objeto es una entidad que tiene un estado y un conjunto definido de operaciones que operan en ese estado. El estado se representa como un conjunto de atributos del objeto. Las operaciones asociadas con el objeto se encargan de prestar servicios a otros objetos (clientes) que solicitan estos servicios cuando se requiere algún cálculo u otra operación.

Los objetos se crean de acuerdo con una definición de clase. Una clase es tanto una especificación de tipo y una plantilla para la creación de objetos. Incluye declaraciones de todos los atributos y las operaciones que deben ser asociados con un objeto de esa clase.

Definición de los casos de uso

Para entender los requerimientos se necesita, en parte, conocer los procesos del dominio y el ambiente externo, o sea los factores externos que participan en los procesos. Dichos procesos de dominio se pueden expresar en casos de uso, es decir, en descripciones narrativas de los procesos del dominio en un formato estructurado de prosa.

Los casos de uso no son un elemento propiamente del análisis y el diseño orientado a objetos; se limitan a describir procesos y pueden ser igualmente eficaces en un proyecto de tecnología no orientada a objetos. No obstante, constituyen un paso preliminar muy útil porque describen las especificaciones de un sistema.

Modelo conceptual

La primera etapa en cualquier proceso de diseño de software es desarrollar un entendimiento de las relaciones entre el software que se está diseñando y su entorno, es decir, el dominio de la aplicación. Para descomponer el dominio del problema hay que identificar los conceptos, los atributos y las asociaciones del dominio que se juzgan importantes. El resultado se puede expresar en un modelo conceptual, el cual se muestra gráficamente en un grupo de diagramas que describen los conceptos (objetos).

El modelo conceptual no es una descripción de los componentes del software; representa los conceptos en el domino del problema en el mundo real.

Diagramas de colaboración

El diseño orientado a objetos tiene por objetivo definir las especificaciones lógicas del software que cumplan con los requisitos funcionales, basándose en la descomposición por clases de objetos. Un paso esencial de esta fase es la asignación de responsabilidades entre los objetos y mostrar cómo interactúan a través de mensajes, expresados en diagramas de colaboración. Estos representan el flujo de mensajes entre las instancias y la invocación de los métodos.

Diseño de clases

Para definir una clase es preciso conocer como se conectan unos objetos a otros y cuáles son los métodos de una clase. Examinando detalladamente los diagramas de colaboración que indican las conexiones necesarias entre objetos y sus métodos, es posible obtener el diagrama de diseño de clases. El diagrama de diseño de clases muestra las clases que han de implementarse en el software.

A diferencia del modelo conceptual, este diagrama no muestra gráficamente conceptos del mundo real, describe únicamente los componentes del software. Para indicar de qué manera los objetos se conectan entre sí a través de atributos, una línea con una flecha en la punta indicara un atributo con una acción.

Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML) se define como "un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software". Es un sistema de notación estándar destinado a los sistemas de modelado que utilizan conceptos orientados a objetos. Es un estándar de la industria para construir modelos orientados a objetos[7].

1.4.11 Diseño de interfaz de usuario

El diseño de un sistema informático abarca un amplio espectro de actividades, desde el hardware hasta el diseño de la interfaz de usuario. Frecuentemente, solo algunas organizaciones emplean diseñadores de interfaces de usuario especializados para su aplicación de software. Por lo tanto, los ingenieros de software a menudo deben hacerse responsables de diseño de interfaz de usuario, así como del diseño del software para implementar esa interfaz.

El diseño cuidadoso de la interfaz de usuario es una parte esencial del proceso global de diseño de software. Para que un sistema de software pueda alcanzar todo su potencial, es esencial que su interfaz de usuario sea diseñada para que coincida con las habilidades, experiencia y expectativas de los usuarios finales. Muchos de los llamados "errores de usuarios" son causados por el hecho de que las interfaces de usuario no consideran las capacidades de los usuarios reales y su entorno de trabajo. Una interfaz de usuario mal diseñada podría significar que los usuarios probablemente no puedan acceder a algunas de las características del sistema, que cometan errores y que sientan que el sistema es un impedimento en lugar de ayudar en la consecución de las tareas para las que utilizan el sistema.

Principios del diseño de interfaz de usuario

Factores humanos como la limitada memoria de corto plazo, la dificultad para procesar demasiada información a la vez, la posibilidad de cometer errores ante situaciones de estrés, distingos rangos de capacidades físicas, inclusive gustos personales, etc, son el fundamento de una serie de principios para el diseño para interfaces de usuario, los cuales se presentan en la tabla 1-1 a continuación:

Tabla 1-1 - Principios de diseño de interfaz de usuario

Principio	Descripción
Familiaridad para el usuario	La interfaz debe usar términos y conceptos a partir de la experiencia de la gente que va a hacer uso del sistema.
Consistencia	La interfaz debe ser consistente en el sentido de que, siempre que sea posible, las operaciones similares se ejecuten de la misma forma.
Sorpresa mínima	Los usuarios no deben ser sorprendidos por el comportamiento del sistema.

Capacidad de recuperación	La interfaz debe contener mecanismos que permitan a los usuarios recuperar de errores.
Asistencia al usuario	La interfaz debe proveer retroalimentación significativa cuando ocurren errores y proveer facilidades de ayuda al usuario.
Diversidad de usuarios	La interfaz debe proveer las facilidades de interacción apropiadas para los diferentes tipos de usuarios del sistema.

El principio de *familiaridad para el usuario* sugiere que no se debe forzar a los usuarios a utilizar una interfaz solo porque es conveniente de implementar. Los términos y objetos a manipular en la aplicación deben estar directamente relacionados al entorno laboral del usuario. Los términos detrás de la implementación de la interfaz deben permanecer ocultos a los usuarios.

El principio de *consistencia de interfaz de usuario* significa que los comandos de sistema y menús deben tener el mismo formato, los parámetros se deben pasar de la misma manera en la misma forma. Las interfaces consistentes reducen el tiempo de aprendizaje de los usuarios. El conocimiento adquirido en un comando es aplicable en el resto de la aplicación o en otras aplicaciones relacionadas. La consistencia a través de diferentes aplicaciones es importante.

El principio de *sorpresa mínima* es apropiado porque los usuarios se molestan cuando una aplicación se comporta de una forma inesperada. Mientras un sistema es utilizado, los usuarios forman un modelo mental de cómo funciona el sistema. Si una acción en un contexto causa un cambio particular, es razonable esperar que dicha acción en otro contexto causara un cambio similar. Pero si algo completamente diferente ocurre, el usuario se sentirá sorprendido y confundido.

El principio de *capacidad de recuperación* es importante porque los usuarios inevitablemente cometen errores cuando se utiliza un sistema. El diseño de la interfaz puede minimizar estos errores (por ejemplo, el uso de menús evita errores de mecanografía), pero los errores no pueden ser nunca completamente eliminados. En consecuencia, se deben incluir las facilidades en la interfaz que permitan a los usuarios recuperarse de sus errores.

Un principio relacionado es el principio de la *asistencia al usuario*. Las interfaces deben contar con facilidades de asistencia de usuario o ayuda. Estos deben integrarse con el sistema y deben proporcionar diferentes niveles de ayuda y asesoramiento. Los niveles deben variar desde información básica hasta una descripción completa de las funcionalidades del sistema. Los sistemas de ayuda deberán estructurarse de manera que los usuarios no se sientan abrumados con la información cuando piden ayuda.

El principio de la *diversidad usuarios* reconoce que, para muchos sistemas interactivos, puede haber diferentes tipos de usuarios. Algunos serán los usuarios ocasionales que interactúan ocasionalmente con el sistema, mientras que otros pueden ser usuarios avanzados que utilizan el sistema durante muchas horas cada día. Los usuarios ocasionales necesitan interfaces que proporcionan orientación, pero los usuarios avanzados requieren accesos directos para que puedan interactuar lo más rápidamente posible. Por otra parte, los usuarios pueden sufrir de

algún tipo de discapacidad física y, si es posible, la interfaz debe ser adaptable para hacer frente a dichas discapacidades.

Proceso de diseño de interfaz de usuario

El proceso de diseño de interfaz de usuario es iterativo porque los usuarios interactúan con los diseñadores y los prototipos de interfaz para decidir sobre las características, la organización y la apariencia de la interfaz de usuario del sistema. A veces, la interfaz se desarrolla como un prototipo por separado en paralelo con otras actividades de ingeniería de software. Más comúnmente, especialmente cuando se utiliza el desarrollo iterativo, el diseño de la interfaz de usuario se crea de forma incremental mientras el software se desarrolla y crece en funcionalidades. El proceso global de diseño de interfaz de usuario se ilustra en la siguiente ilustración 1-13. Hay tres actividades fundamentales en este proceso:

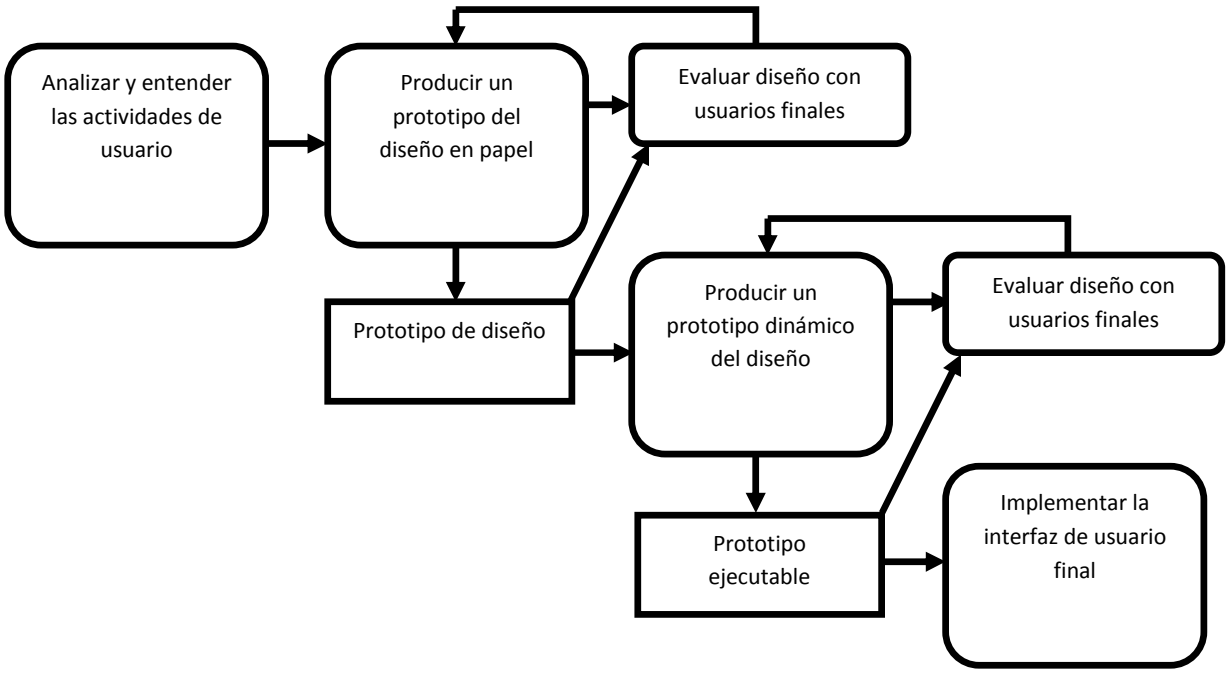


Ilustración 1-13 - Proceso de diseño de interfaz de usuario

1. *Análisis de usuario*. En el proceso de análisis de usuario, se desarrolla el entendimiento de las tareas que el usuario realiza, su entorno de trabajo, los otros sistemas que utiliza, como interactúa con otras personas en su trabajo, etcétera.
2. *Prototipo del sistema*. El diseño y desarrollo de una interfaz de usuario es un proceso iterativo. A pesar de que los usuarios hablen de las funcionalidades que requieren de una interfaz, es muy difícil para ellos ser específicos hasta que pueden ver algo tangible. Por lo tanto, se tienen que desarrollar sistemas prototipo y exponerlos a los usuarios, y así guiar la evolución de la interfaz.
3. *Evaluación de la interfaz*. A pesar de que se tienen discusiones con los usuarios durante el proceso del prototipo, se debe tener una actividad de evaluación más formal donde se recolecte información de los usuarios y sus experiencias con la interfaz.

1.5 Desarrollo de software

El proceso de desarrollo de software se refiere a la creación detallada de software funcional a través de una combinación de codificado en uno o varios lenguajes de programación, verificación del código, pruebas unitarias, pruebas de integración y depuración. Este proceso utiliza como entrada los resultados de los procesos de diseño de software y produce como salida software que debe pasar por la etapa de pruebas. La mayoría del trabajo de diseño se lleva a cabo previamente al proceso de desarrollo, pero cierto trabajo de diseño puede realizarse durante la etapa de desarrollo, por ejemplo si existen cambios en los requerimientos o se ha encontrado algún error de diseño durante el desarrollo de software[8].

1.5.1 Modelos de desarrollo de software

Los modelos de desarrollo son los diversos procesos o metodologías que son seleccionados para el desarrollo del proyecto en función de objetivos y metas del mismo. Hay muchos modelos para el ciclo de vida de desarrollo que se han elaborado con el fin de lograr diferentes objetivos requeridos. Los modelos especifican las diversas etapas del proceso y el orden en que se llevan a cabo.

La selección de modelo tiene muy alto impacto en el tipo de pruebas que se realizarán al software. Definirá el qué, el dónde y el cuándo de las pruebas planeadas, influenciará las pruebas de regresión y en gran medida determina qué técnicas de pruebas se van a usar.

Hay varios modelos o metodologías de desarrollo de software. Se detallarán los siguientes:

1. Modelo Cascada
2. Modelo V
3. Modelo incremental
4. Modelo RAD
5. Modelo Ágil
6. Modelo espiral

La elección de modelo adecuado para el desarrollo del producto o aplicación de software es muy importante. Las pruebas a realizar se basan en el modelo de desarrollo seleccionado.

En las siguientes páginas se revisarán las características generales de cada metodología de desarrollo de software.

Modelo de cascada

El modelo de cascada fue el primer modelo de proceso en ser referenciado. También se le conoce como un modelo de ciclo de vida lineal-secuencial. Es muy sencillo de entender y utilizar. En un modelo de cascada, cada fase debe ser completada antes de que pueda comenzar la fase siguiente. Este tipo de modelo se utiliza básicamente para proyectos que son pequeños y no hay requisitos con incertidumbre o no definidos completamente. Al final de cada fase, una revisión se lleva a cabo para determinar si el proyecto está en el camino correcto y si debe continuar o no, o

ser descartado. En este modelo, la prueba se inicia sólo después de que el desarrollo se ha completado. En el modelo de cascada las fases no se superponen[9]. Ilustración 1-14.

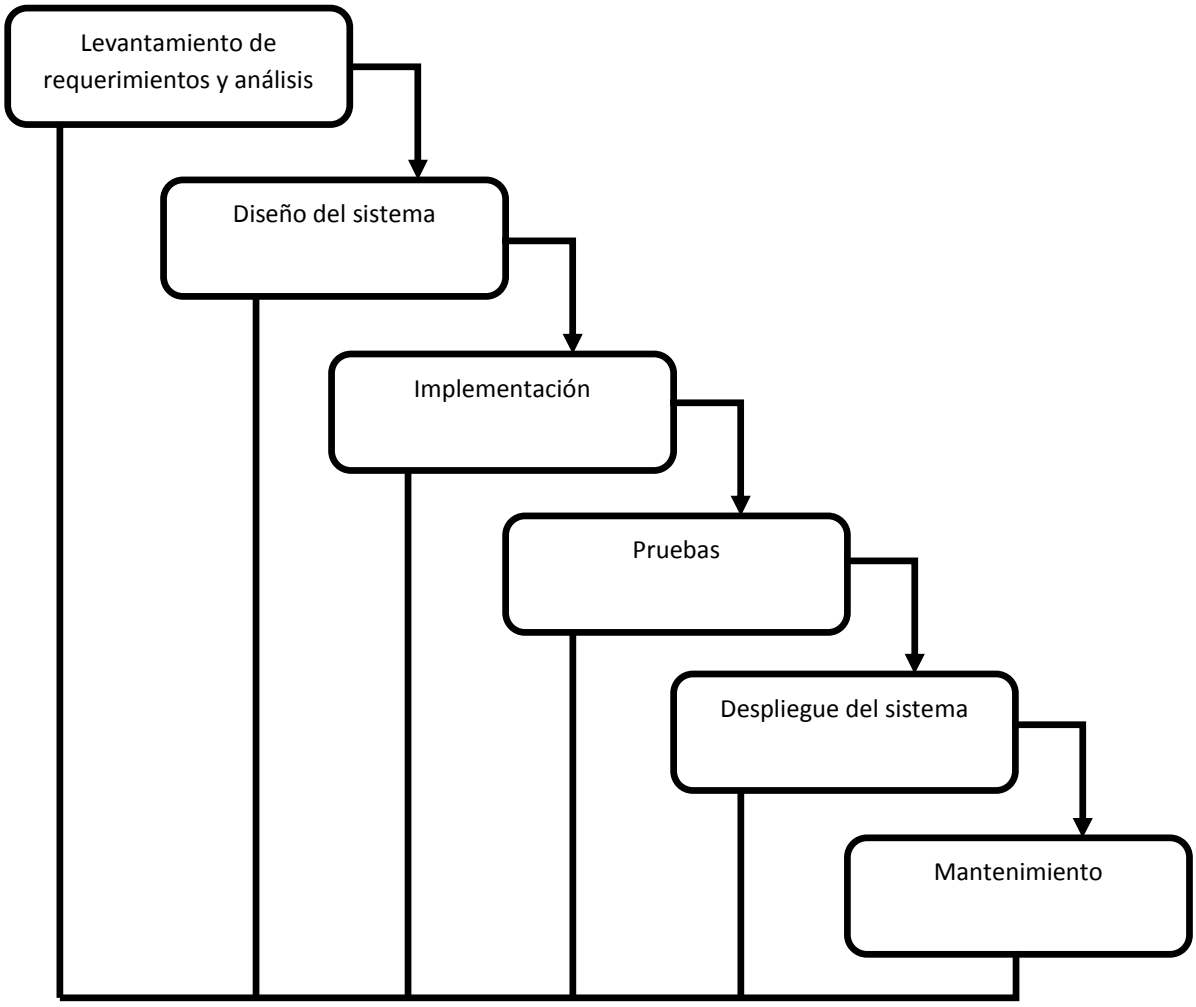


Ilustración 1-14 - Modelo de desarrollo en cascada

Ventajas del modelo en cascada:

- El modelo es simple y fácil de entender y usar.
- Es fácil de administrar debido a la rigidez del modelo, cada fase tiene entregables específicos y un proceso de revisión.
- En el modelo las fases son procesadas y completadas una a la vez, no se superponen.
- El modelo en cascada trabaja bien para proyectos pequeños donde los requerimientos están muy bien entendidos.

Desventajas del modelo en cascada:

- Una vez que la aplicación está en la fase de pruebas, es muy complicado regresar y cambar algo que no fue bien definido en la fase de concepto.

- No se produce software funcional hasta etapas muy tardías del modelo.
- Elevadas cantidades de riesgo e incertidumbre.
- No es un buen modelo para proyectos complejos u orientados a objetos.
- Tampoco es bueno para proyectos largos o que ya se encuentran en proceso.
- No es recomendable para proyectos donde los requerimientos tienen un riesgo moderado de sufrir cambios.

Cuando utilizar el modelo en cascada:

- Este modelo se utiliza sólo cuando los requerimientos son muy bien conocidos, claros y fijos.
- La definición del producto es estable.
- La tecnología se entiende.
- No existen requerimientos ambiguos.
- Existen amplios recursos disponibles con experiencia necesaria.
- El proyecto es corto.

Muy poca interacción con el cliente está involucrada durante en el desarrollo del producto. Una vez que el producto está listo entonces se puede hacer una demostración a los usuarios finales. Una vez que el producto se ha desarrollado, el costo de arreglar problemas que pudieran surgir es muy alto, porque se debe actualizar todo desde la documentación hasta el producto.

Modelo en V

Modelo V significa: Verificación y Validación del modelo. Al igual que el modelo de cascada, el ciclo de la vida en forma de V es un camino secuencial de ejecución de procesos. Cada fase debe ser completada antes de que comience la siguiente fase. Las pruebas del producto están previstas en paralelo con la fase correspondiente del desarrollo. En la ilustración 1-15 se pueden observar las fases de desarrollo y pruebas[10].

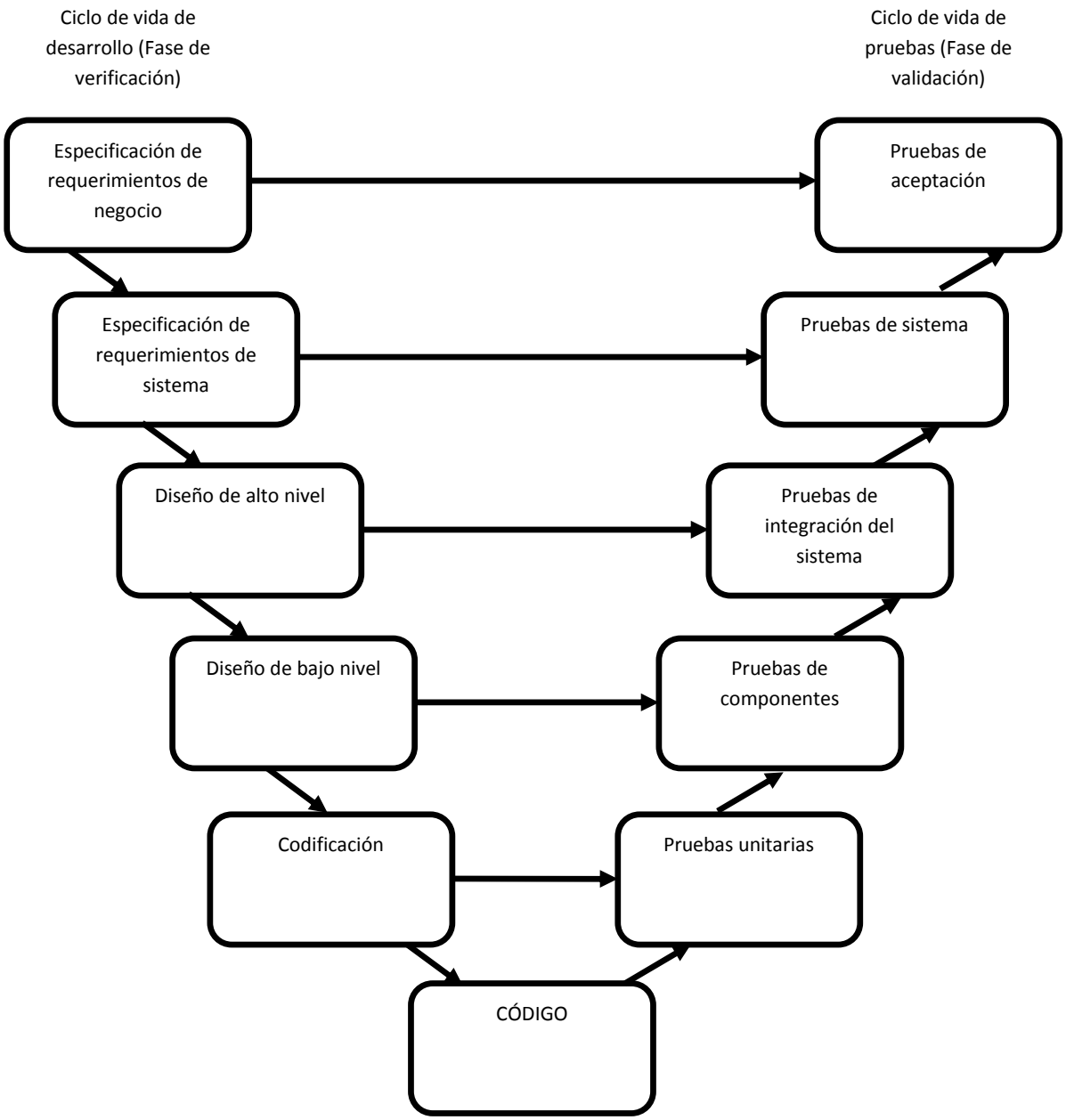


Ilustración 1-15 - Modelo de desarrollo en V

Los requerimientos de negocio y sistema comienzan el ciclo de vida del modelo, al igual que el modelo de cascada. Pero, en este modelo antes de la etapa de desarrollo, se crea un plan de pruebas del sistema. El plan de pruebas se centra en la satisfacción de la funcionalidad especificada en la reunión de los requerimientos.

La fase de diseño de alto nivel se centra en la arquitectura y el diseño del sistema. Se proporciona información general de la solución, la plataforma, el sistema, el producto y el servicio/proceso. Un plan de prueba de integración se crea en esta fase y con el fin de probar la capacidad de los componentes del sistema de software para trabajar conjuntamente.

La fase de diseño de bajo nivel es donde los componentes de software son diseñados. Define la lógica para todos y cada uno de los componentes del sistema. El diagrama de clases con todos los métodos y relación entre las clases se encuentran en esta etapa de diseño. También en esta fase se crean las pruebas de componentes.

La fase de implementación es, de nuevo, donde toda la codificación se lleva a cabo. La codificación está en la parte inferior del modelo en V.

Ventajas del modelo en V:

- Simple y fácil de usar.
- Realizar pruebas en actividades como la planificación, el diseño de pruebas sucede mucho antes de la codificación. Esto ahorra mucho tiempo. Por lo tanto, existen mayores posibilidades de éxito sobre el modelo de cascada.
- Seguimiento de defectos proactiva, las anomalías se encuentran en una etapa temprana.
- Evita el flujo descendente de los defectos.
- Funciona bien para pequeños proyectos en los que se entienden fácilmente los requisitos.

Desventajas del modelo en V:

- Muy rígido y poco flexible.
- El software se desarrolla durante la fase de implementación, por lo que no se producen prototipos de software en etapas tempranas.
- Si los cambios ocurren en medio camino, a continuación, los documentos de prueba, junto con documentos de requisitos tienen que ser actualizados.

Cuándo utilizar el modelo en V:

- Cuando los proyectos son de tamaño pequeño a mediano, donde los requisitos están claramente definidos y establecidos.
- Cuando amplios recursos técnicos están disponibles y tienen los conocimientos técnicos necesarios.

Se requiere una alta confianza de los clientes para elegir el enfoque de modelo en forma de V, ello debido a que no se producen prototipos en etapas tempranas del desarrollo, hay un muy alto riesgo involucrado en el cumplimiento de las expectativas del cliente.

Modelo incremental

En el modelo incremental, el requerimiento completo del sistema se divide en varias partes a ser construidas. Ciclos múltiples de desarrollo tienen lugar aquí, por lo que el ciclo de vida es un ciclo "multi-cascada". Los ciclos se dividen en módulos más pequeños y por lo tanto más fáciles de manejar. Cada módulo pasa a través de las fases de requerimientos, diseño, implementación y pruebas. Una versión funcional del software se produce al terminar el primer módulo, por lo que se entregan versiones funcionales del software desde el principio durante el ciclo de vida del

software. Cada módulo posterior añade nuevas funciones a la versión anterior. El proceso continúa hasta que se alcanza el sistema completo[11]. Ilustración 1-16.

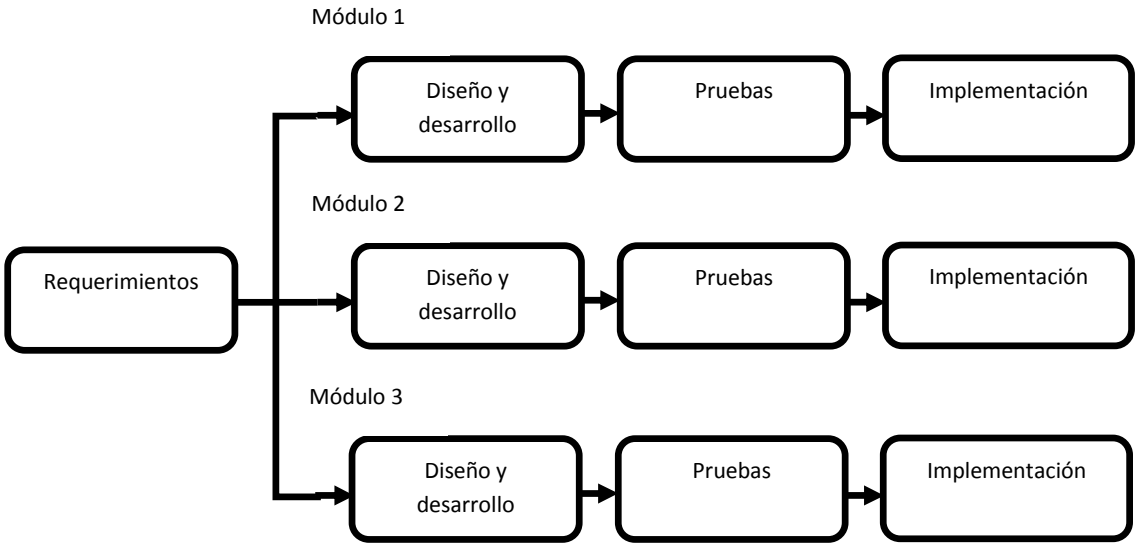


Ilustración 1-16 - Modelo de desarrollo incremental

Ventajas del modelo incremental:

- Genera software que trabaja de forma rápida y temprana durante el ciclo de vida del sistema.
- Este modelo es más flexible, además de que es menos costoso cambiar el alcance y los requerimientos durante el desarrollo.
- Es más fácil de probar y depurar durante cada iteración en comparación con la totalidad del sistema.
- En este modelo el cliente puede ofrecer retroalimentación tras cada etapa concluida.
- Reduce el coste de entrega inicial.
- Es más fácil administrar los riesgos porque son identificados y manejados durante cada iteración.

Desventajas del modelo incremental:

- Necesita una buena planificación y diseño.
- Necesita una definición clara y completa de todo el sistema antes de que pueda ser dividido y construido de forma incremental.
- El costo total es superior a un desarrollo con el modelo de cascada.

Cuando utilizar el modelo incremental:

- Este modelo se puede utilizar cuando los requisitos del sistema completo están claramente definidos y entendidos.

- Los requerimientos mayores deben definirse inicialmente; sin embargo, algunos detalles pueden evolucionar con el tiempo.
- Hay una necesidad de lanzar el producto al mercado rápido.
- Una nueva tecnología se está utilizando en el desarrollo del producto.
- Los recursos con el conjunto de habilidades necesarias no están disponibles.
- Existen características y objetivos de alto riesgo.

Modelo de Desarrollo Rápido de Aplicaciones

El modelo RAD (Rapid Application Development) es un modelo de desarrollo rápido de aplicaciones. Es un tipo de modelo incremental e iterativo que permite la rápida construcción de prototipos para mostrar al cliente. En el modelo RAD se desarrollan los componentes o funciones en paralelo como si fueran pequeños proyectos. Los avances están definidos para terminarse en un tiempo establecido y luego son ensamblados en un prototipo funcional. Esto puede dar rápidamente al cliente algo que ver y usar, además de ofrecer información relacionada con la entrega y sus requisitos[12].

El proceso de desarrollo, ilustración 1-17, comienza con el análisis y diseño de los procesos de negocio utilizando técnicas estructuradas. En la siguiente etapa, los requisitos se construyen, se muestran y se verifican mediante prototipos que, posteriormente, son mejorados. Dichas etapas se repiten iterativamente. Finalmente, se realizan pruebas del sistema completo y se implementa.

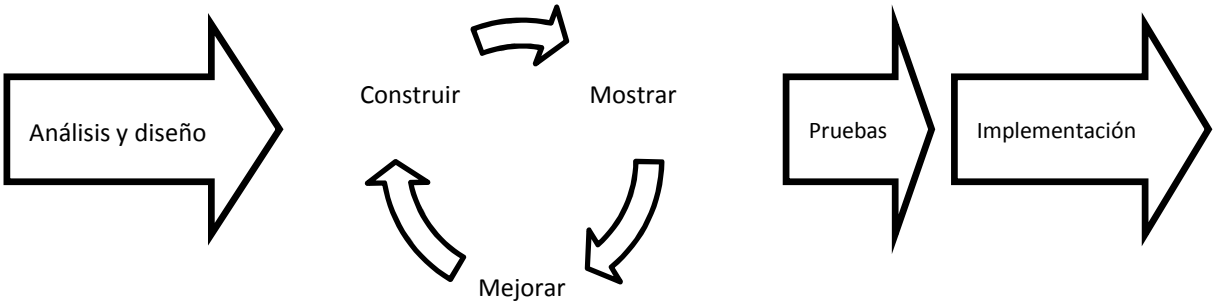


Ilustración 1-17 - Modelo de desarrollo rápido

Ventajas del modelo RAD:

- Reducción del tiempo de desarrollo.
- Aumenta la reutilización de los componentes.
- Se producen revisiones en etapas tempranas del desarrollo.
- Se obtienen comentarios de los clientes.
- Las integraciones desde el principio resuelven muchos de los problemas de la integración final.

Desventajas de modelo RAD:

- Depende de un equipo fuerte y actuaciones individuales para identificar los requerimientos del negocio.

- Solo sistemas que pueden ser separados en módulos pueden ser construidos usando RAD.
- Requiere desarrolladores y diseñadores altamente cualificados.
- Alta dependencia de las habilidades de modelado.

Cuando utilizar el modelo RAD:

- RAD debe utilizarse cuando hay una necesidad de crear un sistema que debe ser dividido en módulos y completarse de dos a tres meses de tiempo.
- Se debe utilizar si hay una alta disponibilidad de diseñadores para el modelado y el presupuesto es lo suficientemente alto como para pagar su costo junto con el costo de las herramientas para generar códigos automatizados.
- Se debe utilizar solo si existen recursos con un elevado conocimiento de las reglas de negocio y que puedan cumplir en producir el sistema en un corto periodo de tiempo (dos a tres meses).

Modelo Ágil

Modelo de desarrollo ágil es también un tipo de modelo incremental. El software se desarrolla en ciclos incrementales y rápidos llamados iteraciones (Ilustración 1-18). Esto se traduce en pequeñas incrementos en la funcionalidad anterior tras cada liberación. Cada versión se prueba a fondo para asegurar que la calidad del software se mantiene. Se utiliza para aplicaciones de tiempo crítico. *Extreme Programming (XP)* y *Scrum* son actualmente los modelos de ciclo de vida de desarrollo ágil más conocidos[13].

El modelo de desarrollo ágil de software es un conjunto de métodos de desarrollo de software en el que las necesidades y soluciones evolucionan a través de la colaboración de equipos auto-organizados y multifuncionales. Promueve la planificación adaptativa, el desarrollo evolutivo, liberaciones tempranas, la mejora continua y además promueve una respuesta rápida y flexible a los cambios.

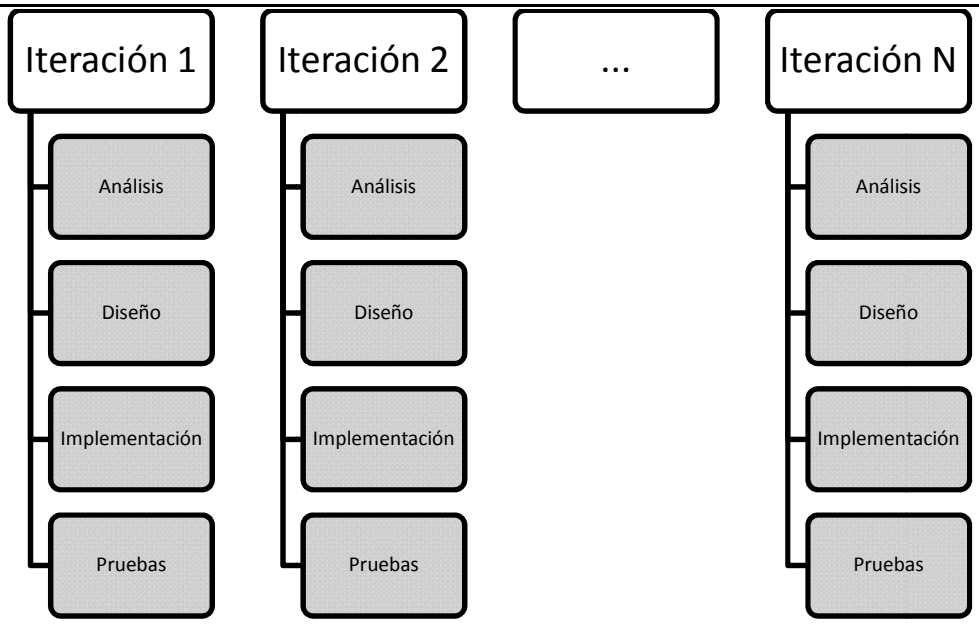


Ilustración 1-18 - Modelo de desarrollo ágil

Ventajas del modelo de desarrollo ágil:

- La satisfacción del cliente debido a la rápida y continua entrega de software útil.
- Las personas y las interacciones se enfatizan en lugar de procesos y herramientas.; los clientes, desarrolladores y testers constantemente interactúan entre sí.
- Liberaciones de software funcional se entregan con frecuencia (semanas en lugar de meses).
- La conversación cara a cara es la mejor forma de comunicación.
- La estrecha cooperación, todos los días entre la gente de negocios y desarrolladores.
- Adaptación a las circunstancias cambiantes.
- Incluso los cambios de última hora en los requisitos son bienvenidos

Desventajas del modelo ágil:

- En el caso de algunos entregables de software, especialmente los grandes, es difícil evaluar el esfuerzo requerido al comienzo del ciclo de vida de desarrollo del software.
- Hay una falta de énfasis en el diseño y la documentación necesaria.
- El proyecto puede fácilmente tomar otro rumbo si el representante del cliente no tiene claro cuál es el resultado final que el cliente espera.
- Sólo los programadores experimentados son capaces de tomar el tipo de decisiones necesarias durante el proceso de desarrollo. Por lo tanto, no hay lugar para los programadores novatos, a menos que se combine su trabajo con recursos más experimentados.

Cuando utilizar el modelo ágil:

- Cuando se necesitan implementar nuevos cambios. La libertad que el modelo ágil da al cambio es muy importante. Nuevos cambios pueden aplicarse a un costo muy bajo, debido a la frecuencia de nuevos incrementos que se producen.
- Para implementar una nueva característica, los desarrolladores necesitan perder sólo el trabajo de unos pocos días o incluso sólo unas horas.
- A diferencia del modelo de cascada, en el modelo ágil se requiere una planificación muy limitada para empezar con el proyecto. El modelo ágil asume que las necesidades de los usuarios finales son siempre cambiantes en el dinámico mundo de los negocios y de TI. Los cambios pueden ser discutidos y las nuevas características pueden ser recién aplicadas o removidas basados en la retroalimentación con el cliente. Esto da efectivamente al cliente el sistema que quieren o necesitan.
- Tanto los desarrolladores de sistemas y partes interesadas por igual, descubren que también obtienen más libertad de tiempo y opciones que si el software fuese desarrollado de una manera secuencial más rígida.

Modelo en Espiral

El modelo de desarrollo en espiral es un conjunto de procesos impulsados por el riesgo para generar proyectos de software. Con base en los patrones de riesgo de un proyecto dado, el modelo espiral guía a un equipo para adoptar elementos de uno o más modelos de desarrollo, tales como los mencionados anteriormente[14].

El modelo en espiral es similar al modelo incremental, pero con más énfasis en el análisis de riesgos. El modelo en espiral tiene cuatro fases: planificación, análisis de riesgos, ingeniería y evaluación. Un proyecto de software pasa repetidamente a través de estas fases en iteraciones (llamados espirales en este modelo). Siguiendo la línea de la espiral base, a partir de la fase de planificación, los requerimientos son reunidos y el riesgo se evalúa. Las espirales posteriores se basan en la espiral de la línea de base. Ver ilustración 1-19 a continuación.

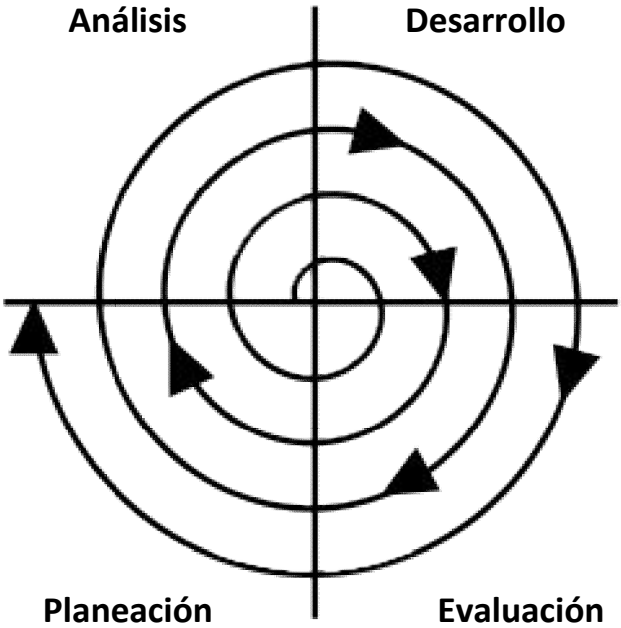


Ilustración 1-19 - Modelo de desarrollo en espiral

Fase de planificación: Los requerimientos se obtienen durante la fase de planificación. Los requerimientos son del tipo de negocio y de sistema.

Fase de análisis de riesgos: Se realiza un proceso de identificación de riesgos y soluciones alternativas. Un prototipo se produce al final de la fase de análisis de riesgos. Si se encuentra algún riesgo durante esta etapa, a continuación se presentan soluciones alternas y son implementadas.

Fase de ingeniería: En esta fase se desarrolla el software y también las pruebas que se van a ejecutar al final de la fase.

Fase de evaluación: Esta fase permite al cliente evaluar el estado actual del proyecto antes de continuar con la siguiente espiral.

Ventajas del modelo de espiral:

- Gran cantidad de análisis de riesgos.
- Bueno para proyectos grandes y de misión crítica.
- Fuerte control en las aprobaciones del software y de la documentación.
- Funcionalidad adicional se puede añadir en una fecha posterior.
- Software se produce temprano en el ciclo de vida.

Desventajas del modelo espiral:

- Puede ser un modelo costoso de utilizar.

- El análisis de riesgos exige una experiencia muy específica.
- El éxito del proyecto depende en gran medida la fase de análisis de riesgos.
- No funciona bien para los proyectos más pequeños.

Cuándo utilizar el modelo espiral:

- Cuando los costos y la evaluación del riesgo son importantes.
- Para proyectos de medio a alto riesgo
- Existe compromiso a largo plazo con el proyecto debido a los posibles cambios en las prioridades económicas.
- Los usuarios están seguros de sus necesidades.
- Los requerimientos son complejos.

1.5.2 Reutilización de software

El proceso de diseño en la mayoría de las disciplinas de ingeniería se basa en la reutilización de sistemas o componentes existentes. Los ingenieros mecánicos o eléctricos normalmente no especifican un diseño en el que cada componente tiene que ser fabricados especialmente, basan su diseño en los componentes que han sido probados en otros sistemas. Estos no son sólo componentes pequeños tales como bridas y válvulas, también incluyen los principales subsistemas tales como motores, condensadores o turbinas.

La construcción de software con base en la reutilización es una estrategia de ingeniería de software donde el proceso de desarrollo está orientado a la reutilización de software existente. Cada vez más empresas consideran a su software como un activo valioso y están promoviendo la reutilización como forma de aumentar el retorno de su inversión en software.

Las unidades de software que se reutilizan pueden ser radicalmente de diferentes tamaños, por ejemplo:

- *Reutilizar un sistema o aplicación*. Un sistema o aplicación completo puede ser reutilizado sin incorporar cambios en otros sistemas, mediante la configuración de la propia aplicación para diferentes clientes o mediante el desarrollo de familias de aplicaciones que tengan una arquitectura común, pero están adaptadas para clientes específicos.
- *Reutilización de componentes*. La reutilización de componentes de una aplicación cuyo tamaño oscila entre los subsistemas hasta los objetos individuales.
- *Reutilización de objetos y funciones*. La reutilización de componentes de software que implementan una sola función, tales como una función matemática o una clase de objeto, pueden ser reutilizados. Esta forma de reutilización, en torno a las bibliotecas estándar, ha sido común desde hace mucho tiempo. Muchas bibliotecas de funciones y clases para diferentes tipos de aplicaciones y plataformas de desarrollo están disponibles. Estos pueden ser usados fácilmente vinculándolos con otro código de la aplicación.

Los sistemas y componentes de software son entidades específicas reutilizables, pero su naturaleza específica a veces implica que es caro modificarlos para una nueva situación. El

concepto de reutilización puede ser realizado a través enfoques tales como patrones de diseño, sistemas configurables y generadores de programas. Se enlistan los beneficios en la siguiente tabla 1-2.

Tabla 1-2 - Beneficios de la reutilización del software

Beneficios	Explicación
Mayor fiabilidad	El software reutilizado, que ha sido probado y comprobado en sistemas funcionando, debería ser más confiable que un desarrollo nuevo, ya que sus fallas en diseño e implementación han sido encontradas y corregidas.
Reducción de riesgo en el proceso	El coste del software existente ya se conoce, mientras que el costo de un nuevo desarrollo es variable.
Uso efectivo de especialistas	En lugar de hacer el mismo trabajo una y otra vez, éstos especialistas pueden desarrollar software reutilizable que encapsule su conocimiento.
Cumplimiento de estándares	Algunos estándares, como los estándares de interfaz de usuario, pueden ser implementados como un conjunto de componentes estándar reutilizables.
Desarrollo acelerado	Llevar un sistema al mercado tan pronto como sea posible es a menudo más importante que los costos generales de desarrollo. La reutilización de software puede acelerar la producción del sistema porque el tiempo de desarrollo y la validación pueden ser reducidos.

Sin embargo, también hay costes y problemas asociados con la reutilización (Tabla 1-3). En particular, hay un coste significativo asociado con la comprensión de si un componente es adecuado para su reutilización en una situación particular y en la prueba de ese componente para asegurar su fiabilidad. Estos costos adicionales pueden inhibir la reutilización de software o que las reducciones en el coste global del desarrollo sean menores de lo previsto.

Tabla 1-3 - Problemas de la reutilización de software

Problemas	Explicación
Incremento en los costos de mantenimiento	Si el código fuente de un sistema de software reutilizado no se encuentra disponible, se incrementarán los costos de mantenimiento porque dichos componentes pueden volverse incompatibles con futuros cambios en el sistema.
Falta de soporte	Ciertos componentes pueden no ser reutilizables del todo, por lo que puede ser imposible integrar dichos componentes con algún elemento del sistema.
Síndrome de "No inventado	Algunos ingenieros de software prefieren reescribir

aquí"	componentes porque creen que pueden mejorarlos, principalmente debido a que existe un mayor reto que utilizar los componentes de alguien más.
Crear y mantener una biblioteca de componentes	Crear una biblioteca de componentes reutilizables y asegurarse de que los desarrolladores de software la utilicen puede ser caro.
Encontrar, entender y adaptar componentes reutilizables	Los componentes en una biblioteca deben ser descubiertos, entendidos y, a veces, adaptados para que funcionen en un nuevo ambiente.

1.5.3 Mantenimiento de software

El mantenimiento del software es el proceso general de cambiar un sistema después de que ha sido entregado. El término se aplica generalmente para software a la medida en el que grupos separados de desarrollo están involucrados antes y después de la entrega. Los cambios realizados en el software pueden ser simples cambios para corregir errores de codificación, cambios más extensos para corregir errores de diseño, mejoras significativas para corregir los errores de especificación o introducir nuevos requerimientos. Los cambios se implementan mediante la modificación de componentes existentes del sistema y, en su caso, mediante la adición de nuevos componentes al sistema.

Hay tres tipos diferentes de mantenimiento de software:

1. *Mantenimiento para reparar errores de software*. Los errores de codificación suelen ser relativamente baratos de corregir; los errores de diseño son más caros, ya que pueden implicar reescribir varios componentes del programa. Errores en los requerimientos son los más caros de reparar debido al extenso rediseño del sistema que puede ser necesario.
2. *Mantenimiento de adaptar el software a un entorno operativo diferente.* Este tipo de mantenimiento se requiere cuando algún aspecto del entorno del sistema tales como el hardware, el sistema operativo u otro software de soporte cambian. La aplicación debe ser modificada para adaptarse y hacer frente a los cambios en el entorno.
3. *Mantenimiento para agregar o modificar la funcionalidad del sistema.* Este tipo de mantenimiento es necesario cuando los requisitos del sistema cambian en respuesta a los cambios organizativos o de negocios. La magnitud de los cambios necesarios para el software es a menudo mucho mayor que para los otros tipos de mantenimiento.

En la práctica, no hay una distinción clara entre estos tipos de mantenimiento. Al adaptar el sistema a un nuevo entorno, se pueden agregar nuevas funcionalidades para aprovechar las nuevas características del entorno. Las fallas en el software se exponen a menudo porque los usuarios utilizan el sistema de manera imprevista. Cambiar el sistema para adaptarse a la forma de trabajar de los usuarios es la mejor manera de solucionar este tipo de fallos.

Este tipo de mantenimientos son generalmente reconocibles, pero algunas personas pueden utilizar diferentes nombres. El *mantenimiento correctivo* se utiliza universalmente para referirse al mantenimiento de reparación de fallas. Sin embargo, el *mantenimiento adaptativo* significa que a veces la adaptación a un nuevo entorno y puede significar tener que cumplir nuevos requerimientos de software. *Mantenimiento perfectivo* puede significar perfeccionar el software mediante la implementación de nuevos requerimientos; en otros casos significa mantener la misma funcionalidad del sistema, pero mejorando su estructura y su funcionamiento.

1.6 Verificación y validación

Durante y después del proceso de implementación, el software que se desarrolla debe ser evaluado para verificar que cumple con su especificación y proporciona la funcionalidad esperada por las personas que pagan por el software. Verificación y validación es el nombre dado a estos procesos de control y de análisis. Las actividades de validación y verificación se llevan a cabo en cada etapa del proceso de software. La verificación y validación comienzan con las revisiones de requerimientos y continúan a través de las revisiones de diseño e inspecciones de código y hasta las pruebas de producto[15].

Verificación y validación no son la misma cosa, a pesar de que a menudo se confunden:

- *Validación*: ¿Estamos construyendo el producto correcto?
- *Verificación*: ¿Estamos construyendo el producto correctamente?

Estas definiciones nos dicen que el papel de la verificación consiste en comprobar que el software cumple con su especificación. Se debe verificar que cumple con los requerimientos funcionales y no funcionales. La validación, sin embargo, es un proceso más general; el objetivo de la validación es garantizar que el sistema de software cumple con las expectativas del cliente.

Los procesos de verificación y validación tienen como objetivo establecer la existencia de defectos en el sistema de software; la depuración es el proceso que localiza y corrige esos defectos.

La localización de los fallos en un programa no es siempre un proceso simple, ya que el error puede no encontrarse cerca del punto donde el programa ha fallado. Para localizar un fallo de programa, puede ser necesario diseñar pruebas adicionales que reproduzcan el fallo original y con ello obtener su ubicación exacta en el programa. Se pueden utilizar herramientas de depuración que recopilan información sobre la ejecución del programa y que también pueden ayudar a localizar el origen de un problema.

Las herramientas de depuración interactivas son generalmente parte de un conjunto de herramientas de los entornos de desarrollo de muchos lenguajes de programación y que también se integran con un sistema de compilación. Proporcionan un ambiente especializado en tiempo de ejecución para el programa en desarrollo que permite el acceso a la tabla de símbolos del compilador y, desde allí, a los valores de las variables del programa. Se puede controlar la ejecución del programa para hacer pausas controladas y así poder examinar los valores de las variables y descubrir la ubicación del fallo.

Después de que un defecto en el programa ha sido descubierto, se debe corregir y revalidar el sistema. Esto puede implicar la reinspección del programa con pruebas de regresión, donde las pruebas existentes se ejecutan de nuevo. Las pruebas de regresión se utilizan para comprobar que los cambios realizados en un programa no han introducido nuevos fallos. La experiencia ha demostrado que una alta proporción de "reparaciones" de fallas son incompletas o introducen nuevas fallas en el programa.

En principio, se debe repetir todas las pruebas después de cada reparación de defectos; en la práctica, esto suele ser demasiado caro. Como parte del plan de pruebas, es necesario identificar las dependencias entre los componentes y las pruebas asociadas con cada componente, así únicamente se realizaran las pruebas asociadas con los componentes modificados y sus dependencias.

1.7 Pruebas de software

El proceso general de pruebas inicia con las pruebas de componentes individuales del programa como funciones y objetos. Esos componentes son integrados en subsistemas y sistemas sobre los que se realizan las pruebas necesarias entre las interacciones de dichos componentes. Finalmente, después de la entrega del software, el cliente puede llevar a cabo una serie de pruebas de aceptación para corroborar que el sistema se ejecuta como esta especificado[16].

Este modelo del proceso de pruebas es apropiado para el desarrollo de grandes sistemas, pero para sistemas más pequeños puede haber menos pasos en el proceso. Las dos actividades fundamentales del proceso de pruebas de software son las pruebas de componentes y las pruebas del sistema completo. Ilustración 1-20.

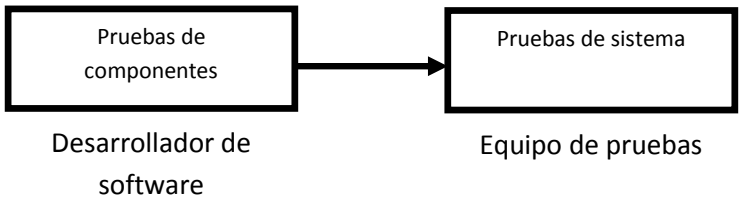


Ilustración 1-20 - Pruebas de software

El proceso de pruebas de software tiene dos metas:

1. *Demostrar al desarrollador y al cliente que el sistema cumple con los requerimientos.* Para software a la medida, esto significa que debe existir al menos una prueba para todo requerimiento que se encuentre en los documentos de requerimientos de usuario y de sistema. Para software más sencillo se pueden hacer pruebas de las características generales.
2. *Descubrir fallas o defectos donde el comportamiento del software es incorrecto.* Las pruebas de defectos se basan en encontrar toda clase de comportamiento indeseable o inesperado del sistema, por ejemplo caídas del sistema, interacciones no solicitadas con otros sistemas, cálculos incorrectos y corrupción de datos.

Las pruebas no pueden demostrar que el software está libre de defectos o se va a comportar correctamente en toda circunstancia; después de todo, la meta de las pruebas de software es convencer a los desarrolladores y clientes de que el sistema es suficientemente bueno para la operar.

Un modelo general del proceso de pruebas se presenta a continuación en la ilustración 1-21. Los casos de pruebas son las especificaciones de entrada para las pruebas y la salida esperada del sistema, además de un enunciado o descripción de qué es lo que se va a probar. Los datos de prueba son las entradas ideadas para probar el sistema. Los datos de prueba se pueden generar automáticamente en algunas ocasiones.

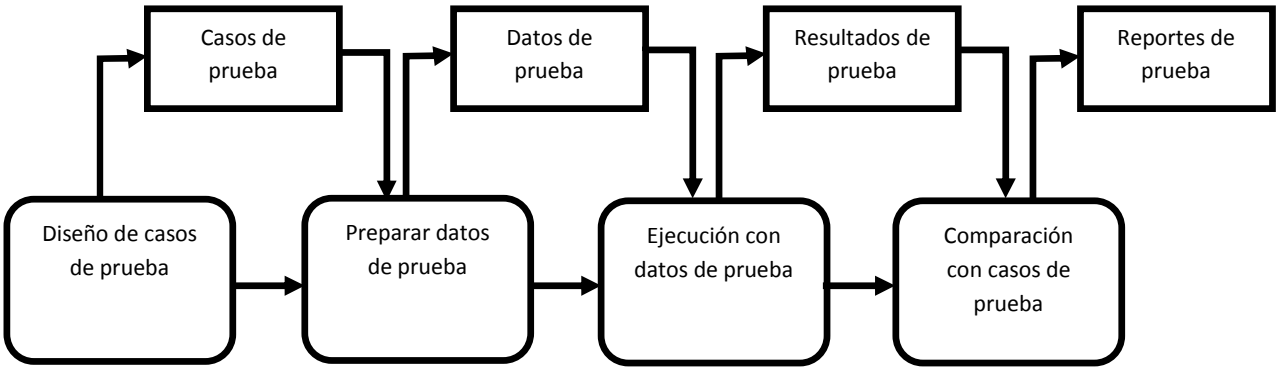


Ilustración 1-21 - Proceso de pruebas de software

1.7.1 Pruebas de sistema

Las pruebas del sistema involucran la integración de dos o más componentes que implementan las funciones o características del sistema para posteriormente probar el sistema integrado. En un proceso de desarrollo iterativo, las pruebas del sistema se refieren a las pruebas realizadas sobre un incremento a ser entregado al cliente; en un proceso en cascada, las pruebas del sistema se ocupan de probar el sistema completo.

Para la mayoría de los sistemas complejos, hay dos fases distintas a las pruebas del sistema:

1. *Pruebas de integración*, cuando el equipo de pruebas tiene acceso al código fuente del sistema. Cuando se descubre un problema, el equipo de integración intenta encontrar el origen del problema e identificar los componentes que tienen que ser depurados. Las pruebas de integración se refieren principalmente a la búsqueda de defectos en el sistema.
2. *Pruebas de liberación*, cuando una nueva versión del sistema podría ser liberado a los usuarios se prueba. El equipo de pruebas valida que el sistema cumple con sus requerimientos y con la garantía de que el sistema es fiable. Las pruebas de liberación suelen ser pruebas “caja negra” en las que el equipo de pruebas simplemente demuestra que el sistema funciona o no funciona correctamente. Los problemas son reportados al equipo de desarrollo cuyo trabajo es depurar el programa. Los casos donde los clientes están involucrados en las pruebas de liberación, a veces se llaman pruebas de aceptación.

Si la liberación del sistema es lo suficientemente buena, el cliente puede entonces aceptarla para su uso.

1.7.2 Pruebas de componentes

La prueba de componentes (a veces llamadas pruebas unitarias) es el proceso de pruebas de componentes individuales del sistema. Este es un proceso de pruebas de defectos por lo que su objetivo es exponer fallos en estos componentes. Para la mayoría de los sistemas, los desarrolladores de los componentes son los responsables de las pruebas de dichos componentes.

Hay diferentes tipos de componentes que pueden ser probados en esta etapa:

1. Funciones individuales o métodos de un objeto.
2. Clases de objetos que tienen varios atributos y métodos.
3. Componentes compuestos formados por varios objetos o funciones diferentes.

Las funciones o métodos individuales son el tipo más sencillo de los componentes y las pruebas consisten en un conjunto de llamadas a estas rutinas con diferentes parámetros de entrada.

Al probar las clases de objetos, se deben diseñar sus pruebas para proporcionar cobertura de todas las características del objeto. Por lo tanto, las pruebas de las clases objetos deben incluir:

1. Pruebas en aislamiento de todas las operaciones asociadas con el objeto.
2. Las asignaciones y las llamadas de todos los atributos asociados con el objeto.
3. El ejercicio del objeto en todos los estados posibles. Esto significa que todos los eventos que pueden causar un cambio de estado en el objeto deben ser simulados.

Si se utiliza la herencia, esto hace que sea más difícil diseñar pruebas de clase de objeto. Cuando una superclase proporciona operaciones que se heredan por un número de subclases, deben hacerse las pruebas para todas estas subclases con todas las operaciones heredadas. La razón de esto es que la operación de herencia puede hacer suposiciones acerca de otras operaciones y atributos, y estos pueden haber sido cambiados cuando se hereda. Igualmente, cuando una operación de superclase se anula entonces la operación de sobrescrita debe ser probada.

1.7.3 Diseño de casos de pruebas

El diseño de casos de prueba es una parte del proceso de pruebas del sistema y de sus componentes en el cual se diseñan las entradas y salidas previstas que ponen a prueba las características del sistema. La meta de este proceso es crear un conjunto de casos de pruebas que son efectivos en descubrir defectos del software y mostrar que el sistema cumple con sus requerimientos.

Para diseñar un caso de pruebas, se selecciona una característica del sistema o del componente que se desea probar. Entonces, se selecciona una serie de entradas que ejecutan dicha característica, se documentan las salidas y/o rangos esperados a la salida y, cuando sea posible, se diseña una verificación automática de que los resultados reales de las pruebas y los resultados esperados son los mismos.

Hay varios métodos que se pueden seguir para el diseño de casos de prueba:

1. *Las pruebas basadas en requerimientos,* es donde los casos de prueba están diseñados para poner a prueba los requerimientos del sistema. Es mayormente usado en la etapa de pruebas del sistema porque los requerimientos usualmente se implementan por varios componentes juntos. Para cada requerimiento, se identifican los casos de prueba que puedan demostrar que el sistema cumple con ese requerimiento.
2. *Las pruebas de partición*, es donde se identifican particiones de entrada y salida, y se diseñan pruebas de modo que el sistema ejecuta las entradas de todas las particiones y genera salidas de todas las particiones. Las particiones son grupos de datos que tienen características comunes tales como todos los números negativos, todos los nombres de menos de 30 caracteres, todos los eventos que surgen de la elección de los elementos de un menú, y así sucesivamente.
3. *Las pruebas estructurales,* es donde se usa el conocimiento de la estructura del programa para diseñar pruebas que ejecutan todas las partes del programa. Esencialmente, cuando se prueba un programa, se debe ejecutar cada sentencia al menos una vez; las pruebas estructurales ayudan a identificar casos de prueba que permiten hacer esto posible.

En general, en el diseño de casos de prueba, se debe comenzar con las pruebas de los requerimientos de más alto nivel y a continuación, añadir progresivamente pruebas más detalladas usando las pruebas de partición y las pruebas estructurales.

CAPÍTULO 2 - DISPOSITIVOS MÓVILES CON SISTEMA OPERATIVO ANDROID

2.1 Introducción al cómputo móvil

El término cómputo móvil se refiere a la interacción humano-computadora en la cual la computadora es transportada durante su uso. El cómputo móvil involucra comunicaciones móviles, dispositivos o hardware móvil y software móvil.

Actualmente, las comunicaciones móviles son accesibles para un gran número de personas principalmente a través de redes inalámbricas en sus hogares y oficinas, con redes inalámbricas abiertas en lugares públicos concurridos y también a través de compañías de telefonía celular. Inclusive, es posible compartir las conexiones anteriormente mencionadas desde un dispositivo móvil a otros dispositivos cercanos que por alguna limitación no puedan acceder a alguna de las redes citadas en un momento o circunstancia particular.

El hardware móvil se puede categorizar en tres clases diferentes:

- *Computadoras portátiles.* Son versiones de computadoras personales compactadas y de bajo peso que pueden ejecutar sistemas operativos de escritorio completos como las distintas versiones de Windows, Linux o Mac OS, por ejemplo: laptops o notebooks, ultrabooks y netbooks.
- *Teléfonos móviles y sus derivados.* Inicialmente la función básica de estos dispositivos fue la comunicación por voz, como es el caso de los celulares y los teléfonos inteligentes. Sin embargo, dicha funcionalidad ha pasado a segundo término debido a la proliferación de servicios que permiten comunicarse a través de internet y no necesariamente a través de una línea telefónica convencional. También en este segmento se enumeran las tabletas.
- *Computadoras vestibles.* Originalmente conocidos como *weareables,* son dispositivos electrónicos que pueden encontrarse en la ropa o directamente en el cuerpo del usuario, por ejemplo en relojes, pulseras, collares e implantes.

Con tal diversidad de dispositivos, actualmente es común encontrar software especializado en una o varías de las plataformas antes citadas según las características de cada una. Las computadoras portátiles continúan siendo un medio común para labores productivas como trabajo de oficina, creación de contenidos, entretenimiento demandante en recursos de hardware, etc. Los teléfonos móviles tienen a su disposición software para realizar prácticamente en tiempo real tareas como las comunicaciones y obtención de información relevante basada en el perfil del usuario, además del entretenimiento casual. Finalmente, los dispositivos vestibles están más enfocados a la obtención de información del usuario, principalmente indicadores biométricos y de localización geográfica que pueden ser procesados y analizados en dispositivos de mayor tamaño y capacidad como teléfonos inteligentes o computadoras personales.

Es de esperarse que en general la distinción entre clases de hardware móvil se mantenga porque ninguna de las tecnologías previamente citadas reemplaza a las otras, si no que son complementarías entre sí y en muchos escenarios es claro que cada clase tiene ventajas específicas sobre las otras tecnologías.

2.2 Arquitectura de hardware para el sistema operativo Android

La aplicación desarrollada para el presente trabajo está dirigida al sistema operativo Android, particularmente para la gama de teléfonos inteligentes y tabletas, aunque los esfuerzos de dicho sistema operativo actualmente se enfocan a una variada gama de dispositivos que, como se puede ver, no necesariamente son móviles o llevados por el usuario:

- Teléfonos inteligentes y tabletas.
- Relojes.
- Televisión.
- Automóviles.

En general, una computadora se define como un dispositivo electrónico que permite recibir, procesar y generar información. Por lo tanto y para simplificar, se pueden dividir los elementos de una computadora en tres partes empleando la arquitectura de computadoras de Von Neumann:

- Dispositivos de entrada.
- Dispositivos de salida.
- Unidad central de proceso.

Dispositivos de entrada. Son los dispositivos que permiten ingresar información a la computadora ya sea de forma manual o automática. Para ingresar información de forma manual se pueden utilizar dispositivos como el ratón, el teclado, unidades de disco, pantallas táctiles, etc. Para el ingreso automático de información se pueden utilizar sensores previamente configurados que leen constantemente información sin intervención del usuario.

Dispositivos de salida. Permiten obtener la información resultante tras el ingreso de algún tipo de información a la computadora. Algunos ejemplos son las pantallas, impresoras, unidades de disco, etc.

Unidad central de proceso. Es el hardware que ejecuta las instrucciones recibidas por la computadora a través de operaciones aritméticas, lógicas, de control y de entrada y salida.

También se pueden considerar dentro de dicha arquitectura los dispositivos de entrada y salida, es decir que pueden funcionar como ambos tipos, por ejemplo las pantallas táctiles.

En un teléfono inteligente y/o tableta electrónica es posible encontrar el siguiente hardware (Tabla 2-1):

Tabla 2-1 - Tipos de dispositivos según la arquitectura Von Neumann

Tipo de dispositivo	Ejemplos
Dispositivos de entrada	Antenas, pantalla táctil, micrófono, lectores de memorias, cámaras. Sensores de luminosidad, proximidad, barómetro, giroscopio, etc. Dispositivos externos conectados vía alambica o inalámbrica como teclados, mouse, controles de juegos, etc.
Dispositivos de salida	Pantalla, bocinas, motores, antenas, memorias, etc.
CPU	Procesadores de los fabricantes ARM e Intel con arquitecturas RISC, principalmente.

En lo que respecta al tipo de unidades de procesamiento, procesadores, para los dispositivos móviles existe una variación con respecto a los empleados en sistemas de escritorio y de mayor tamaño. Los procesadores empleados en los dispositivos móviles tienen un consumo mucho menor de energía y un conjunto de instrucciones reducido en comparación con los procesadores para computadoras de escritorio y laptops.

Un conjunto de instrucciones reducidas para computadora (RISC del inglés Reduced Instrucción Set Computer) es una estrategia de diseño de procesadores basada en la idea de que un conjunto reducido de instrucciones, al contrario de un conjunto complejo de instrucciones para computadora CISC, provee mayor rendimiento cuando se combina con una arquitectura de hardware capaz de ejecutar dichas instrucciones usando menos ciclos de procesador por cada instrucción.

2.3 Arquitectura de software de Android

Android es un sistema operativo móvil basado en el núcleo de Linux y actualmente desarrollado por Google. Con una interfaz de usuario basada en manipulación directa por parte del usuario, Android está diseñado principalmente para dispositivos móviles con pantallas táctiles como teléfonos inteligentes y tabletas electrónicas, además de contar con versiones especializadas en automóviles, televisiones y relojes de pulsera.

Android posee una arquitectura en forma de una pila o *stack* de software compuesta de aplicaciones, el sistema operativo, un entorno de ejecución, middleware, servicios y bibliotecas. En la ilustración 2-1 a continuación se presenta cada una de las capas de la pila y sus elementos correspondientes.

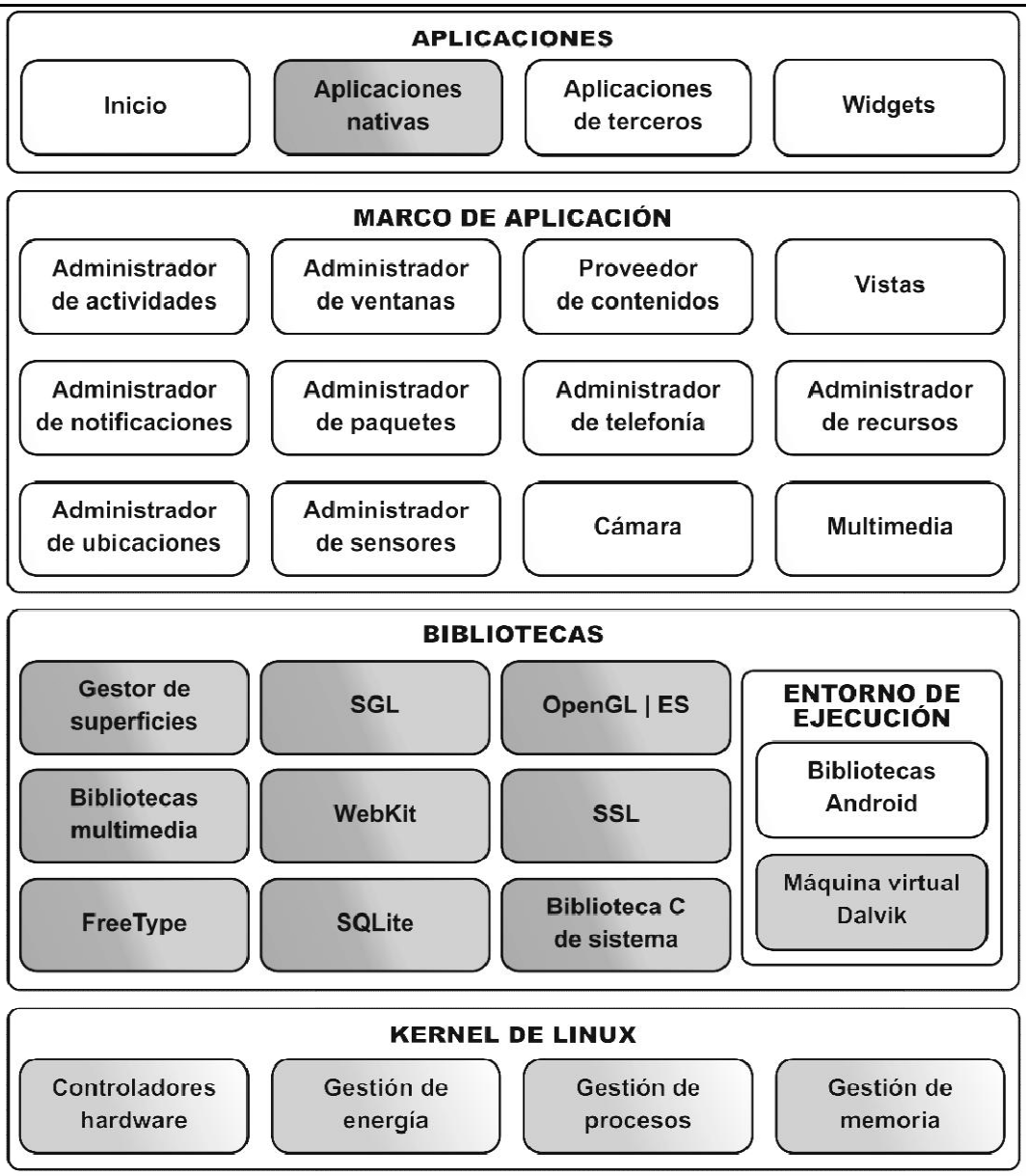


Ilustración 2-1 - Stack del sistema operativo Android

2.3.1 El núcleo o kernel de Linux

Posicionado en la base del stack de Android, el núcleo de Linux provee la abstracción entre los dispositivos de hardware y las capas superiores del stack de Android. Inicialmente, Android 1.0 se basó en la versión 2.6.23 del kernel de Linux y, actualmente, Android 5.0 se basa en la versión 3.14.

Sin embargo, la variante de Linux empleada en Android posee cambios adicionales en su arquitectura que han sido implementados por Google y fuera del ciclo típico de desarrollo del núcleo de Linux.

El núcleo original de Linux fue desarrollado en 1991 por Linus Torvalds y se combinó con un conjunto de herramientas, utilidades y compiladores desarrollados por Richard Stallman en la Free Software Foundation para crear un sistema operativo completo denominado GNU/Linux. Muchas son las distribuciones de Linux se han derivado de estos fundamentos.

2.3.2 Entorno de ejecución de Android

El núcleo de Linux proporciona un entorno de ejecución multitarea que permite que varios procesos se ejecuten al mismo tiempo. Sería fácil suponer, por tanto, que cada aplicación Android simplemente se ejecuta como un proceso directamente en el núcleo de Linux. Pero en realidad, cada aplicación que se ejecuta en un dispositivo Android lo hace dentro de su propia instancia de la máquina virtual Dalvik (VM).

La ejecución de aplicaciones en máquinas virtuales proporciona una serie de ventajas. En primer lugar, las aplicaciones no pueden interferir negativamente (intencionadamente o no) con el sistema operativo u otras aplicaciones, ni pueden acceder directamente al hardware del dispositivo. En segundo lugar, este nivel de abstracción forzada hace que la plataforma de aplicaciones sea neutral en cuanto a que las aplicaciones nunca están atadas a ningún hardware específico.

La máquina virtual Dalvik se basa en el kernel de Linux subyacente para las funcionalidades de bajo nivel. Es más eficiente que el estándar de la máquina virtual de Java, Java VM, en términos de uso de la memoria, y específicamente diseñado para permitir que varias instancias se ejecuten de manera eficiente dentro de los limitados recursos de un dispositivo móvil.

Para ejecutar una aplicación dentro de una máquina virtual Dalvik, el código de la aplicación debe ser transformada a partir de archivos estándar de clase Java a el formato ejecutable Dalvik (.dex), el cual tiene una huella de memoria 50% más pequeño que el bytecode de Java estándar.

Bibliotecas principales

Las bibliotecas del núcleo de Android, también conocidas como las bibliotecas de Dalvik (en referencia a la maquina virtual) se dividen en tres categorías.

- *Bibliotecas especificas de la máquina virtual Dalvik.* Es un conjunto de bibliotecas para interactuar directamente con una instancia de la máquina virtual y no es común que sea empleada por la mayoría de los desarrolladores de aplicaciones para Android.
- *Bibliotecas de interoperabilidad con Java.* Las aplicaciones Android se desarrollan predominantemente utilizando el lenguaje de programación Java. El entorno de desarrollo estándar de Java incluye una amplia gama de clases que están contenidas en las bibliotecas del entorno de ejecución básico de Java. Estas bibliotecas proporcionan apoyo para tareas tales como manejo de cadenas, redes y manipulación de archivos, por nombrar algunos, y son a la vez familiares y ampliamente utilizadas por los desarrolladores de Java, independientemente de la plataforma. Estas bibliotecas son una implementación de código abierto (basado en el proyecto Apache Harmony) de un subconjunto de las bibliotecas del núcleo estándar de Java que han sido adaptados y transformados para su uso por aplicaciones que se ejecutan dentro de una máquina virtual Dalvik.
- *Bibliotecas de Android.* Esta categoría abarca aquellas bibliotecas basadas en Java que son específicas para el desarrollo de Android. Los ejemplos de las bibliotecas en esta categoría incluyen las bibliotecas de marco de aplicación, además de las que facilitan la creación de interfaz de usuario, el dibujo de gráficos, el acceso de base de datos y el acceso al hardware del dispositivo móvil.

Las bibliotecas del núcleo de ejecución Android están basadas en Java y proporcionan la interfaz principal para los desarrolladores que escriben aplicaciones de Android. Es importante señalar, sin embargo, que las bibliotecas del núcleo en realidad no realizan gran parte del trabajo real y encapsulan esencialmente a un conjunto de bibliotecas basadas en los lenguajes C/C++. Al realizar llamadas, por ejemplo, a la biblioteca basada en Java *android.opengl* para dibujar gráficos en 3D en la pantalla del dispositivo, la biblioteca en realidad hace llamadas a la biblioteca de OpenGL que está escrita en el lenguaje C++, que, a su vez, funciona directamente con el núcleo Linux subyacente para realizar las tareas de dibujo.

En la práctica, el desarrollador de aplicaciones Android típicamente accederá a estas bibliotecas únicamente a través de las bibliotecas principales de Android basadas en Java. En el caso de que se necesite acceso directo a estas bibliotecas de bajo nivel, se puede lograr utilizando el Kit de Desarrollo Nativo de Android (*Native Development Kit, NDK*), cuyo propósito es llamar a los métodos nativos de lenguajes de programación no Java (tales como C y C++) en el código Java utilizando la interfaz nativa de Java (*Java Native Interface*, *JNI*).

Marco de trabajo de la aplicación

El marco de trabajo de la aplicación es un conjunto de servicios que forman colectivamente el entorno en el que las aplicaciones de Android se ejecutan y se gestionan. Este marco implementa el concepto de que las aplicaciones de Android se construyen a partir de componentes reutilizables, intercambiables y reemplazables. Este concepto se toma en las aplicaciones porque son capaces de publicar sus capacidades junto con los datos correspondientes de modo que se pueden encontrar y ser reutilizados por otras aplicaciones.

El entorno de trabajo de Android incluye principalmente los siguientes servicios:

- *Activity Manager.* Administrador de actividades, controla todos los aspectos del ciclo de vida de la aplicación y la pila de actividades.
- *Content Providers.* Proveedores de contenido, habilitan a las aplicaciones para publicar y compartir datos con otras aplicaciones.
- *Resource Manager.* Administrador de recursos, proporciona acceso a los recursos visuales y de contenido incrustado en la aplicación como cadenas, ajustes de color y diseños de interfaz de usuario.
- *Notifications Manager*. Administrador de notificaciones, permite a las aplicaciones mostrar alertas y notificaciones para el usuario.
- *Package Manager*. Gestor de paquetes, es el sistema por el cual las aplicaciones son capaces de buscar información acerca de otras aplicaciones instaladas actualmente en el dispositivo.
- *Telephony Manager*. Administrador de telefonía, proporciona información a la aplicación de los servicios de telefonía disponibles en el dispositivo.
- *Location Manager.* Administrador de localización, proporciona acceso a los servicios de localización que permiten a una aplicación recibir actualizaciones sobre los cambios de ubicación geográfica del dispositivo.

Aplicaciones

En la parte superior de la pila de software Android están las aplicaciones. Estas comprenden tanto las aplicaciones nativas proporcionadas con el sistema Android en particular (por ejemplo, las aplicaciones de navegador web y correo electrónico) y las aplicaciones de terceros instaladas por el usuario después de la compra del dispositivo.

CAPÍTULO 3 - INGENIERÍA DEL SOFTWARE DEL PROYECTO

A continuación, se presentan los procesos que se han seguido para la obtención de la aplicación de auto préstamo para la Biblioteca Central.

3.1 Requerimientos de la aplicación

Durante el planteamiento de la aplicación se obtienen de forma inmediata los requerimientos iníciales que, de forma general, se pueden clasificar en requerimientos de usuario y requerimientos de dominio.

A continuación se muestran los requerimientos de usuario de la aplicación:

Tabla 3-1 - Requerimientos de usuario

Clave	Enunciado
RU1	El usuario podrá autentificarse en línea.
RU2	El usuario tendrá acceso a su información personal e historial de actividades.
RU3	El usuario podrá llevar a cabo el auto préstamo y la renovación de dichos préstamos de material bibliográfico.
RU4	Debe existir una sección con información sobre la Biblioteca Central, sus servicios y su reglamento.

Los requerimientos de usuario describen los requerimientos para los usuarios finales de la aplicación, en este caso los usuarios de la Biblioteca Central. Por otro lado, los requerimientos de dominio se basan en la legislación e infraestructura disponibles de la institución. A continuación de enlistan los requerimientos de dominio detectados, los cuales en su mayoría se derivan de las reglas de negocio o reglas de funcionamiento de la Biblioteca Central:

Tabla 3-2 - Requerimientos de dominio

Clave	Titulo	Enunciado
RD1	Tipo de usuario	Se deben considerar diferentes tipos de usuario a partir de un identificador llamado *Estatus*.
RD2	Máximo de libros	Todos los tipos de usuario tienen derecho al préstamo simultáneo de hasta tres libros.
RD3	Duración del préstamo	Todos los usuarios tienen derecho a 7 días naturales de préstamo por cada libro.
RD4	Máximo de renovaciones	El usuario tiene derecho a dos renovaciones del mismo libro de forma consecutiva.
RD5	Permisos del usuario	El usuario debe estar vigente en el sistema para que pueda realizar operaciones de consulta de libros, renovación y auto préstamos.
RD6	Validez del usuario	El usuario debe estar correctamente autenticado en la aplicación, y tener permisos para llevar a cabo operaciones de consulta de préstamos, renovaciones y auto préstamos.

RD7	Préstamo vencido	El usuario no puede tener un préstamo de libro vencido para poder efectuar un nuevo préstamo/renovación.
RD8	Validez del material para renovación	La renovación de un libro puede efectuarse siempre y cuando el material no se encuentre en la lista de espera por otro usuario.
RD9	Validez del material para préstamo	No serán objeto de préstamo a domicilio, los materiales deteriorados, únicos, obras de consulta, publicaciones periódicas, tesis, documentos del Fondo Antiguo y colecciones especiales, materiales audiovisuales y todos aquellos que la Biblioteca Central determine.
RD10	Formato de la clave del usuario	La clave del usuario está compuesta por dígitos.
RD11	Usuario registrado en el sistema	El usuario debe tener una cuenta vigente en la Biblioteca Central.
RD12	Formato del número de adquisición	El número de adquisición del acervo de la Biblioteca Central está compuesto por dígitos.

Ahora, se presentan los requerimientos no funcionales, es decir que están relacionados directamente con la infraestructura y los servicios disponibles, además de la calidad esperada al utilizar la aplicación.

Tabla 3-3 - Requerimientos no funcionales

Clave	Titulo	Enunciado
RNF1	Disponibilidad	El sistema debe estar disponible “7 por 24 por 365”.
RNF2	Despliegue	Debe desplegarse en tamaños de pantalla chico, mediano y grande de dispositivos Android.
RNF3	Versión API	La aplicación debe visualizarse en la mayor cantidad de dispositivos móviles Android, con API superior a la 4.0.
RNF4	Comunicación	La comunicación entre la aplicación y los servicios de la biblioteca son a través de Web Services.
RNF5	Formato de la información	El formato de la información intercambiada es usando XML.
RNF6	Encriptación	Se debe encriptar la información enviada a los servicios web.
RNF7	Ofuscación	El código fuente de la aplicación debe estar ofuscado.
RNF8	Autenticación	No existen intentos máximos de autenticación en la aplicación.
RNF9	Validación usuario	En comunicación sensible con el servidor se enviarán las credenciales del usuario en cada mensaje.
RNF10	Alcance	La aplicación solo funcionará para el acervo de la Biblioteca Central.
RNF11	Cuenta	Sólo una cuenta puede estar asignada al dispositivo.
RNF12	Sesión	La sesión del usuario permanecerá activa de forma indefinida. Sólo terminará si el usuario borra sus datos en la aplicación.

Finalmente, se presentan los requerimientos funcionales donde se detallan los servicios que la aplicación debe proveer al usuario. Posteriormente los requerimientos funcionales se analizarán a mayor detalle con sus entradas y sus salidas en los diagramas de casos de uso.

Tabla 3-4 - Requerimientos funcionales

Clave	Titulo	Enunciado
RF1	Autenticación	En la primera pantalla se solicitará al usuario que proporcione su clave y contraseña. Si la información es correcta, podrá acceder a las secciones de: Mis Datos y Auto préstamo.
RF2	Registro	Si el usuario no se encuentra registrado, deberá proporcionar la información necesaria para completar el registro a través de la aplicación.

RF3	Datos de la cuenta	Si el usuario ya se encuentra registrado, podrá ver sus datos generales.
		• Nombre
		• Clave
		• Listado con 'Mis préstamos'
		  ○ Al seleccionar un ítem del listado podrá renovar el préstamo, si el sistema lo permite; también podrá compartir ese libro con otras aplicaciones.
		• Eliminar cuenta
		  ○ Borrar datos de la cuenta dentro del dispositivo.

RF4	Auto préstamo	Al escanear el código de barras identificador de un libro, el usuario podrá auto-prestarse el libro si el sistema lo permite.

RF5	Acerca de la biblioteca	En esta sección de la aplicación se podrán obtener información útil y cultural de la Biblioteca Central:
		• Ubicación, horarios de servicio
		La ubicación se detalla a través de un mapa nativo del sistema.
		• Directorio
		• Información
		  ○ La biblioteca
		  ○ Historia
		• Murales
		Muestra una galería que, dependiendo del elemento seleccionado, muestra información detallada del mismo.
		• Distribución del acervo
		Muestra la distribución del acervo a lo largo del edificio de la Biblioteca Central.

RF6	Buscador	Sección donde el usuario podrá buscar libros del acervo de la Biblioteca Central. Esta sección constará de dos pantallas. En la primera pantalla, el usuario tendrá la posibilidad de buscar un libro a partir de:

		• Título • Autor • Tema • Todos los campos Al dar clic en el botón de buscar, saltará a la siguiente pantalla donde aparecerá un listado con los libros que cumplen los criterios de la búsqueda.
RF7	Reglamento	Se mostrará un archivo PDF con el reglamento de la Biblioteca Central.

RF8	Configuración	En esta pantalla, el usuario podrá Activar/Desactivar las notificaciones de los préstamos de sus libros.
		• Número de días previos que lanzarán la alarma.
		• Hora para lanzar la notificación.
		• Volumen.
		• Aplazar
		  ○ Duración
		  ○ Repetir

3.2 Diseño

El diseño de la aplicación parte de una arquitectura cliente servidor, en la cual la aplicación utilizará los servicios disponibles de la Biblioteca Central. Ilustración 3-1.

Para la interacción entre la aplicación y los servidores de la biblioteca solo es necesario conocer la interfaz o el contrato con los servicios proporcionados por la biblioteca, los detalles de la implementación de dichos servicios quedan fuera del alcance de la aplicación. La comunicación se llevará a cabo a través de la tecnología de servicios web (Web Services) que utiliza un conjunto de protocolos y estándares para intercambiar datos entre aplicaciones, ello independientemente de la tecnología con que se haya creado cada aplicación.

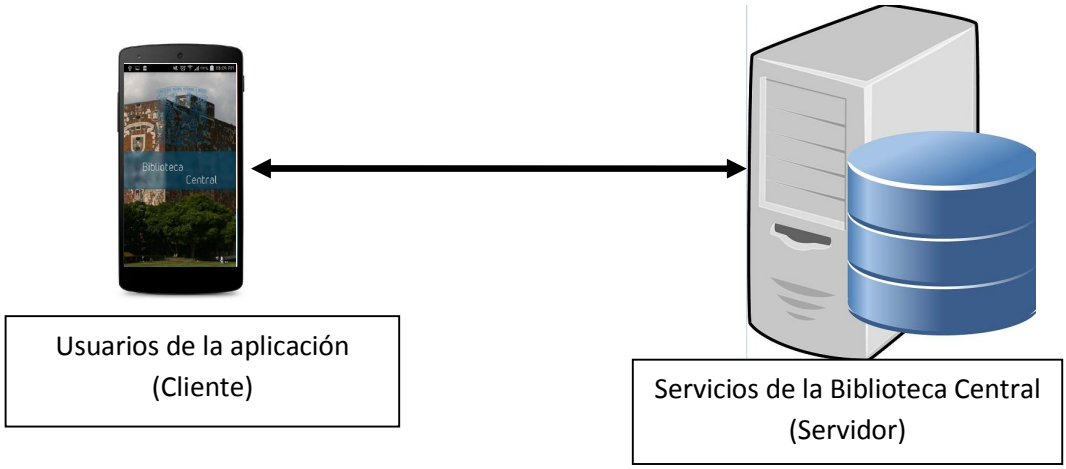


Ilustración 3-1 - Arquitectura general de la aplicación

El diagrama de navegación es un bosquejo del funcionamiento general de la aplicación, es decir los pasos o la ruta que debe seguir el usuario para realizar una determinada acción dentro de la

aplicación, partiendo de los requerimientos funcionales y no funcionales previamente definidos. Ilustración 3-2 a continuación.

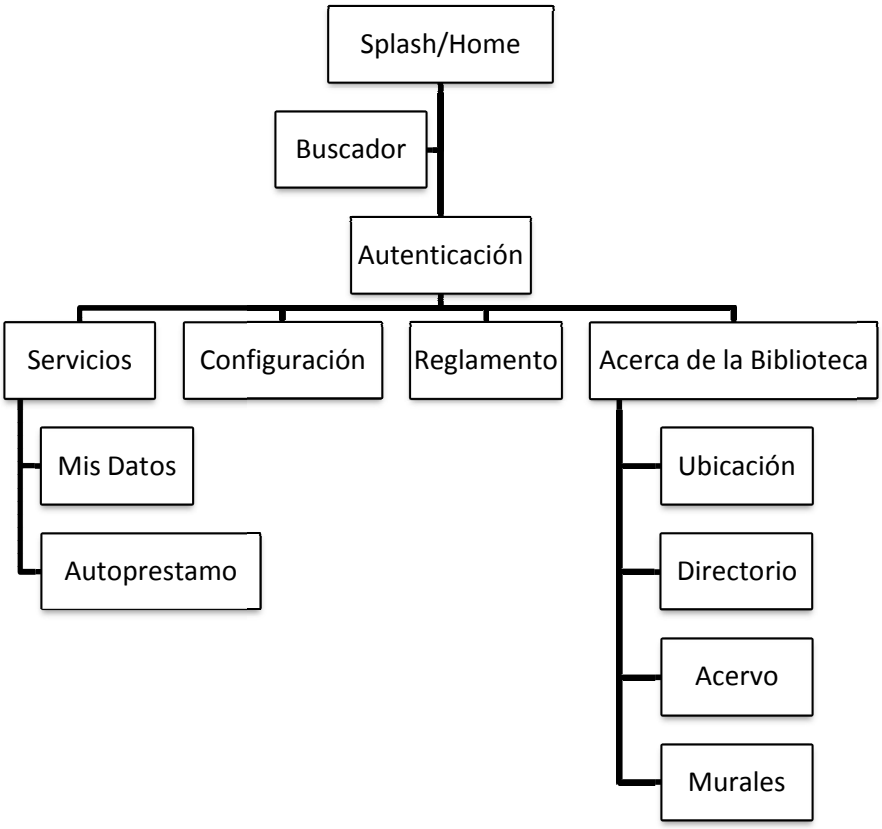


Ilustración 3-2 - Diagrama de navegación

A partir del diagrama de navegación y de la funcionalidad definida para cada pantalla se puede diseñar la interfaz gráfica, que a su vez será evaluada por el cliente y potenciales usuarios para garantizar que cumple con requisitos como usabilidad, sencilla navegación, que sea entendible y que sea congruente en cómo se invocan las tareas a través de las distintas pantallas de la aplicación. En el caso de esta aplicación se contó con el apoyo de un diseñador gráfico especializado en aplicaciones móviles para generar un tema visual y los componentes, imágenes, para integrarse en la aplicación.

Los elementos de la aplicación que aportan la funcionalidad pueden ser definidos a través de casos de uso, los cuales se generan a partir de las listas de requerimientos. Los casos de uso definen de forma concreta a los actores o usuarios de la aplicación, las interacciones disponibles para cada uno de ellos e inclusive algún detalle sobre que módulo o parte de la aplicación será la encargada de atender las acciones. A continuación se presentan las ilustraciones 1-3 a 1-7 con los diagramas de casos de uso para la aplicación de la Biblioteca Central.

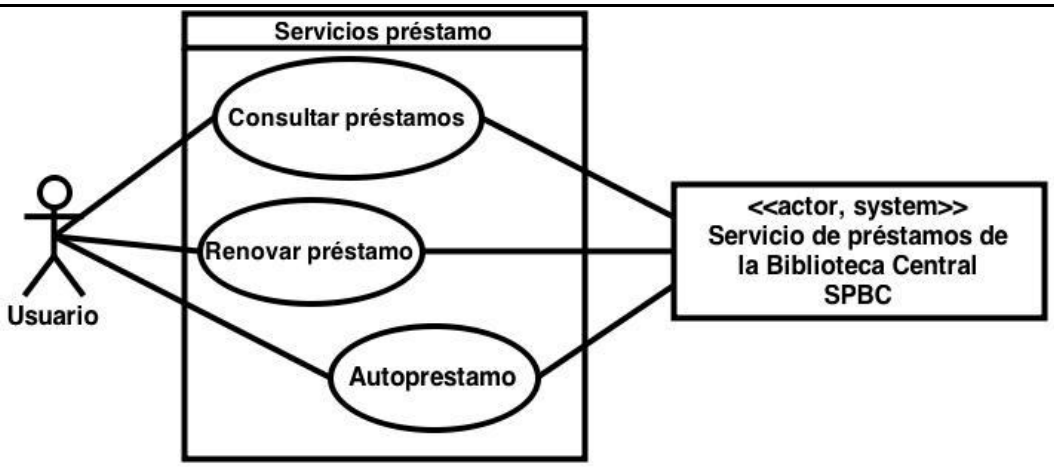


Ilustración 3-3 - Caso de uso para el módulo de préstamos

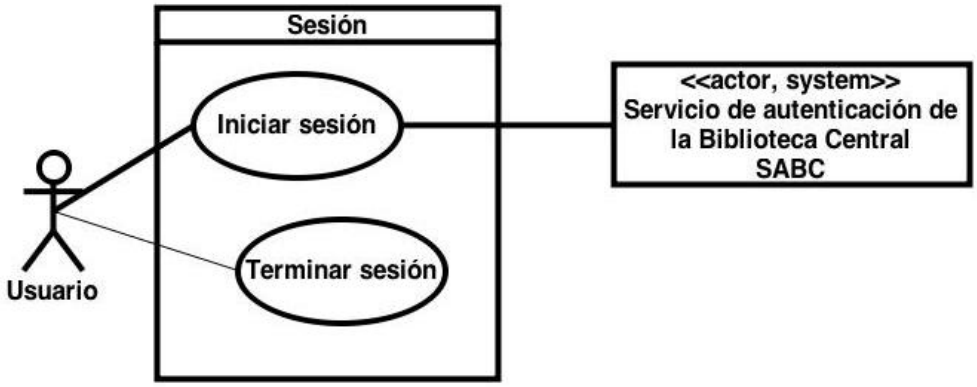


Ilustración 3-4 - Caso de uso para el módulo de sesión

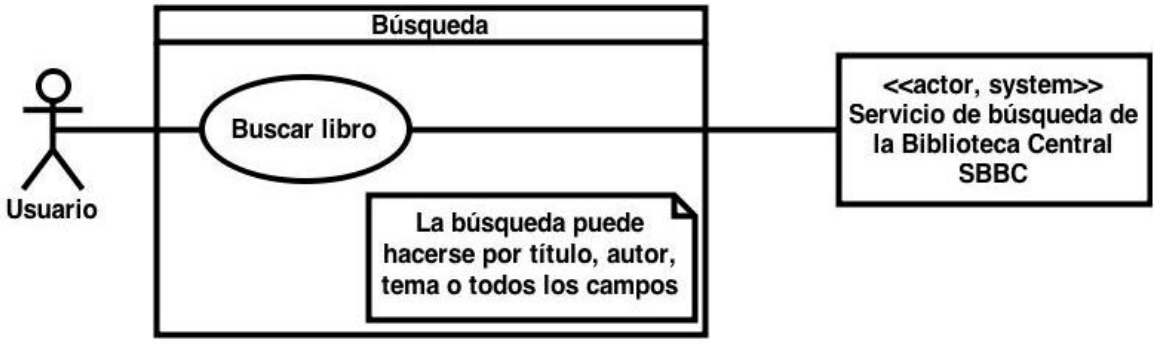


Ilustración 3-5 - Caso de uso para el módulo de búsqueda

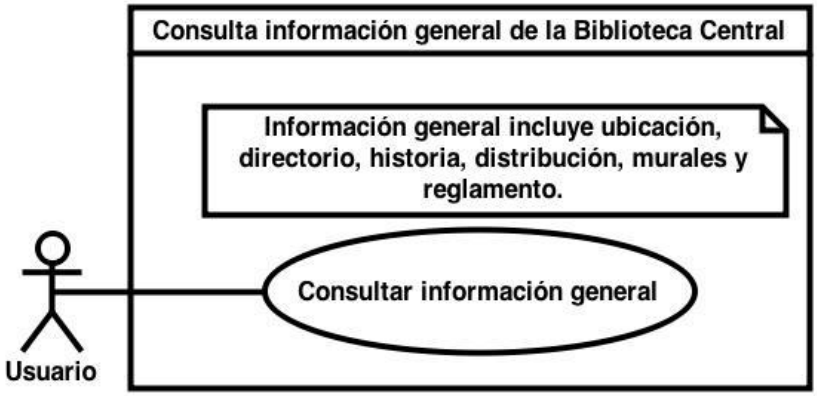


Ilustración 3-6 - Caso de uso para el módulo de información de la Biblioteca

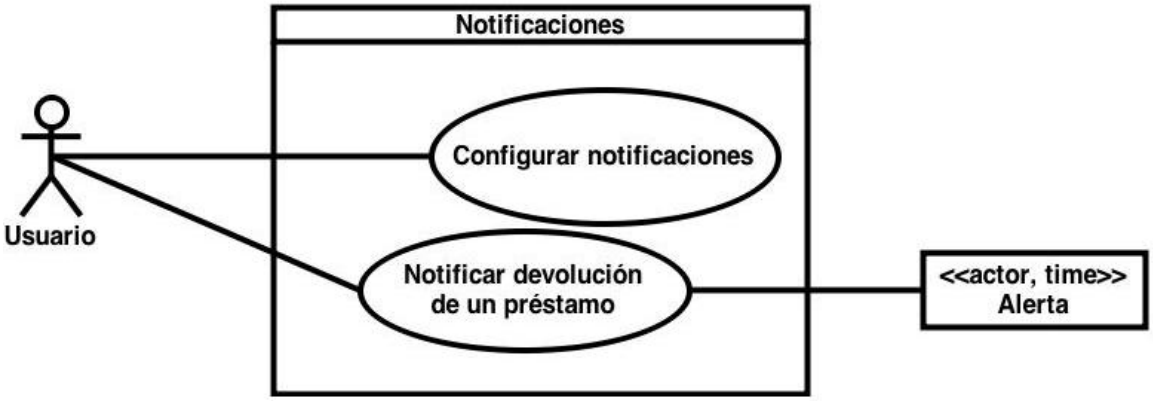


Ilustración 3-7 - Caso de uso para el módulo de notificaciones

También es útil contar con los casos de uso en una forma más detallada, en las siguientes tablas (tablas 3-5 a 3-10) para cada caso de uso se incluyen enunciados con más información sobre cada caso, prioridades, actores, riesgos, dependencias, que requerimientos y que reglas de negocio están cubriendo.

Todos los casos de uso deben cumplir los siguientes requerimientos no funcionales (*RNF*):

- *RNF1*. Disponibilidad
- *RNF2*. Despliegue
- *RNF3*. Versión API

Tabla 3-5 - Detalle de caso de uso de consulta de prestamos

Nombre del caso de uso	**Consultar préstamos.**
Descripción	El usuario visualiza los libros que tiene en préstamo, junto con información detallada de cada préstamo.
Actores	Primario: Usuario Secundario: SPBC (Sistema de préstamos de la Biblioteca Central).
Prioridad	Must have, este caso de uso es esencial para el sistema.

Riesgo	Alto, el proceso de comunicación con el servicio puede fallar por varias razones.
Precondiciones y supuestos	El usuario cuenta con un registro válido en la Biblioteca Central, se encuentra autenticado en el sistema, y cuenta con conectividad a Internet.
Puntos de extensión	Compartir información del libro.
Extiende	Ninguno.
Trigger	El usuario desea consultar el listado con los libros que tiene en préstamo, junto con la información de cada libro.
Flujo principal de eventos	1. El caso de uso inicia cuando el usuario solicita visualizar sus préstamos vigentes. 2. El sistema envía a SPBC los datos de sesión del usuario, y la clave de la Biblioteca. [A1] [A3] [RNF4] [RNF5] [RNF6][RNF9] 3. SPBC recibe los datos de sesión del usuario, y la clave de la Biblioteca. 4. SPBC valida los datos de sesión del usuario. [A2] 5. SPBC consulta la información de los préstamos del usuario. [A4] [RD1] [RD5] [RD6] 6. SPBC regresa al sistema un listado con los préstamos vigentes del usuario. [RNF10] 7. El sistema recibe el listado con los préstamos vigentes del usuario. [A1] [A3] 8. El sistema despliega la información de cada libro en préstamo al usuario: Título, autor, número de adquisición, fecha de préstamo, fecha de renovación, fecha a devolver. [A5] [A6] 9. El caso de uso termina.
Flujo alternativo de eventos (A)	A1. El sistema no tiene conectividad a Internet. Ir a [AE] A2. Los datos de sesión del usuario no son válidos. Ir a [AE] A3. Ocurrió un error en el proceso de comunicación con el servicio. Ir a [AE] A4. El usuario no cuenta con los permisos ni con una cuenta válida para consultar los préstamos. Ir a [AE] A5. La información no tiene un formato correcto y no es posible desplegarla. Ir a [AE] A6. El usuario desea compartir la información del libro. AE. Se notifica al usuario que ocurrió un error. El caso de uso termina. En cualquier momento el usuario puede cancelar el caso de uso. El caso de uso termina.
Postcondiciones	Ninguno.
Requerimientos de Dominio (RD)	RD1. Tipo de usuario RD5. Permisos del usuario RD6. Validez del usuario
Requerimientos no funcionales (RNF)	RNF4. Comunicación RNF5. Formato de la información RNF6. Encriptación RNF9. Validación usuario RNF10. Alcance
Notas	Ninguna.
Nombre del caso de uso	Auto préstamo.
Descripción	El usuario efectúa un préstamo directamente desde el sistema sin necesidad de efectuar la operación en estantería abierta.
Actores	Primario: Usuario Secundario: SPBC (Sistema de préstamos de la Biblioteca Central)
Prioridad	Must have, este caso de uso es esencial para el sistema.
Riesgo	Alto, el proceso de comunicación con el servicio puede fallar por varias razones.

Precondiciones y supuestos	El usuario cuenta con un registro válido en la Biblioteca Central , se encuentra autenticado en el sistema, se encuentra en las instalaciones de la Biblioteca Central, tiene conectividad a Internet, y el libro que desea adquirir en préstamo está disponible.
Puntos de extensión	Ninguno.
Extiende	Ninguno.
Trigger	El usuario se encuentra en las instalaciones de la Biblioteca Central y tiene un libro que desea auto prestarse.
Flujo principal de eventos	1. El caso de uso inicia cuando el usuario ingresa el número de adquisición del libro. [A12] 2. El sistema envía los datos de sesión del usuario, la clave de la Biblioteca, y el número de adquisición del libro a SPBC. [A1] [A3] [RD1] [RD6] [RNF4] [RNF5] [RNF6] [RNF10] 3. SPBC recibe los datos de sesión del usuario, la clave de la Biblioteca, y el número de adquisición del libro. 4. SPBC valida los datos de sesión del usuario. [A2] 5. SPBC consulta la información del libro. [A4] [A7] 6. SPBC regresa la información del libro al sistema. 7. El sistema recibe la información del libro. [A1] [A3] 8. El sistema despliega la información del libro al usuario y le pregunta si es el libro que desea auto prestarse. [A11] 9. El usuario confirma que es el libro que desea auto prestarse. [AE] 10. El Sistema envía los datos de sesión del usuario, la clave de la Biblioteca, y la información del libro a SPBC. [A1] [A3] [RNF4] [RNF5] [RNF6] [RNF10] 11. SPBC recibe los datos de sesión del usuario, la clave de la Biblioteca y la información del libro. 12. SPBC valida los datos de sesión del usuario, y el estatus del libro. [A2] [A5] [A6] [A7] [A8] [A9] [RD2] [RD5] [RD6] [RD7] [RD9] 13. SPBC actualiza el estatus de préstamo del libro. [A10] 14. SPBC envía el resultado de la actualización del estatus del préstamo al sistema. 15. El sistema recibe el resultado de la actualización del estatus del préstamo. [A1] [A3] 16. El sistema notifica al usuario el resultado de la operación de auto préstamo. [A11] 17. El caso de uso termina.
Flujo alternativo de eventos (A)	A1. El dispositivo no tiene conectividad a Internet. Ir a [AE] A2. Los datos de sesión del usuario no son válidos. Ir a [AE] A3. Ocurrió un error en el proceso de comunicación con el servicio. Ir a [AE] A4. El libro no está dado de alta en el acervo de la Biblioteca Central. Ir a [AE] A5. La vigencia del usuario es extemporánea. Ir a [AE] A6. El usuario no tiene permisos para efectuar el préstamo. Ir a [AE] A7. Existen inconsistencias en el acervo. Ir a [AE] A8. El usuario ya tiene tres préstamos vigentes y no puede solicitar uno adicionar. Ir a [AE] A9. El usuario tiene un préstamo extemporáneo. Ir a [AE] A10. Ocurrió un error al actualizar la información. Ir a [AE] A11. La información no tiene un formato correcto y no es posible desplegarla. Ir a [AE] A12. El usuario escanea el número de adquisición del libro. AE. Se notifica al usuario que ocurrió un error. El caso de uso termina. En cualquier momento el usuario puede cancelar el caso de uso. El caso de uso

	termina.
Postcondiciones	El estatus del libro es prestado al usuario.
Requerimiento de Dominio (RD)	RD1. Tipo de usuario RD2. Máximo de libros RD5. Permisos del usuario RD6. Validez del usuario RD7. Préstamo vencido RD9. Validez material para el préstamo
Requerimientos no funcionales (RNF)	RNF4. Comunicación RNF5. Formato de la información RNF6. Encriptación RNF9. Validación usuario RNF10. Alcance
Notas	Ninguna.

Tabla 3-6 - Detalle de caso de renovar préstamo

Nombre del caso de uso	**Renovar préstamo.**
Descripción	El usuario renueva un préstamo vigente.
Actores	Primario: Usuario Secundario: SPBC (Sistema de préstamos de la Biblioteca Central).
Prioridad	Must have, este caso de uso es esencial para el sistema.
Riesgo	Alto, el proceso de comunicación con el servicio puede fallar por varias razones.
Precondiciones y supuestos	El usuario cuenta con un registro válido en la Biblioteca Central, se encuentra autenticado en el sistema, tiene conectividad a Internet, y tiene por lo menos un libro en préstamo.
Puntos de extensión	Ninguno.
Extiende	Ninguno.
Trigger	El usuario desea renovar un préstamo vigente.
Flujo principal de eventos	1. El caso de uso inicia cuando el usuario solicita renovar el préstamo vigente de un libro. 2. El sistema envía a SPBC los datos de sesión del usuario, la clave de la Biblioteca, y el número de adquisición del libro. [A1] [A3] [RD1] [RNF4] [RNF5] [RNF6] [RNF9] [RNF10] 3. SPBC recibe los datos de sesión del usuario, la clave de la Biblioteca y el número de adquisición del libro. 4. SPBC valida los datos de sesión del usuario, y el estatus del libro. [A2] [A4] [A5] [A6] [A7] [A8] [A9] [RD3] [RD4] [RD5] [RD6] [RD7] [RD8] 5. SPBC actualiza el estatus del préstamo del libro. [A10] 6. SPBC envía el resultado de la renovación al sistema. 7. El sistema recibe de SPBC el resultado de la renovación. [A1] [A3] 8. El sistema notifica al usuario el resultado de la renovación. [A11] 9. El sistema actualiza la información de las fechas de renovación y de devolución del libro. 10. El caso de uso termina.
Flujo alternativo de eventos (A)	A1. El dispositivo no tiene conectividad a Internet. Ir a [AE] A2. Los datos de sesión del usuario no son válidos. Ir a [AE] A3. Ocurrió un error en el proceso de comunicación con el servicio. Ir a [AE] A4. La vigencia del libro es extemporánea. Ir a [AE] A5. El usuario no tiene permisos para efectuar la renovación. Ir a [AE]

	A6. Existen inconsistencias en el acervo de la Biblioteca Central. Ir a [AE] A7. No existe registro del préstamo en el acervo de la Biblioteca Central. Ir a [AE] A8. El usuario ya tiene dos renovaciones del libro. Ir a [AE] A9. El usuario tiene algún libro con préstamo extemporáneo. Ir a [AE] A10. Existe un error al actualizar el estatus del préstamo. Ir a [AE] A11. La información no tiene un formato correcto y no es posible desplegarla. Ir a [AE] AE. Se notifica al usuario que ocurrió un error. El caso de uso termina. En cualquier momento el usuario puede cancelar el caso de uso. El caso de uso termina
Postcondiciones	El estatus del préstamo del libro es renovado.
Requerimiento de Dominio (RD)	RD1. Tipo de usuario RD3. Duración del préstamo RD4. Máximo de renovaciones RD5. Permisos del usuario RD6. Validez del usuario RD7. Préstamo vencido RD8. Validez del material para renovación
Requerimientos no funcionales (RNF)	RNF4. Comunicación RNF5. Formato de la información RNF6. Encriptación RNF9. Validación usuario RNF10. Alcance
Notas	Ninguna.

Tabla 3-7 - Detalle de caso de uso de inicio de sesión

Nombre del caso de uso	**Iniciar sesión**
Descripción	Permite al usuario autenticarse en el sistema.
Actores	Primario: Usuario Secundario: SABC (Sistema de autenticación de la Biblioteca Central)
Prioridad	Must have, este caso de uso es esencial para el sistema.
Riesgo	Alto, el proceso de comunicación con el servicio puede fallar por varias razones.
Precondiciones y supuestos	El usuario tiene una cuenta válida en la Biblioteca Central, y tiene conectividad a Internet.
Puntos de extensión	Ninguno.
Extiende	Ninguno.
Trigger	El usuario desea autenticarse en el sistema.
Flujo principal de eventos	1. El usuario introduce sus datos de sesión en el sistema. [RNF8] 2. El sistema valida que los datos de sesión del usuario tengan un formato válido. [A5] [RD10] 3. El sistema envía los datos de sesión del usuario, y la clave de la Biblioteca a SABC. [A1] [A3] [RNF4] [RNF5] [RNF6] 4. SABC recibe los datos de sesión del usuario y la clave de la Biblioteca. 5. SABC valida los datos de sesión del usuario. [A2] [RD1] [RD11] 6. SABC envía al sistema el resultado de la validación del usuario. 7. El sistema recibe el resultado de la validación de SABC. [A1] [A3] 8. El sistema notifica al usuario el resultado de la validación. [A4] 9. El sistema almacena los datos de sesión del usuario. [RNF11] [RNF12]

	10. El caso de uso termina.
Flujo alternativo de eventos (A)	A1. El dispositivo no tiene conectividad a Internet. Ir a [AE] A2. Los datos de sesión del usuario no son válidos. Ir a [AE] A3. Ocurrió un error en el proceso de comunicación con el servicio. Ir a [AE] A4. La información no tiene un formato correcto y no es posible desplegarla. Ir a [AE] A5. La información introducida por el usuario no contiene un formato correcto. Notificar al usuario e ir a 1. A6. Ocurre un error al guardar la sesión del usuario. Ir a [AE]. AE. Se notifica al usuario que ocurrió un error. El caso de uso termina. En cualquier momento el usuario puede cancelar el caso de uso. El caso de uso termina.
Postcondiciones	El usuario se autentifica en el sistema.
Requerimiento de Dominio (RD)	RD1. Tipo de usuario RD10. Formato de la clave del usuario RD11. Usuario registrado en el sistema
Requerimientos no funcionales (RNF)	RNF4. Comunicación RNF5. Formato de la información RNF6. Encriptación RNF8. Autenticación RNF11. Cuenta RNF12. Sesión
Notas	Ninguna.

Tabla 3-8 - Detalle de caso de uso de terminar sesión

Nombre del caso de uso	**Terminar sesión**
Descripción	Permite al usuario eliminar su clave y contraseña del sistema.
Actores	Principal: Usuario Secundario: Ninguno
Prioridad	Must have, este caso de uso es esencial para el sistema.
Riesgo	Bajo, el sistema no afecta la información de la cuenta del usuario.
Precondiciones y supuestos	El usuario cuenta con un registro válido en la Biblioteca Central y se encuentra autenticado en el sistema.
Puntos de extensión	Ninguno.
Extiende	Ninguno.
Trigger	El usuario desea eliminar la información de su cuenta en el sistema.
Flujo principal de eventos	1. El caso de uso inicia cuando el usuario solicita eliminar su cuenta del sistema. [RD11] [RNF11] 2. El sistema le pregunta al usuario si está seguro de eliminar su cuenta del sistema. 3. El usuario confirma que desea eliminar su cuenta del sistema. [A2] 4. El sistema elimina la cuenta del usuario. [A1] [RNF12] 5. El caso de uso finaliza.
Flujo alternativo de eventos (A)	A1. Ocurrió un error en el proceso de eliminación de la cuenta del usuario. Ir a [AE] A2. El usuario cancela proceso de eliminar cuenta. El caso de uso finaliza. AE. Se notifica al usuario que ocurrió un error. El caso de uso termina.

	En cualquier momento el usuario puede cancelar el caso de uso. El caso de uso termina.
Postcondiciones	El sistema elimina la clave y contraseña del usuario en el sistema.
Requerimiento de Dominio (RD)	RD11. Usuario registrado en el sistema
Requerimientos no funcionales (RNF)	RNF11. Cuenta RNF12. Sesión
Notas	Ninguna.

Tabla 3-9 - Detalle de caso de uso de buscar libro

Nombre del caso de uso	**Buscar libro.**
Descripción	El sistema permite buscar un libro en el acervo de la Biblioteca Central, ya sea por título y/o autor y/o tema/todos.
Actores	Principal: Usuario Secundario: SBBC (Sistema de Búsqueda de la Biblioteca Central)
Prioridad	Must have, este caso de uso es esencial para el sistema.
Riesgo	Alto, el proceso de comunicación con el servicio puede fallar por varias razones.
Precondiciones y supuestos	El usuario tiene conectividad a Internet.
Puntos de extensión	Ninguno.
Extiende	Ninguno.
Trigger	El usuario desea obtener información de un libro que pertenezca al acervo de la Biblioteca Central.
Flujo principal de eventos	1. El caso de uso inicia cuando el usuario selecciona buscar por título. [A4] [A5] [A6] 2. El usuario introduce un texto. 3. El sistema valida que el usuario introduce un texto válido. [A7] 4. El sistema envía a SBBC la información introducida por el usuario. [A1] [A2] [RNF4] [RNF5] 5. SBBC recibe los parámetros de búsqueda. 6. SBBC consulta el acervo para obtener los libros que coinciden con los parámetros de búsqueda. [A8] [RNF10] 7. SBBC envía al sistema el resultado de la búsqueda. [A1] [A2] 8. El sistema recibe de SBBC el resultado de la búsqueda. 9. El sistema despliega al usuario el resultado de la búsqueda. [A3] 10. El caso de uso termina.
Flujo alternativo de eventos (A)	A1. El dispositivo no tiene conectividad a Internet. Ir a [AE] A2. Ocurrió un error en el proceso de comunicación con el servicio. Ir a [AE] A3. La información no tiene un formato correcto y no es posible desplegarla. Ir a [AE] A4. El usuario selecciona buscar por autor. A5. El usuario selecciona buscar por tema. A6. El usuario selecciona buscar por todos los campos. A7. El usuario no introdujo un texto de búsqueda válido. Notificar al usuario. Ir a 2 A8. Ocurre un error al efectuar la consulta en SBBC. Ir a [AE] AE. Se notifica al usuario que ocurrió un error. El caso de uso termina. En cualquier momento el usuario puede cancelar el caso de uso. El caso de uso termina.

Postcondiciones	Ninguna.
Requerimiento de Dominio (RD)	Ninguna(s).
Requerimientos no funcionales (RNF)	RNF4. Comunicación RNF5. Formato de la información RNF10. Alcance
Notas	Ninguna.

Tabla 3-10 - Detalle de caso de uso de consulta de información general

Nombre del caso de uso	**Consultar información general**
Descripción	El sistema despliega información de la ubicación/directorio/historia/distribución del acervo/murales/reglamento de la Biblioteca Central.
Actores	Principal: Usuario Secundario: Ninguno
Prioridad	Should have, este caso de uso es deseable en el sistema.
Riesgo	Bajo, la información de la cuenta del usuario no se ve comprometida.
Precondiciones y supuestos	Ninguna.
Puntos de extensión	Ninguno.
Extiende	Ninguno.
Trigger	El usuario desea consultar información general de la Biblioteca Central.
Flujo principal de eventos	1. El caso de uso inicia cuando el usuario selecciona desplegar el directorio de la Biblioteca Central. [A3] [A4] [A5] [A6] [A7] [RNF10] 2. El sistema consulta la información del directorio de la Biblioteca Central. [A1] [A8] [A9] [A10] [A11] [A12] 3. El sistema despliega el directorio de la Biblioteca Central. [A2] [A13] [A14] [A15] [A16] [A17] 4. El caso de uso termina.
Flujo alternativo de eventos (A)	A1. Ocurrió un error en el proceso de consulta de la información. Ir a [AE] A2. La información no tiene un formato correcto y no es posible desplegarla. Ir a [AE] A3. El usuario selecciona consultar la historia. A4. El usuario selecciona consultar la distribución del acervo. A5. El usuario selecciona consultar los murales. A6. El usuario selecciona consultar ubicación. A7. El usuario selecciona consultar el reglamento. A8. El sistema consulta la historia. A9. El sistema consulta la distribución del acervo. A10. El sistema consulta los murales. A11. El sistema consulta la ubicación. A12. El sistema consulta el reglamento A13. El sistema despliega la historia. A14. El sistema despliega la distribución del acervo. A15. El sistema despliega los murales. A16. El sistema despliega la ubicación. A17. El sistema despliega el reglamento. AE. Se notifica al usuario que ocurrió un error. El caso de uso termina. En cualquier momento el usuario puede cancelar el caso de uso. El caso de uso termina.

Postcondiciones	Ninguna.
Requerimiento de Dominio (RD)	Ninguna(s).
Requerimientos no funcionales (RNF)	RNF10. Alcance
Notas	Ninguna.

Para completar cada caso de uso se requiere seguir un flujo de trabajo, un proceso de negocio o un proceso de software a través de una serie de acciones y condiciones. Estas acciones pueden ser llevadas a cabo por personas o componentes de software. A continuación, en las ilustraciones 3-8 a 3-14, se muestran los diagramas de actividades que describen los flujos de trabajo requeridos entre los usuarios, la aplicación móvil y el servidor de la Biblioteca Central para ejecutar satisfactoriamente cada caso de uso.

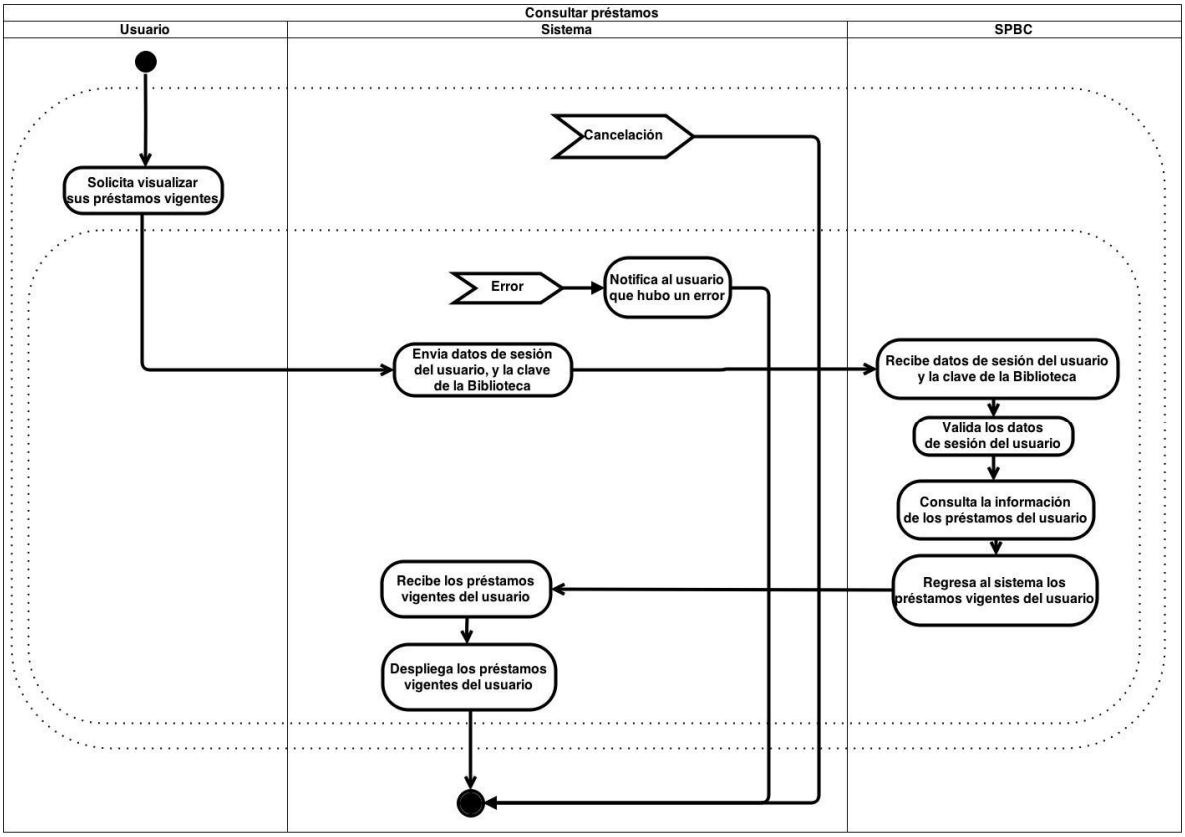


Ilustración 3-8 - Diagrama de actividades: Consultar préstamos

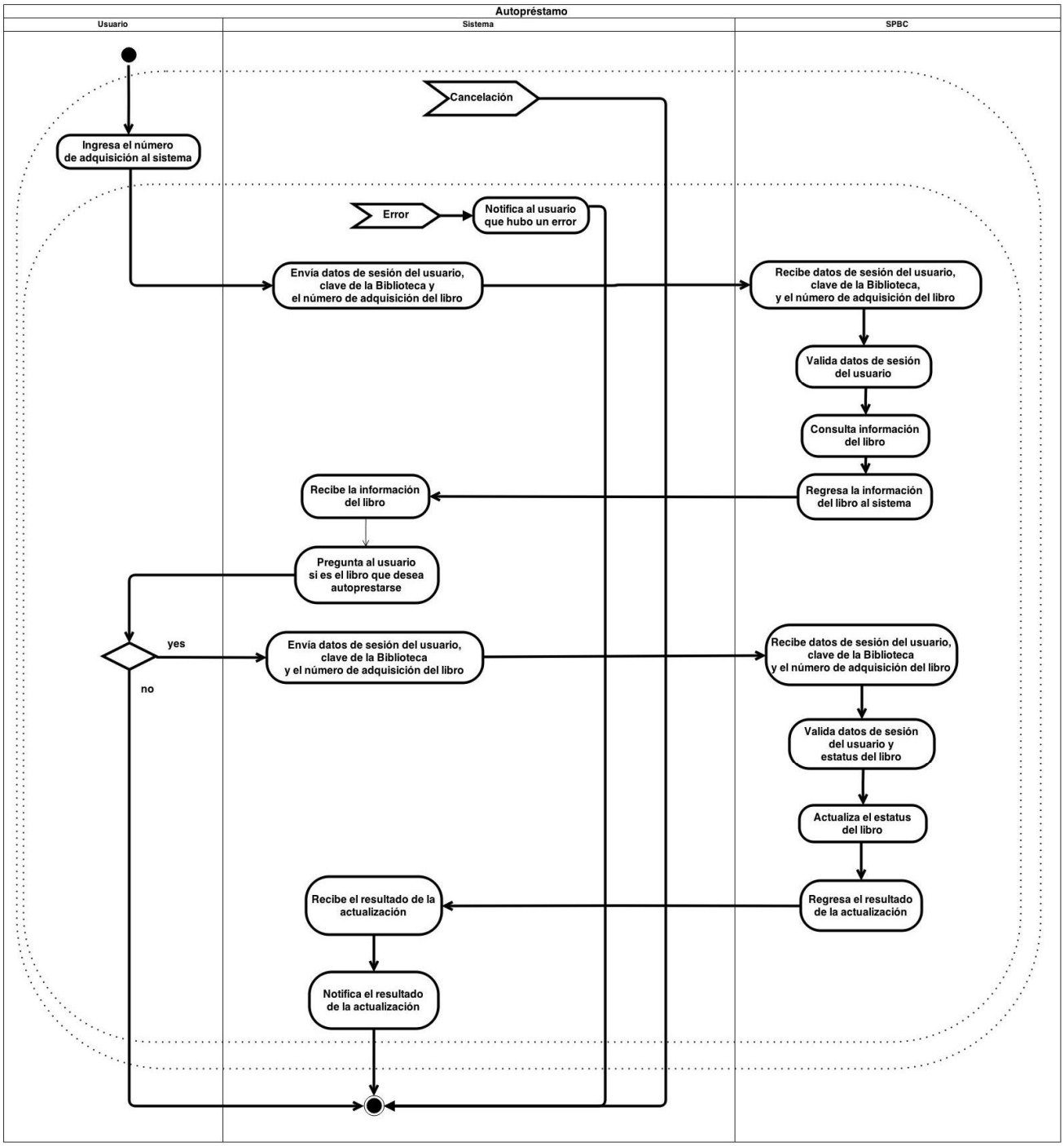


Ilustración 3-9 - Diagrama de actividades: Auto préstamo

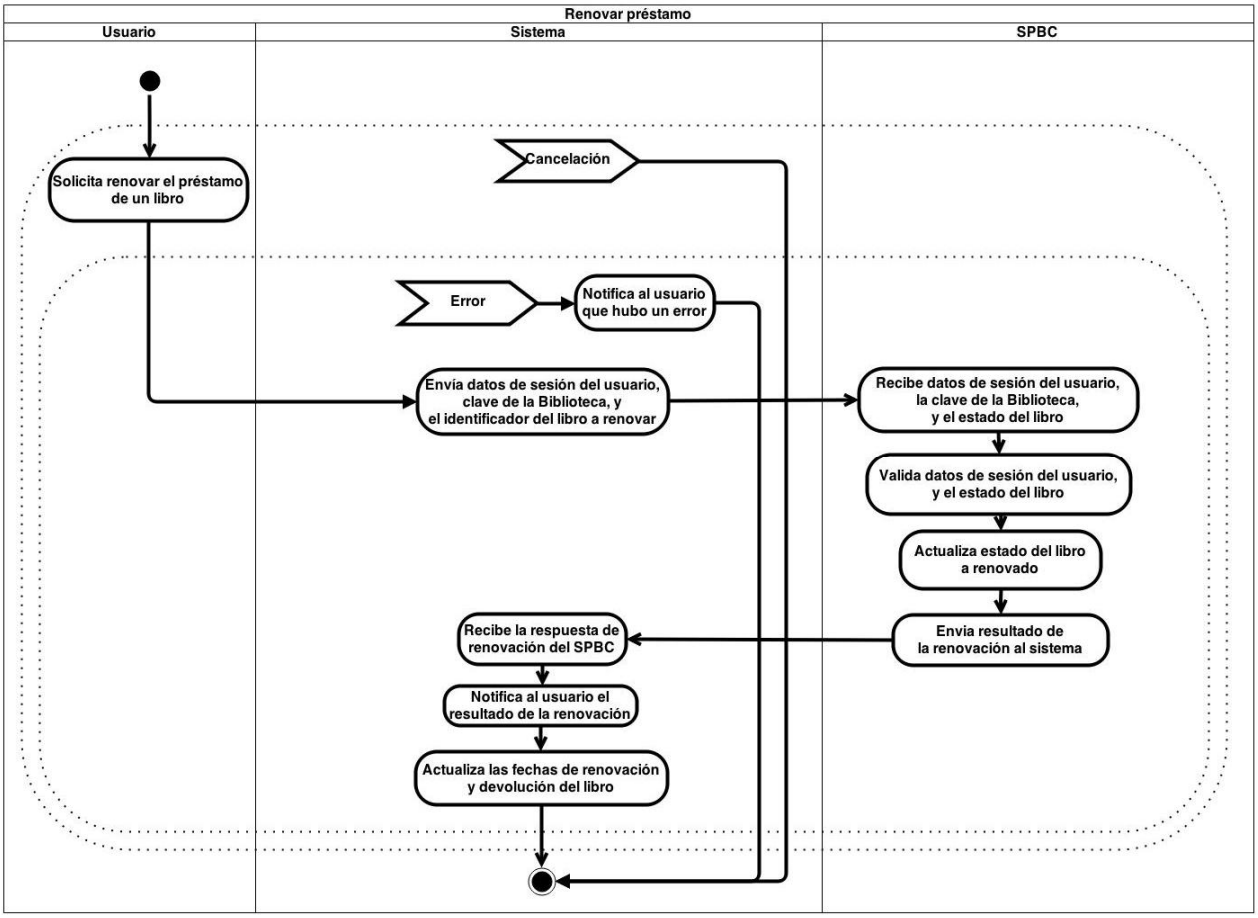


Ilustración 3-10 - Diagrama de actividades: Renovar préstamo

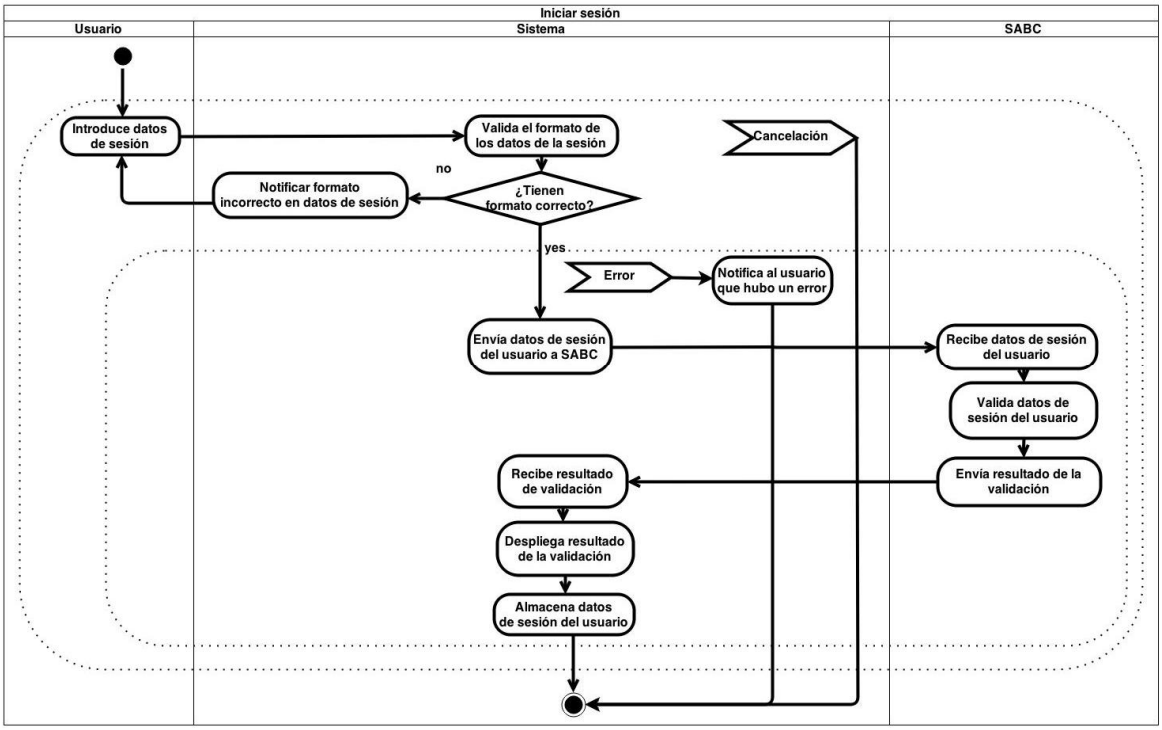


Ilustración 3-11 - Diagrama de actividades: Iniciar sesión

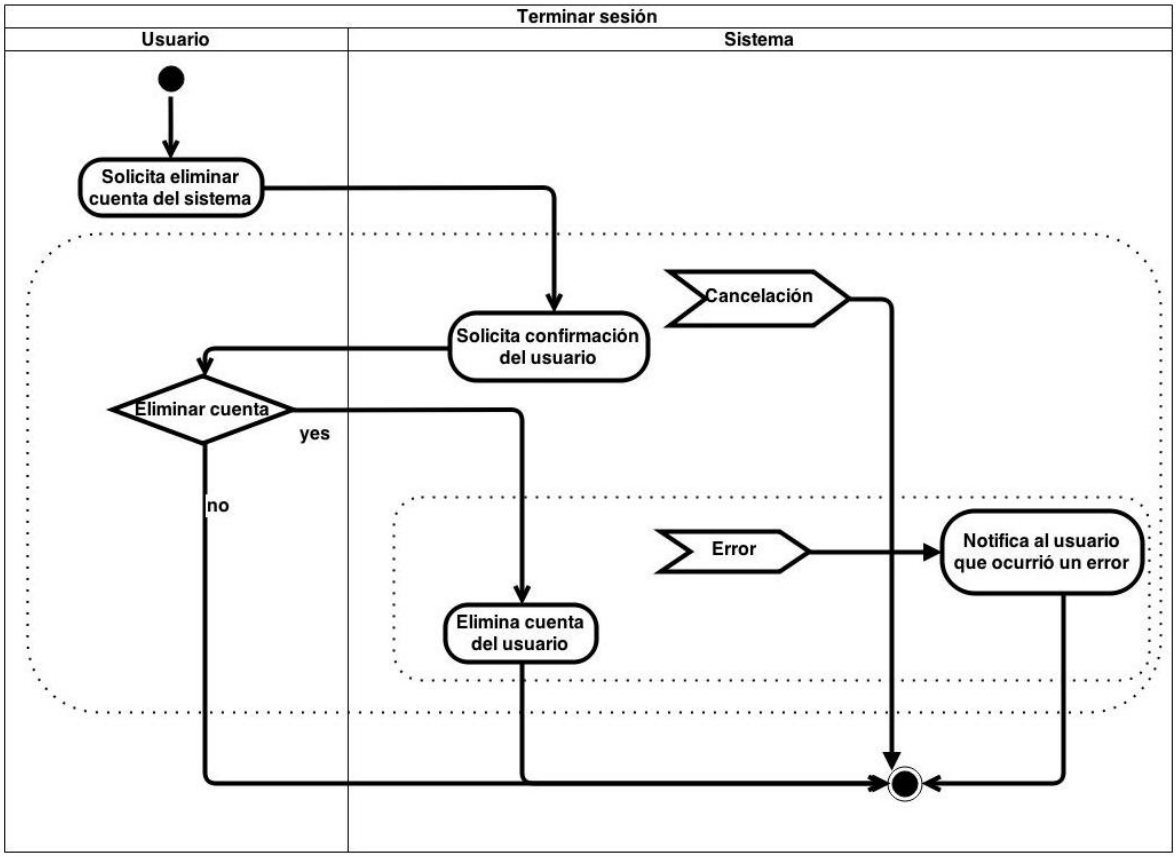


Ilustración 3-12 - Diagrama de actividades: Terminar sesión

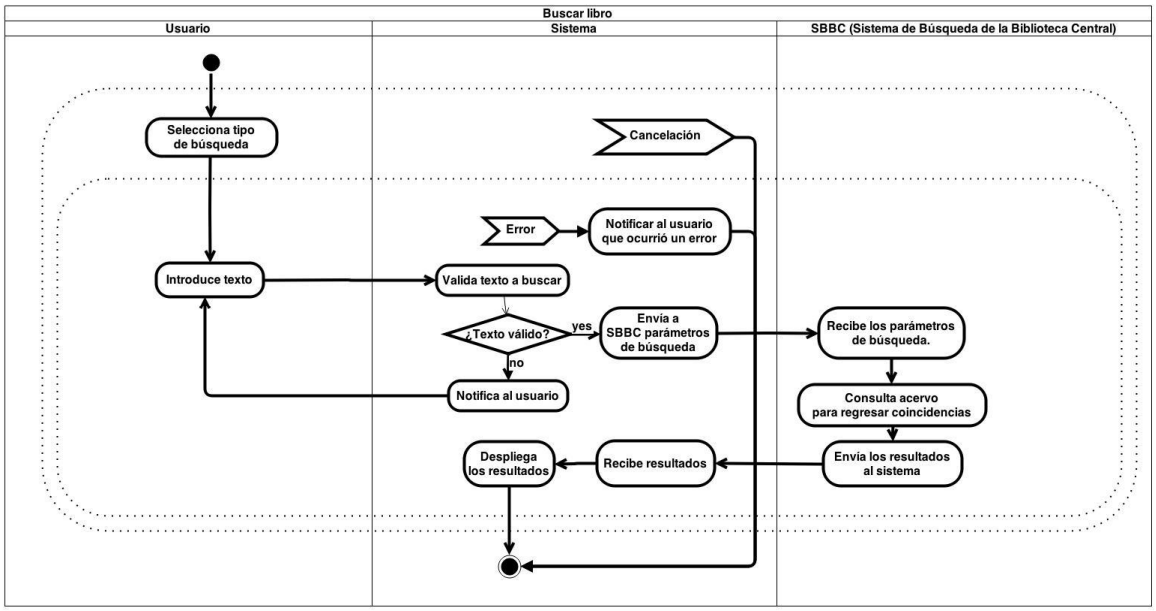


Ilustración 3-13 - Diagrama de actividades: Buscar libro

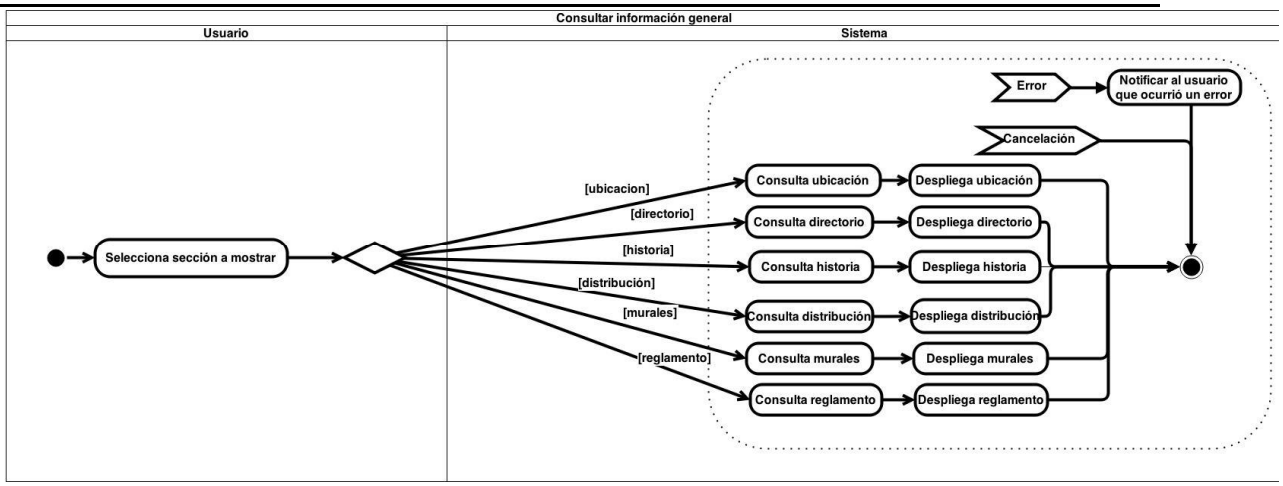


Ilustración 3-14 - Diagrama de actividades: Consultar información general

Con los todos los elementos previamente definidos (requerimientos funcionales y no funcionales, diseño de la navegación e interfaz de usuario, diagramas de casos de uso y de actividades) es posible pasar al siguiente paso del proceso de ingeniería de software establecido para la aplicación, es decir el proceso de desarrollo.

3.3 Desarrollo

Dadas las características del desarrollo de la aplicación, es posible clasificarla como un modelo de desarrollo incremental, es decir que los módulos de la aplicación fueron desarrollados uno a la vez. Cada módulo pasó por sus propias fases de requerimientos, diseño, implementación y pruebas. A continuación, se muestra la tabla 3-11 con la organización en paquetes del código fuente de la aplicación.

Tabla 3-11 - Descripción de los paquetes de la aplicación

Paquete	Descripción
mx.saudade.bc.activity	Contiene las actividades para navegar por la aplicación.
mx.saudade.bc.fragment	Contiene los fragmentos que encapsulan cada vista de la aplicación.
mx.saudade.bc.fragment.dialog	Contiene los diálogos necesarios para desplegar el estado de las tareas que se están ejecutando en segundo plano; o para notificar el resultado de una operación.
mx.saudade.bc.adapters	Contiene adaptadores que permiten customizar el despliegue de listas en las vistas.
mx.saudade.bc.ui	Contiene vistas personalizadas.
com.google.zxing.integration.android	Contiene los componentes necesarios para conectar la biblioteca zxing con la aplicación.
mx.saudade.bc.vo	Contiene los modelos utilizados para encapsular información.
mx.saudade.bc.parsers	Contiene clases que permiten extraer contenido en formato xml a objetos.
mx.saudade.bc.web	Contiene las clases necesarias para entablar comunicación con los servicios web.
mx.saudade.bc.code	Contiene los códigos de respuesta devueltos por los servicios

	web.
mx.saudade.bc.requests	Contiene los componentes que encapsulan las solicitudes a los servicios web.
mx.saudade.bc.responses	Contiene los objetos que encapsulan la respuesta obtenida por los servicios web.
mx.saudade.bc.utils	Diversas clases utilitarias.

3.3.1 Tecnologías empleadas

Android SDK – Kit de desarrollo de aplicaciones para dispositivos Android. Google ofrece de forma gratuita el SDK oficial, una serie de drivers, herramientas y recursos diversos para programar en Android, su sistema operativo móvil. El kit de desarrollo puede obtenerse en el paquete Developer Tools donde además se incluye el IDE Eclipse o descargarse de forma independiente para utilizar un editor diferente o realizar otras tareas.

En Android SDK se incluyen las herramientas necesarias para dar los primeros pasos programando para esta plataforma: distintas APIs facilitadas por Google tanto para el control de las funciones del dispositivo como para la integración de servicios, un depurador, un emulador para testear las aplicaciones y toda la documentación necesaria para dar tus primeros pasos programando en Android.

Eclipse JUNO – Entorno de desarrollo. Eclipse es una plataforma de desarrollo, diseñada para ser extendida de forma indefinida a través de plugins. Fue concebida desde sus orígenes para convertirse en una plataforma de integración de herramientas de desarrollo. No tiene como objetivo un lenguaje específico, sino que es un IDE genérico, aunque goza de mucha popularidad entre la comunidad de desarrolladores del lenguaje Java usando el plug-in JDT que viene incluido en la distribución estándar del IDE. Para el desarrollo de aplicaciones Android con el IDE Eclipse se utiliza el plugin ADT (Android Development Tools).

Android Development Tools (ADT). Es un plugin para el IDE Eclipse que amplía las capacidades de Eclipse para que pueda configurar rápidamente nuevos proyectos enfocados en la plataforma Android, crear una interfaz de usuario de la aplicación, agregar paquetes basados en la API del Framework de Android, depurar sus aplicaciones usando las herramientas del SDK de Android, e incluso generar y exportar archivos APK con el fin de distribuir la aplicación.

BitBucket. Servicio de hospedaje web de proyectos. Bitbucket es un sitio de alojamiento para los sistemas de control de versiones distribuidos (DVCS) Git y Mercurial. La oferta de servicios incluye un gestor de incidencias y wiki, así como la integración con una serie de servicios populares como Basecamp, Flowdock y Twitter.

Git. Sistema de control de versiones. Git es un sistema de control de versiones distribuido, es gratuito y de código abierto diseñado para manejar desde pequeños a grandes proyectos con rapidez y eficiencia.

GIMP. Manipulación de imágenes. GIMP es un acrónimo de GNU Image Manipulation Program. Es un programa de distribución gratuita para tareas como retoque fotográfico, composición de imágenes y creación de imágenes.

Balsamiq Mockups. Creación de modelos o maquetas de la aplicación.

Draw.IO – Creación de diagramas.

3.4 Verificación y validación

Dado que el desarrollo fue de tipo incremental, las tareas de verificación y validación se realizaron tras finalizar cada módulo de la aplicación y su integración con el resto de los componentes existentes.

Primero, el desarrollador fue responsable de verificar que los módulos construidos cumplen con las especificaciones, requerimientos funcionales y no funcionales. Posteriormente, los funcionarios de la Biblioteca Central, en su rol de clientes de la aplicación, se encargaron de realizar las pruebas correspondientes y validar que la aplicación efectivamente cumple con sus expectativas.

CAPÍTULO 4 - RESULTADOS

Como resultado se ha generado la aplicación móvil cumpliendo con los lineamientos establecidos para su funcionamiento en el contexto de la Biblioteca Central.

Para efectos de implementación y mantenimiento, la Biblioteca Central ha recibido el código fuente y todos los componentes del proyecto para poder generar por propia cuenta el paquete de la aplicación, el archivo APK, para su posterior distribución. El proceso de distribución se realizará, si no hay impedimentos para ello, a través del sitio oficial de aplicaciones móviles de la UNAM (https://apps.unam.mx/).

Sin embargo, recientemente, se ha solicitado apoyo para llevar a cabo algunos cambios en la aplicación a nivel de la conexión con los servicios de la Biblioteca, los cuales es obviamente son mucho más sencillos de hacer por quienes se encargaron del desarrollo. Con esto se demuestra que la evolución del software es un proceso continuo que depende en gran medida de la comunicación con el cliente para temas de soporte y el monitoreo de la propia aplicación.

La implementación para el público en general de la aplicación móvil aún se encuentra pendiente debido a que las pruebas realizadas se han ejecutado sobre servicios de prueba de la institución, por lo que hasta que no se encuentren los servicios definitivos disponibles no se podrá llevar a cabo un préstamo real. Se tiene la expectativa de iniciar la implementación durante los meses de mayo y junio del presente año 2015.

La ampliación del proyecto, en caso de realizarse por parte de la Biblioteca Central, consistiría en el desarrollo de la aplicación para el sistema operativo iOS, el sistema operativo de los dispositivos iPhone y iPad del fabricante Apple.

Otro posible camino es la realización de aplicaciones móviles para otras bibliotecas de la UNAM que se interesen en el proyecto de la Biblioteca Central. Lamentablemente, y dados los requerimientos inícials y las condiciones particulares del proyecto, es poco probable que se pueda reutilizar la mayoría del código existente actualmente pues cada biblioteca de la UNAM funciona de forma independiente y tienen sus propias reglas de negocio.

Finalmente, en las siguientes páginas, se anexan capturas de pantalla de la aplicación resultado del presente trabajo de titulación. (Ilustraciones 4-1 a 4-15).

Ilustración 4-1 - Splash (bienvenida)

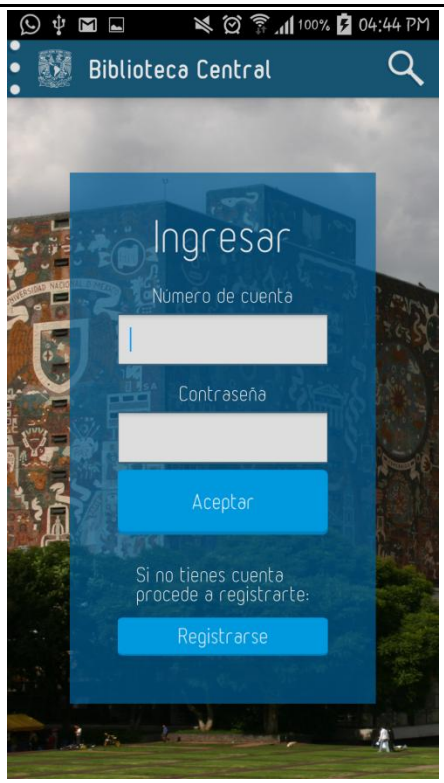


Ilustración 4-2 - Autenticación del usuario

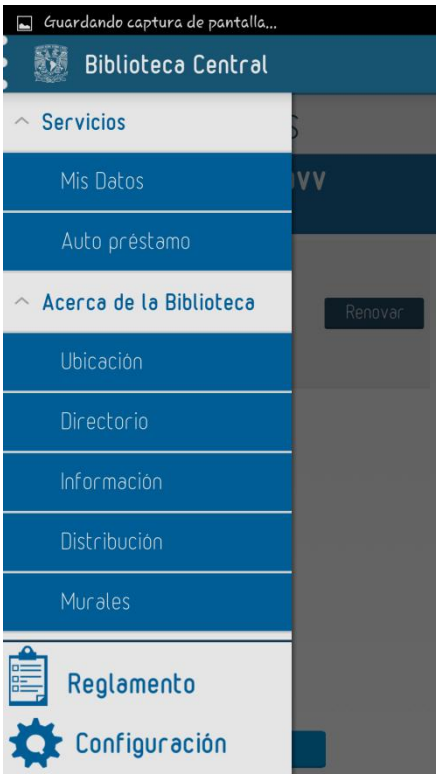


Ilustración 4-3 - Menú principal

Ilustración 4-4 - Mis datos

Ilustración 4-5 - Auto préstamo

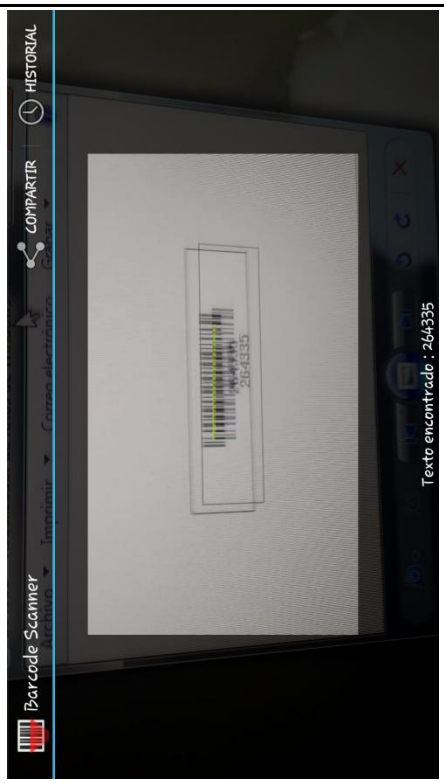


Ilustración 4-6 - Escanear número de adquisición

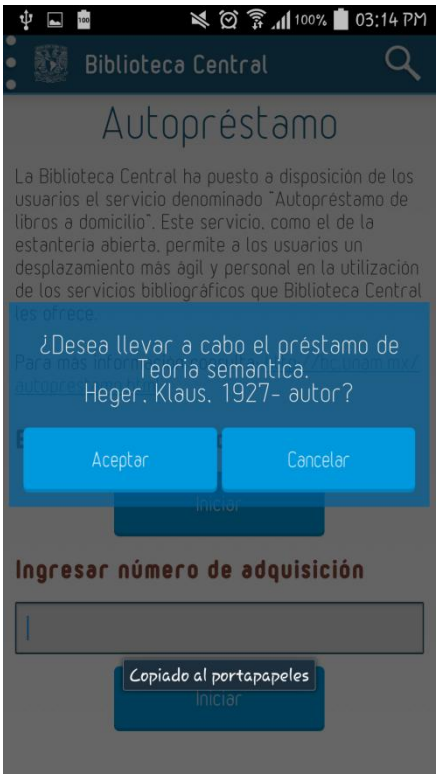


Ilustración 4-7 - Mensaje de confirmación

Ilustración 4-8 - Ubicación de la Biblioteca

Ilustración 4-9 - Directorio

Ilustración 4-10 - Información

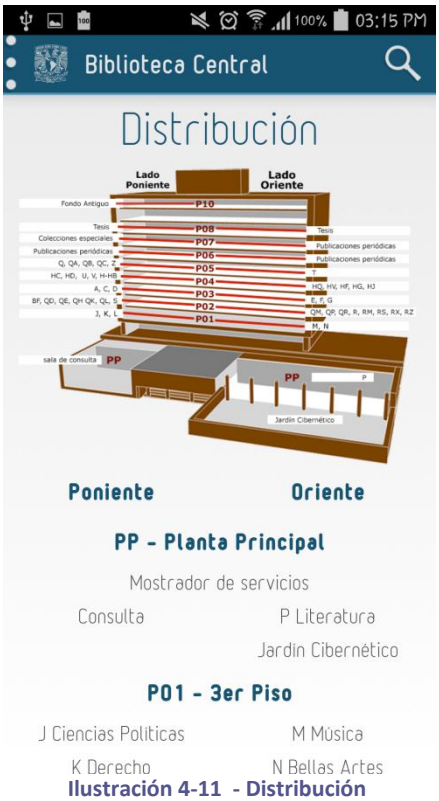


Ilustración 4-11 - Distribución

Ilustración 4-12 - Murales

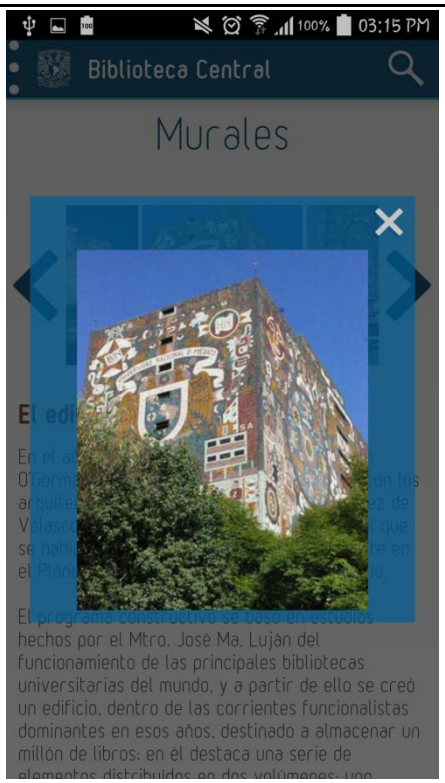


Ilustración 4-13 - Mural

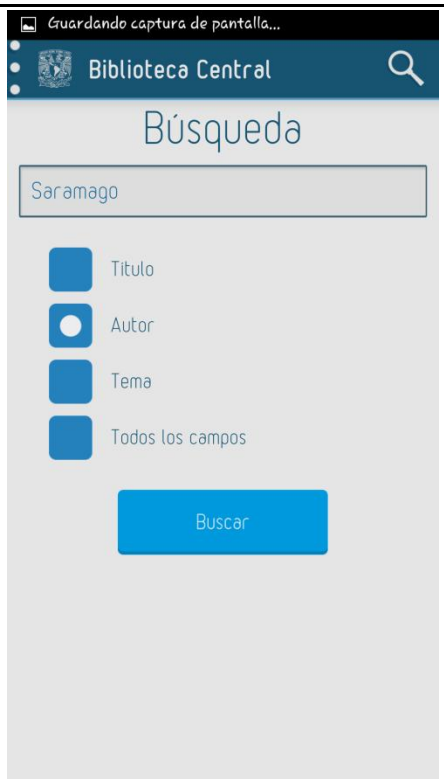


Ilustración 4-14 - Búsqueda

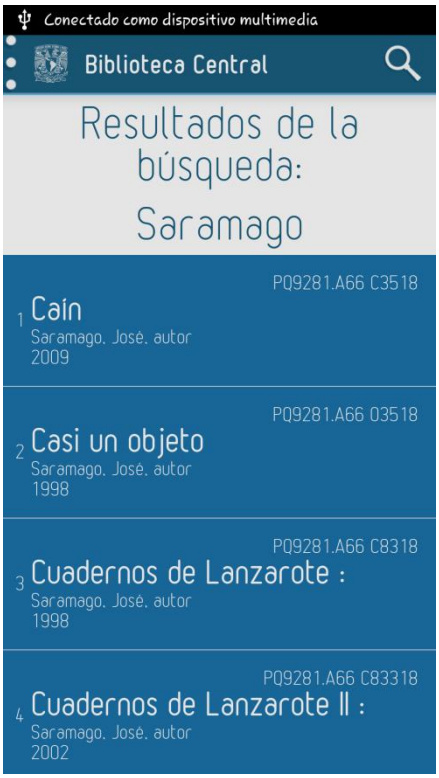


Ilustración 4-15 - Resultados de la búsqueda

Conclusiones

José:

Tras el desarrollo de la aplicación móvil de la Biblioteca Central y la revisión de los procesos de ingeniería de software, se puede evaluar que realizar proyectos de software de forma satisfactoria es una meta que depende de muchos factores y de todos los participantes en el ciclo de vida de la aplicación. La comunicación entre el equipo de ingeniería y los interesados de la aplicación resulta ser una parte fundamental para el éxito de un proyecto y poder medir de forma constante el progreso, además de corroborar que efectivamente el equipo de ingeniería ha comprendido las reglas del negocio y los objetivos del cliente.

El proyecto pasó por las etapas de levantamiento y análisis de requerimientos, diseño de los componentes de software y de los componentes visuales, desarrollo de la aplicación y finalmente las pruebas de verificación y validación. Sin embargo, en el ciclo de vida del software se encuentran otros procesos posteriores a la entrega del proyecto. En este caso el cliente, la Biblioteca Central, se encargará de la administración del proyecto y de los procesos siguientes, es decir la implementación y liberación del proyecto para uso de la comunidad universitaria, así como también se encargarán del proceso de mantenimiento y ampliación del proyecto en caso de que tenga un impacto significativo.

Angélica:

Así como en años recientes, una empresa sin proyección en Internet estaba en desventaja contra sus competidores que si tenían presencia en la red, actualmente ocurre lo mismo con las aplicaciones móviles. Aquellas empresas que no cuentan con una aplicación móvil se encuentran en desventaja. Además, el beneficio de estar siempre en contacto con tus clientes es muy atractivo, como nunca antes. Con esta aplicación se busca fortalecer la imagen de la Biblioteca para con sus usuarios, y su compromiso de estar siempre a la vanguardia.

Recomendaciones para la puesta en marcha de la aplicación:

- Abrir un espacio donde recibir retroalimentación de los usuarios para efectuar mejoras a la aplicación.

- Monitoreo constante para conocer el comportamiento de los usuarios y aplicar las medidas adecuadas a ello (peticiones, tiempo de respuesta, horario de uso, tipo de dispositivos, etc.). Para esto es posible usar herramientas como Google Analytics.

En este punto del desarrollo es difícil medir el impacto de la aplicación porque aún no se encuentra en producción. Pero el propósito de esta aplicación es mejorar tiempos de atención para con los estudiantes al efectuar sus préstamos, renovaciones y agilizar las consultas del acervo con el servicio de búsqueda.

Glosario

Término	Definición
Biblioteca Central	Entidad académico-administrativa dependiente de la Dirección General de Bibliotecas, creada para coadyuvar a las tareas sustantivas de la Universidad Nacional Autónoma de México (UNAM) por medio de los servicios bibliotecarios que ofrece a su comunidad.
Servicios bibliotecarios	Son todas aquellas actividades académicas, técnicas y administrativas de la BC, encaminadas a satisfacer las necesidades de información de los usuarios
Usuario	Toda persona o institución que solicite los servicios de la BC. Puede ser usuario interno o externo. Los usuarios internos son todos aquellos que conforman la comunidad UNAM, que cumplan con los requisitos establecidos en el presente Reglamento y obtengan su registro en la BC, los cuales gozarán de todos los servicios. Los usuarios externos son todos aquellos que no son miembros de la comunidad UNAM y usan los servicios de la Biblioteca Central.
Préstamo a domicilio	Consiste en la autorización que se otorga a los usuarios internos para llevar a su domicilio un máximo de tres libros por siete días naturales. El plazo podrá prorrogarse, ya sea presencialmente o por vía telefónica por dos períodos iguales siempre y cuando el préstamo no esté vencido o el material no esté en lista de espera por otro usuario y podrá devolverse en la fecha establecida o antes de la fecha que se indique.
Acervo	Conjunto de obras de consulta, libros, publicaciones periódicas, tesis, documentos del Fondo Antiguo, colecciones especiales, y materiales audiovisuales que pertenecen a la BC.
Clave	Identificador único asignado a un usuario al registrarse en la BC.
Contraseña	Código que el usuario utiliza en conjunto con su clave, para autenticarse en la BC.
Clave biblioteca	Identificador único asignado a cada entidad académico-administrativa que pertenece a la Dirección General de Bibliotecas.
Auto préstamo	Servicio, como el préstamo en estantería abierta, que se lleva a cabo en el dispositivo móvil del usuario. Permite a los usuarios un desplazamiento más ágil y personal en la utilización de los servicios bibliográficos que Biblioteca Central les ofrece.
Renovación	Extender la duración del préstamo por un periodo adicional de 7 días hábiles.
Periodo del préstamo	Período en el que un usuario dispone de un bien de la BC para su consulta personal. El periodo de préstamo en general son 7 días naturales, pero puede extenderse al renovar el préstamo. El período del préstamo puede ser menor si el usuario devuelve el material antes de los 7 días naturales.
Número de adquisición	Identificador único asignado a cada libro.
Código de barras	Sinónimo de Número de adquisición.
Distribución	Colocación del material bibliográfico, catalogado por tema; a lo largo del edificio de la BC.
Datos de sesión	Clave y contraseña que permiten a un usuario identificarse en el sistema.

Referencias

2, 3, 4, 16. SOMMERVILLE Ian. Software Engineering, Addison-Wesley, Eight Edition, 2007.

6, 7. LARMAN Craig. UML Y PATRONES. Una introducción al análisis y diseño orientado a objetos y al proceso unificado, Pearson, Segunda edición, 2003.

5, 8, 15. BOURQUE Pierre, FAIRLEY Richard, SWEBOK V3.0 Guide to the Software Engineering Body of Knowledge, IEEE Computer Society, Third Edition, 2014.

1. http://pascal.computer.org/sev_display/index.action, *Software and Systems Engineering Vocabulary,* (17 de Febrero de 2015)

9, 10, 11, 12, 13, 14. http://istqbexamcertification.com/what-are-the-software-development-models/, What are the Software Development Models? (17 de Febrero de 2015)

Anexo: Manual de usuario de la aplicación

APLICACIÓN MÓVIL DE LA BIBLIOTECA CENTRAL

MANUAL DE USUARIO

Versión 1.0

Marzo 2015

INTRODUCCIÓN AL MANUAL DE USUARIO

El presente manual sirve como referencia de uso para la aplicación móvil de la Biblioteca Central. Con esta aplicación es posible que sus usuarios conozcan más acerca de la institución, así como consultar sus préstamos actuales y efectuar auto préstamos.

Las imágenes y descripciones van orientadas a la primer versión de la aplicación, que fue implementada para smartphones Android con api superior a la versión 4.

Como descripción general, en el primer capítulo se explora la navegación y funciones de la aplicación, en el siguiente capítulo se detallan los errores que pueden generarse al usar la aplicación y la solución pertinente para resolverlos. Finalmente, en el último capítulo se añaden los medios de contacto para reportar un problema al usar la aplicación.

En cada descripción se utiliza la siguiente iconografía:

Para efectuar esta operación, necesitas tener acceso a Internet.

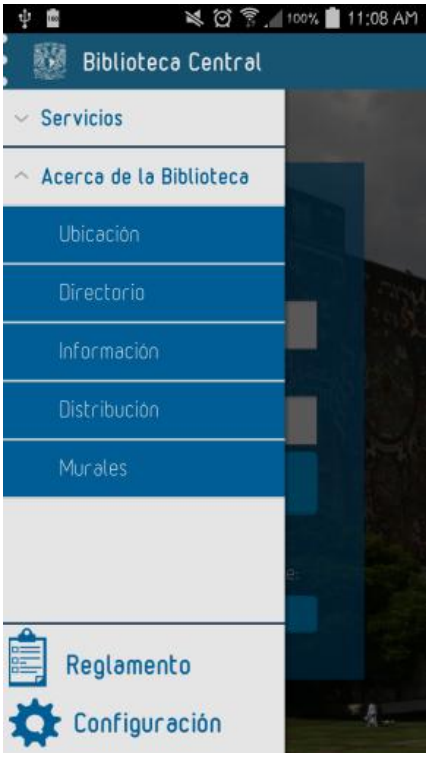
Para acceder a esta operación, necesitas tener una sesión válida registrada en el dispositivo.

MENÚ PRINCIPAL

Desde el menú principal podrás acceder a las funcionalidades principales de la aplicación.

Menú principal

En la parte superior izquierda verás el ícono del menú principal.



Para desplegar el menú, presiona la imagen del escudo de la universidad.

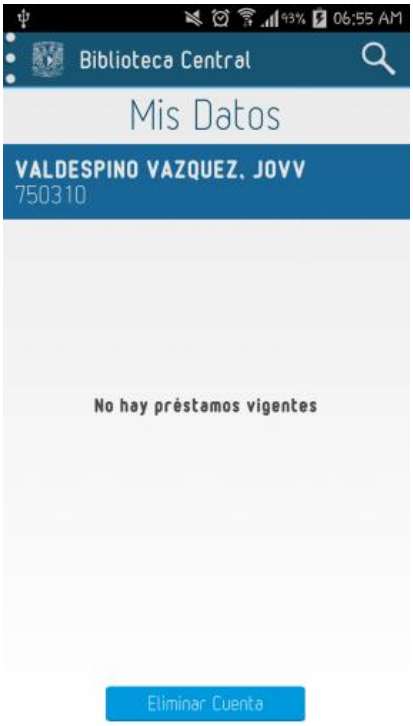
SESIÓN

Al iniciar la aplicación por primera vez, se despliega la pantalla de inicio de sesión.

Iniciar sesión

Para iniciar sesión en la aplicación, introduce tu clave y contraseña.

Es necesario que estés conectado a Internet.



Si los datos proporcionados son correctos, pasarás a la pantalla de **Mis datos**.

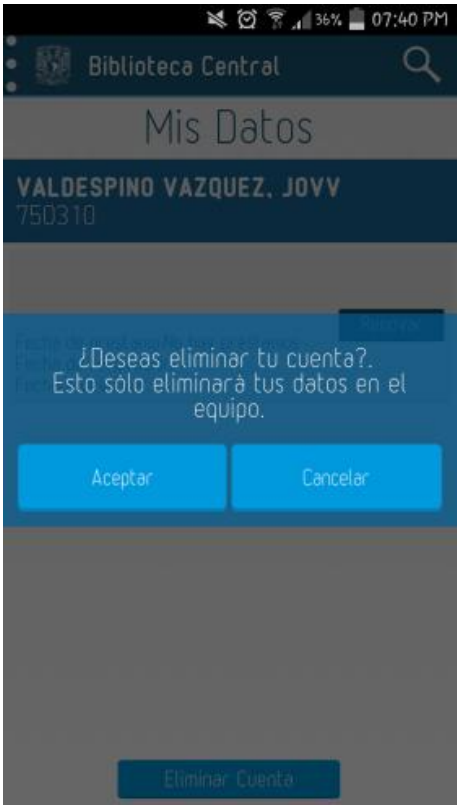
En siguientes ocasiones que inicies la aplicación, no será necesario que proporciones nuevamente tu número de cuenta y contraseña; y se desplegará automáticamente la pantalla de **Mis datos**.

Registrarse

En caso de no contar con una cuenta puedes registrarte en línea.

Al presionar Registrarse, se desplegará un formulario que debes completar.

Borrar Sesión

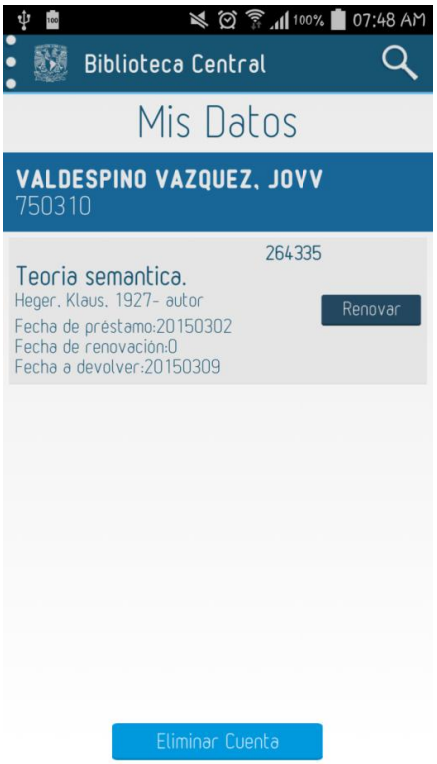


Para borrar tu sesión, presiona el botón de eliminar cuenta en la pantalla de **Mis datos**.
Se desplegará un diálogo con un mensaje de confirmación.
Presiona aceptar.

MIS DATOS

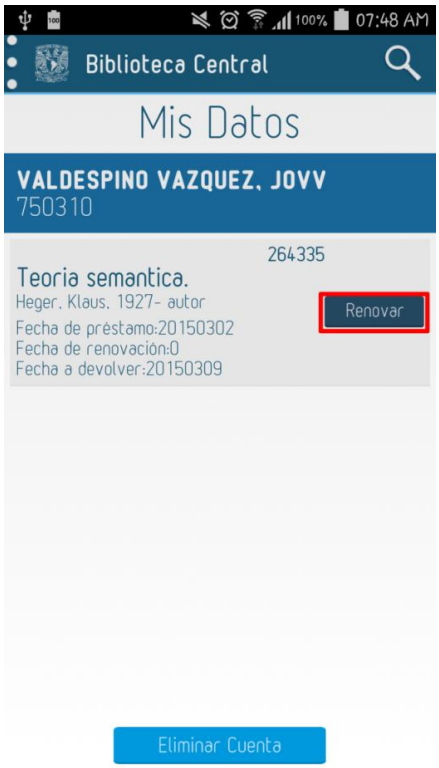
En la pantalla de mis datos puedes consultar y renovar tus préstamos vigentes.

Préstamos actuales



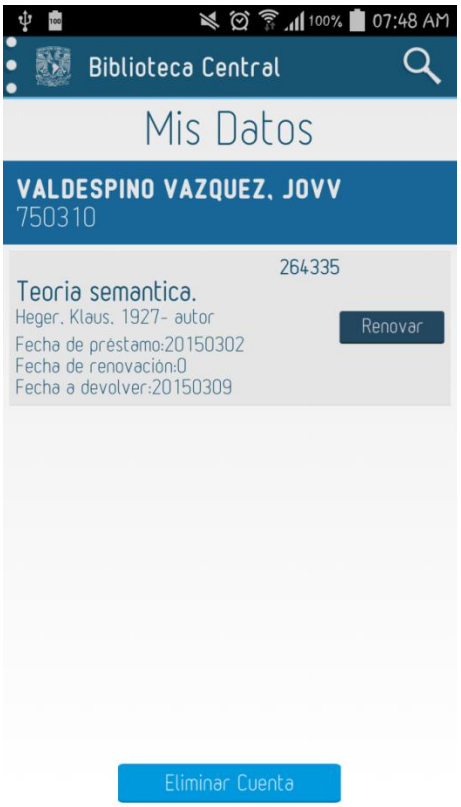
Una vez que has iniciado sesión, se mostrará la pantalla de **Mis datos**. En ella verás tus préstamos actuales.

Renovar préstamo

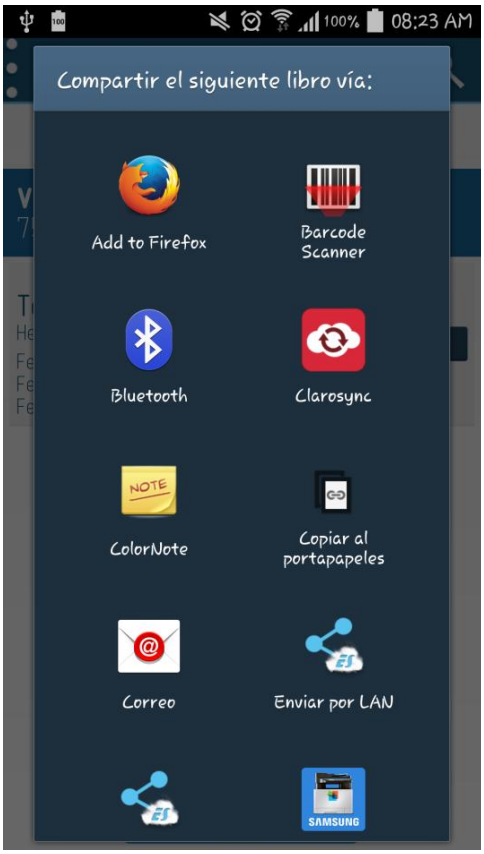


Para renovar un préstamo, presiona el botón de **Renovar**.

Compartir



Para compartir la información del libro, mantén presionado el elemento que corresponde al libro que deseas compartir.

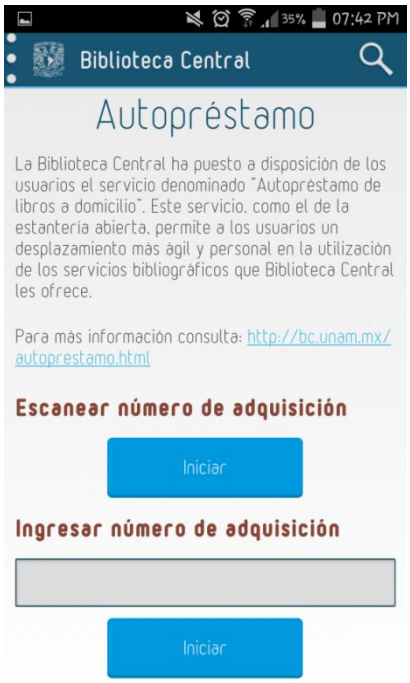


Pasados unos segundos, se te desplegará un listado donde podrás seleccionar la aplicación con la que deseas efectuar el préstamo.

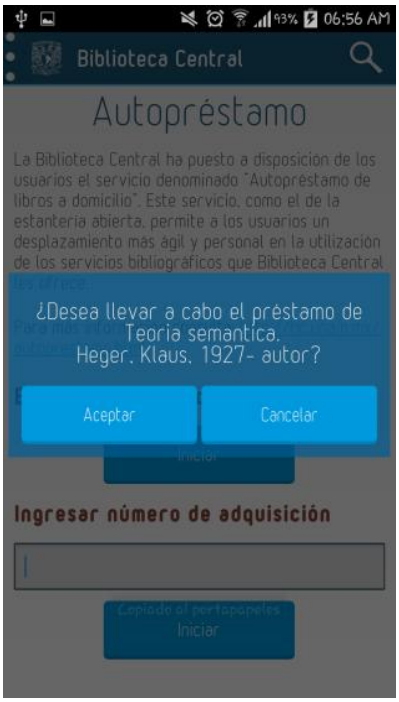
AUTO PRÉSTAMO

En la sección de auto préstamo podrás efectuar un préstamo directamente desde tu dispositivo.

Auto préstamos

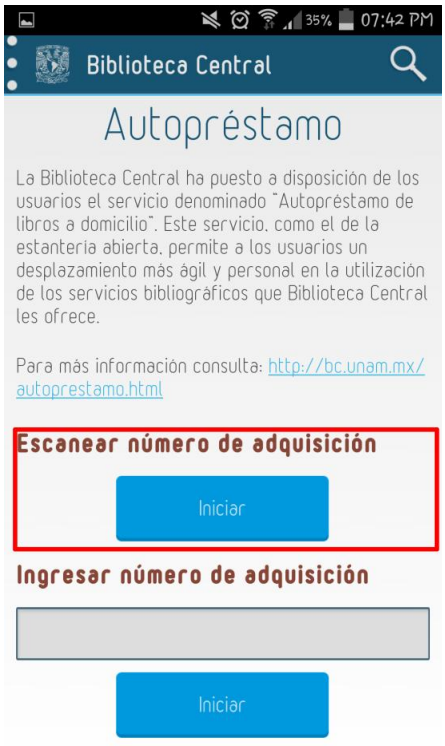


Para efectuar un auto préstamo, accede a la sección de **Autopréstamo**.
Para ingresar el número de adquisición puedes escanearlo a través de la cámara de tu smartphone, o capturarlo en el campo de texto.

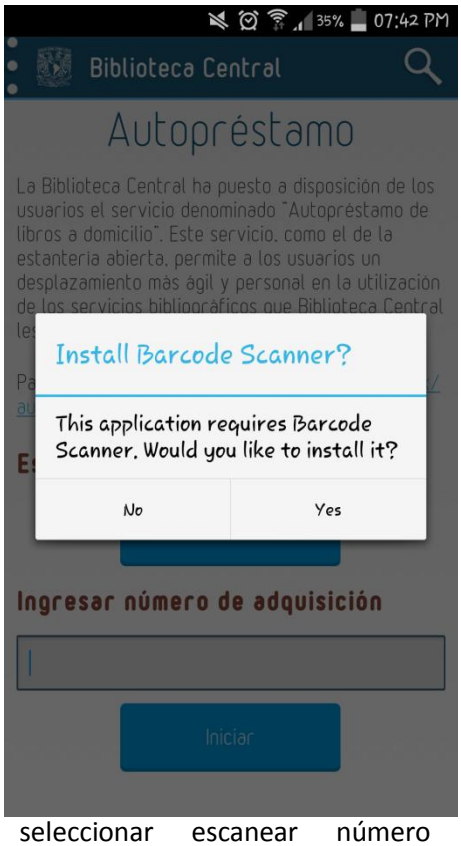


Ya sea que hayas capturado o escaneado el número de adquisición, se mostrará un diálogo preguntando si los datos ingresados corresponden al libro que deseas auto prestarte.
Si son correctos, presiona aceptar.

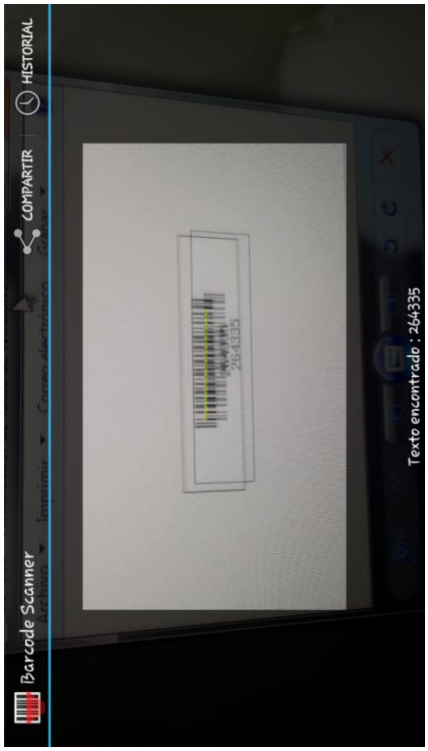
Escanear número de adquisición



Se deseas escanearlo, presiona iniciar.



Al seleccionar escanear número de adquisición, se te solicitará que instales la aplicación **Barcode Scanner.**



Una vez instalada la aplicación, escanea el código de barras del material que deseas auto prestarte.

Introducir número de adquisición

Si decides capturar el número de adquisición, introduce el valor en el campo de texto y presiona iniciar.

BÚSQUEDAS

La sección de búsquedas te permite buscar libros directamente en el acervo de la Biblioteca Central.

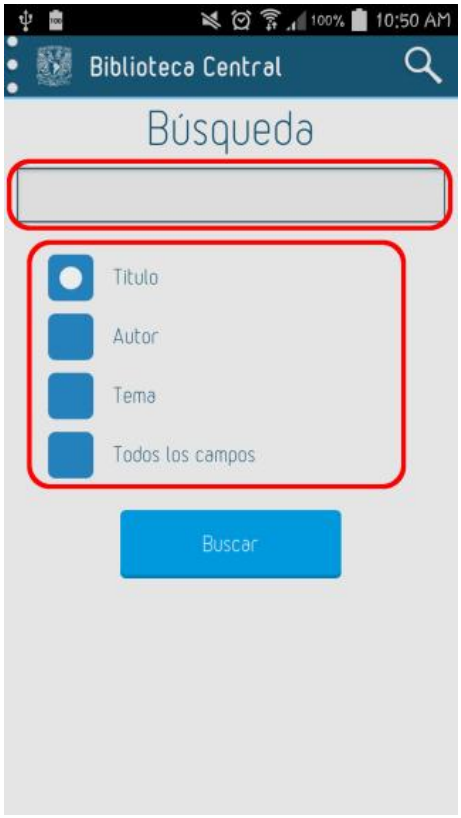
Realizar una búsqueda

Para llevar a cabo una búsqueda en particular, presiona el botón de Buscar.

No es necesario que hayas iniciado sesión en la aplicación.

Para llevar a cabo búsquedas en el acervo de la Biblioteca Central es necesario que estés conectado a Internet.

Esta opción se encuentra disponible en todas las pantallas de la aplicación.



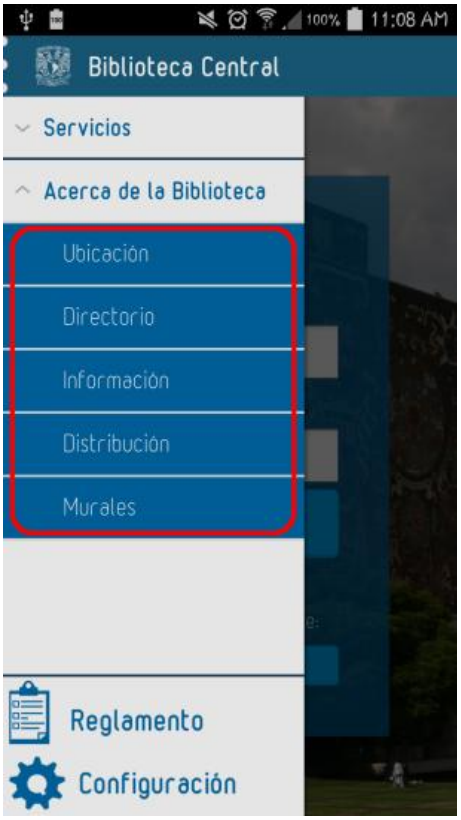
Al ingresar a la pantalla de Búsqueda, verás un formulario donde introducir el texto a buscar, y el criterio a valorar: título, autor, tema, todos los campos.

Si hubo coincidencias con los parámetros de búsqueda, verás una nueva pantalla con el listado de resultados.
Puedes desplazarte entre el listado con los botones de anterior/siguiente.

ACERCA DE LA BIBLIOTECA

En esta sección podrás conocer más acerca de la Biblioteca Central.

Menú de opciones



Para saber más acerca de la Biblioteca Central, puedes ingresar desde cualquier pantalla al menú donde podrás ver cada una de las categorías.

Para acceder a este contenido no es necesario que estés conectado a Internet.

Ubicación

En la pantalla de Ubicación podrás ver la ubicación de la Biblioteca Central y horarios de atención.

Directorio

En la pantalla de directorio verás nombre y correo electrónico del personal administrativo de la Biblioteca Central.

Información

En la pantalla de información verás información general de la Biblioteca Central, junto con su historia.

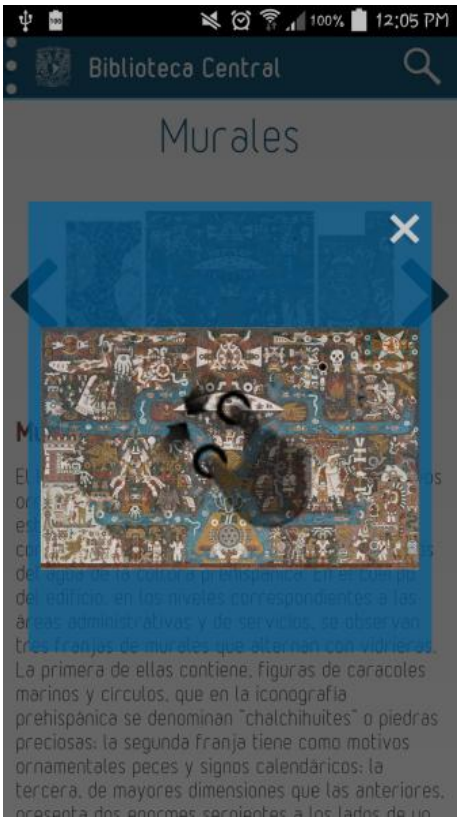
Distribución

En la pantalla de distribución podrás conocer la colocación del acervo en cada uno de sus pisos.

Para visualizar mejor la imagen, puedes aplicar un zoom para verla más grande.

Murales

En la pantalla de murales podrás conocer de manera general, información sobre los murales que cubren la Biblioteca Central.



Al seleccionar cada imagen podrás verla con más de detalle.

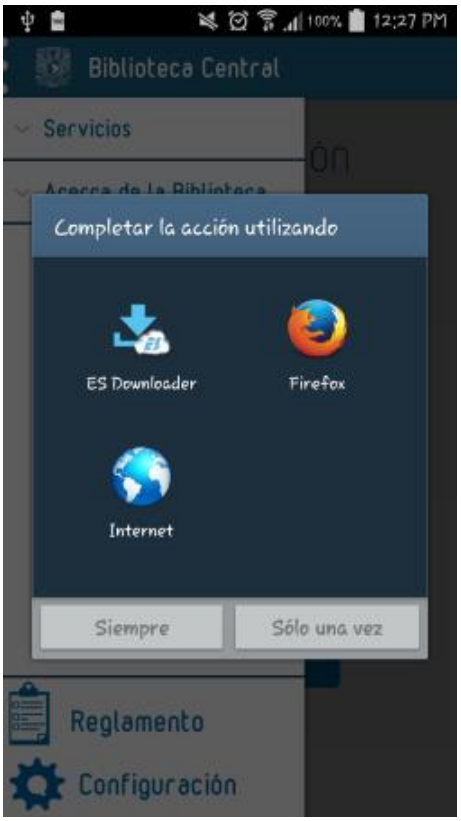
Para visualizar mejor la imagen, puedes aplicar un zoom.

REGLAMENTO

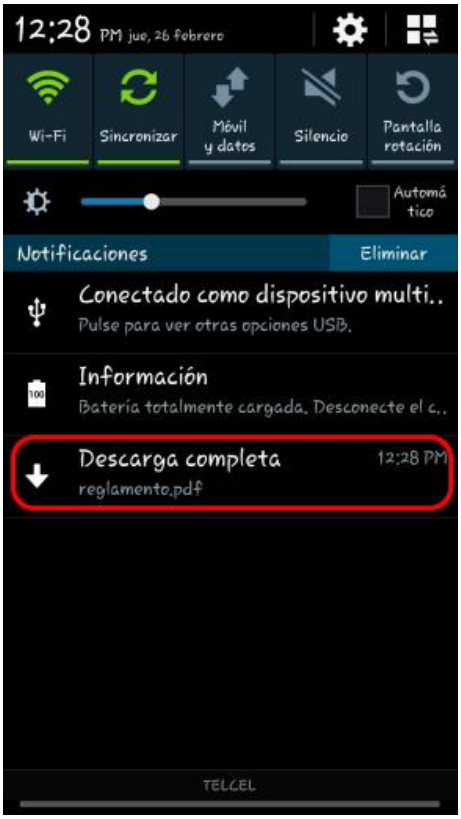
Reglamento vigente de la Biblioteca Central.

Descargar reglamento

Desde el menú principal de la aplicación podrás acceder al reglamento de la Biblioteca Central.



Al presionar Reglamento, se te preguntará con que aplicación deseas llevar a cabo la descarga.



Una vez finalizada, accede a tu menú de notificaciones y selecciona el archivo del reglamento para visualizarlo.

CONFIGURACIÓN

En esta sección podrás configurar notificaciones que te alertarán cuando sea la fecha de devolver alguno de tus préstamos vigentes.

Configurar notificaciones

En la pantalla de configuración de notificaciones puedes activar y personalizar notificaciones que te avisarán cuando sea el día en que tengas que devolver un libro.

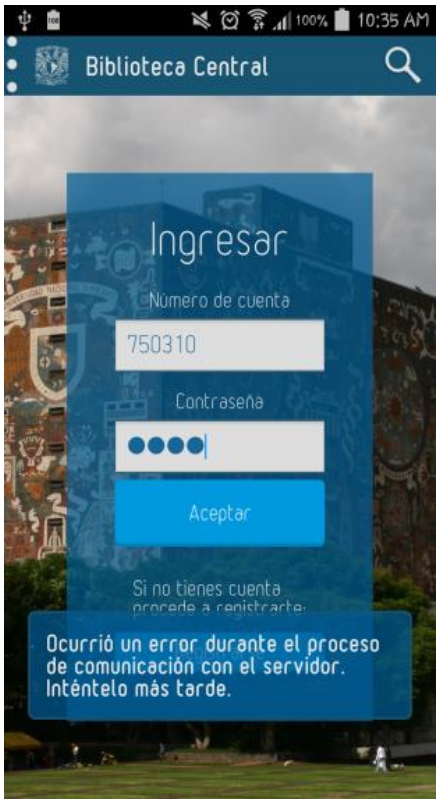
SOLUCIÓN DE PROBLEMAS

Errores generales

- Ocurrió un error durante el proceso de comunicación con el servidor. Inténtelo más tarde.

Este error ocurre cuando el servidor no puede atender satisfactoriamente la petición.

Intenta nuevamente hacer la solicitud tras pasados algunos minutos.



- Ha ocurrido un error al tratar de conectarse a Internet, verifica que estás conectado e inténtalo de nuevo.

Este error ocurre cuando no te encuentras conectado a Internet, activa tu wi-fi o datos e inténtalo de nuevo.

Errores en la sesión

- Credenciales incorrectas, verifique su información e inténtelo nuevamente.

Este error ocurre cuando los datos proporcionados no corresponden a un usuario válido en el sistema.

Verifica la información e inténtalo nuevamente.

Errores en prestamos / renovación

- ERROR: la clave y/o contraseña no son válidas.

Este error se presenta porque los datos de tu sesión no son correctos. Si es necesario, borra tu sesión e ingresa nuevamente.

- ERROR: no tienes permisos para llevar a cabo la renovación.
- ERROR: ya tienes dos renovaciones del libro.
- ERROR: alguno de los préstamos que tienes actualmente tiene fecha de devolución extemporánea.

Estos errores se presentan porque el estado de tu cuenta en la Biblioteca Central no te permite realizar la renovación.
Acude a cualquier módulo de ayuda para aclarar o solucionar el problema.

- ERROR: no existe el registro del préstamo.
- ERROR: no fue posible llevar a cabo la renovación.

Estos errores ocurren porque existen inconsistencias en el préstamo.
Acude a cualquier módulo de ayuda para aclarar o solucionar el problema.

Errores en autopréstamo

- ERROR: clave y/o contraseña no validas.

Este error se presenta porque los datos de tu sesión no son correctos. Si es necesario, borra tu sesión e ingresa nuevamente.

- ERROR: la vigencia de tu cuenta de usuario es extemporánea.
- ERROR: no tienes permisos para llevar a cabo el préstamo.
- ERROR: ya tiene 3 préstamos vigentes.
- ERROR: alguno de los préstamos que tienes actualmente tiene fecha de devolución extemporánea.

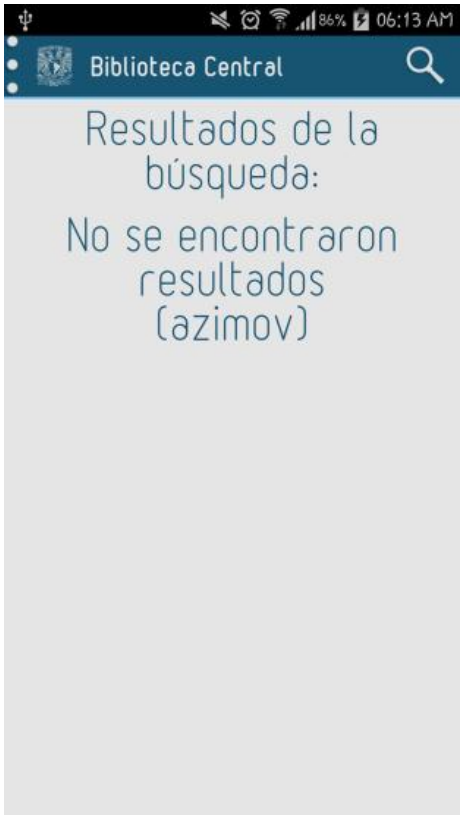
Estos errores se presentan porque el estado de tu cuenta en la Biblioteca Central no te permite realizar el auto préstamo.
Acude a cualquier módulo de ayuda para aclarar o solucionar el problema.

- ERROR: el libro no está dado de alta en el acervo.
- ERROR: El libro no puede prestarse a domicilio.
- ERROR: no fue posible llevar a cabo el préstamo.

Estos errores ocurren porque existen inconsistencias en el material.

Acude a cualquier módulo de ayuda para aclarar o solucionar el problema.

Errores en las búsquedas



En caso de no encontrar resultados, puedes regresar a la pantalla anterior e introducir nuevamente los parámetros de búsqueda.

CONTACTO

Para cualquier duda o comentario relacionado a la aplicación, favor de comunicarse a los siguientes correos:

Ingeniero Daniel Corte García

tesis@dgb.unam.mx

Maestro Jovv Valdespino Vázquez

jovval@unam.mx

Angélica Méndez Vega

angie@saudade.mx