UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valéncia

Desarrollo de una aplicación de Gestión de Quejas y Solicitudes

Proyecto Final de Carrera

Ingeniería Informática

Autor: Jaime Ferrer Sánchez

Directores: Jose Vicente Ballester Server

Manuel Herrero Más

01/09/2013

Resumen

El objetivo del presente PFC es la creación de una aplicación cuya funcionalidad es gestionar las quejas recibidas en el ayuntamiento de Torrent por parte del ciudadano, para conseguirlo he aplicando los conocimientos adquiridos durante los estudios universitarios. Dicha aplicación estandariza los procesos administrativos de análisis de las quejas y de contestación al ciudadano.

Palabras clave: Gestión, Quejas, Ayuntamiento

A mis padres, pues sin ellos no estaría hoy aquí.

A Voro, por la guía ofrecida durante el desarrollo del presente proyecto.

A Manuel Herrero, por haber confiado en este becario y haber sido una parte indispensable en su formación.

A José Vicente Ballester por aceptar formar parte de este proyecto y los consejos ofrecidos para la redacción del presente documento.

Y sobre todo a Ester, por su paciencia, el cariño ofrecido en estos años y hacerlo siempre todo más fácil.

Tabla de contenidos

1. Introducción

El objetivo de este proyecto final de carrera es aplicar los conocimientos adquiridos a lo largo de los estudios universitarios a un entorno de producción de software real, haciendo especial énfasis en la Ingeniería de requisitos, actividad fundamental para lograr implementar un software de calidad que responda a las necesidades de los usuarios.

Como aplicación práctica he escogido la creación de una aplicación de gestión. La aplicación a desarrollar tiene como misión permitir al ayuntamiento de Torrent poder contestar al ciudadano en un corto periodo de tiempo y de una forma estandarizada. Hasta el momento, en el Ayuntamiento no se disponía de una gestión formal. Una persona debía encargarse de recuperar todas las quejas presentes del sistema, solicitar información al departamento adecuado en caso de ser necesario y elaborar una respuesta al ciudadano. No existía ningún tipo de control de tiempos ni histórico de quejas. Las quejas contestadas eran descartadas e incluso podía darse el caso de que una queja se perdiera al no estar registrada su entrada del sistema o su recorrido. Estas circunstancias justifican sobradamente la necesidad y utilidad de la aplicación desarrollada.

Este documento comenzará con una explicación teórica que tratara diversos puntos, entre los que cabe destacar entre otros:

- Que es la Ingeniería del Software.
- Que es y cómo se desarrolla un producto software.
- Que es un modelo.
- Que es la gestión documental.

La respuesta a estas preguntas describen las actividades fundamentales en el desarrollo de cualquier aplicación informática de gestión y, en concreto, en la implementación del Sistema de Gestión de Quejas y Solicitudes del Ayuntamiento de Torrent.

Y tras esta explicación teórica, se pondrán en práctica para el desarrollo de la aplicación.

2. La Ingeniería del Software

2.1 ¿Qué es la Ingeniería del Software?

Son muchas las definiciones que existen para el término Ingeniería del Software. No obstante ninguna de ellas ha conseguido destacar sobre el resto, siendo aceptada íntegramente por todos los expertos. La ingeniería del software es algo complejo y es una disciplina joven, que evoluciona muy rápidamente y por tanto también lo hace su campo de acción, sus herramientas y, evidentemente, su definición.

La primera definición de Ingeniería del Software fue acuñada en 1968 en la primera conferencia organizada por la OTAN: "Establecimiento y uso de principios de ingeniería para obtener software económico que trabaje de forma eficiente en máquinas reales". Este concepto ha ido evolucionado y adaptándose, por ello es frecuente encontrar diversas definiciones sobre este término que han sido citadas por algunos de los más conocidos y prestigiosos autores del campo de la producción del software. Algunas de estas definiciones son:

- "La Ingeniería del Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software" (IEEE).
- "Es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas software" (Zelkovitz).
- "Se define la Ingeniería del Software como la aplicación práctica del conocimiento científico al diseño y construcción de programas de computador y a la documentación asociada necesaria para desarrollar, operar y mantenerlos. También se conoce como Producción o Desarrollo de Software" (Bohem).
- "Se conoce como Ingeniería del Software a la disciplina que integra métodos, herramientas y procedimientos para el desarrollo de software de computador" (Pressman).
- "La Ingeniería del Software es el establecimiento y uso de principios robustos de la ingeniería con el fin de obtener económicamente software que sea fiable y que funcione sobre máquinas reales" (Bauer).

Como se puede observar, a pesar de sus diferencias, todas las definiciones comparten que la Ingeniería del Software es la disciplina o área de la Ingeniería que ofrece una serie de herramientas, técnicas y métodos para el desarrollo y mantenimiento de productos software.

Cabe destacar que dichas herramientas y técnicas dependerán del método de desarrollo que se esté empleando. Cualquier desarrollo de un producto, ya sea software o de otro tipo, ha de seguir una serie de pasos establecidos. Es decir, tiene un ciclo de vida, que en nuestro caso se compone de cinco etapas principales: análisis, diseño, codificación, testeo y mantenimiento. En el apartado 3 del presente documento se tratará con más detalle este aspecto.

2.2 ¿Qué es un producto Software?

Tras haber analizado en el apartado anterior que es la Ingeniería del Software, la pregunta obligada es la del presente apartado: ¿Qué es un producto Software?

Dicho término fue usado por primera vez por John W. Tukey en 1957.Se denomina producto Software a cualquier producto desarrollado o creado a través del seguimiento de las pautas y normas que establece la Ingeniería del Software. No obstante, al igual que ocurría con la definición de Ingeniería del Software, existen múltiples definiciones para producto Software. Pero probablemente la más formal sea la siguiente: *Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación* (IEEE).

Así pues el término producto Software incluye todas las aplicaciones informáticas que permiten al usuario interaccionar con el ordenador. Dependiendo del tipo de tarea a la cual puede estar destinado el producto software es posible realizar una clasificación de los diferentes tipos de productos software existentes. Encontramos:

- **Software de sistema**: Su objetivo es desvincular adecuadamente al usuario y al programador de los detalles del sistema informático en particular que se use, aislándolo especialmente del procesamiento referido a las características internas de: memoria, discos, puertos y dispositivos de comunicaciones,

impresoras, pantallas, teclados, etc. El software de sistema le procura al usuario y al programador adecuadas interfaces de alto nivel, controladores, herramientas y utilidades de apoyo que permiten el mantenimiento del sistema global. Ejemplos de software de este tipo serían:

- Sistemas operativos
- Controladores de dispositivos
- Herramientas de diagnóstico
- Herramientas de Corrección y Optimización
- Servidores

- **Software de programación**: Es el conjunto de herramientas que permiten al programador desarrollar programas informáticos, usando diferentes alternativas y lenguajes de programación, de una manera práctica. Ejemplos de software de este tipo serían:
  - Editores de texto
  - Compiladores
  - Intérpretes
  - Enlazadores
  - Depuradores
  - Entornos de Desarrollo Integrados (IDE): Agrupan las anteriores herramientas, usualmente en un entorno visual, de forma tal que el programador no necesite introducir múltiples comandos para compilar, interpretar, depurar, etc. Habitualmente cuentan con una avanzada interfaz gráfica de usuario (GUI).
- **Software de aplicación**: Es aquel que permite a los usuarios llevar a cabo una o varias tareas específicas, en cualquier campo de actividad susceptible de ser automatizado o asistido, con especial énfasis en los negocios. Ejemplos de software de este tipo serían:
  - Aplicaciones para Control de sistemas y automatización industrial
  - Aplicaciones ofimáticas
  - Software educativo
  - Software empresarial
  - Bases de datos
  - Telecomunicaciones (por ejemplo Internet y toda su estructura lógica)

- Videojuegos
- Software médico
- Software de cálculo numérico y simbólico.
- Software de diseño asistido (CAD)

Este último tipo de software, el software de aplicación, es en el que se incluirá la aplicación a desarrollar por el presente proyecto, ya que se trata de una aplicación destinada a gestionar las quejas o solicitudes de los ciudadanos.

2.3 Características de un Producto Software

Llegados a este punto es fácil deducir que un producto software no es un componente físico, sino que se trata de un componente lógico. Es necesario señalar que cualquier producto software comparte una serie de características, de las que destacaremos las siguientes:

- El software se desarrolla, no se fabrica.
- El software no se estropea, sino que se deteriora debido a los cambios. Un dispositivo físico es susceptible al paso del tiempo, pues presenta un desgaste en sus componentes. Esto no ocurre a uno producto software al ser un componente lógico. No obstante el software es sensible a los cambios que se produzcan en el sistema y será necesario reajustarlo cuando se produzcan cambio en los elementos que consume la aplicación (cambios de ruta en ficheros, cambios de versiones de servidor…). En la figura 1 podemos ver una diferencia entre la tasa de fallos de un producto software y un producto hardware genérico.

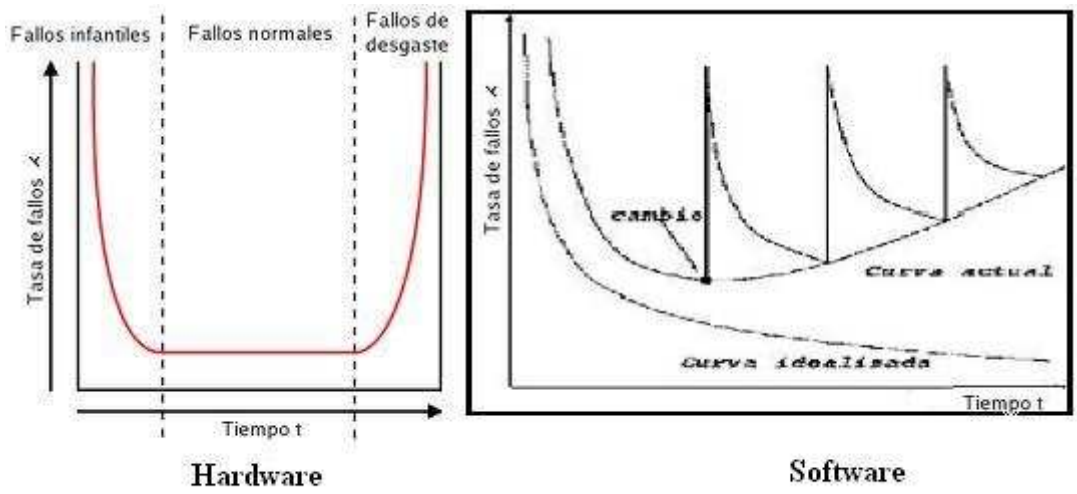


Figura 1: Gráfica Tasa de Fallos

- A pesar de que la tendencia de la industria es hacia la construcción por componentes, es conveniente remarcar que la gran mayoría del Software se construye a medida.

2.4 Factores de calidad de un Producto Software

Por último será necesario establecer una serie de criterios para comprobar la calidad de nuestros productos software. Pero, ¿Qué es la calidad software? Nuevamente nos encontramos con una multitud de definiciones en las que destacaremos:

- La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuarios. (IEEE)
- Concordancia del software producido con los requerimientos explícitamente establecidos , con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente, que desea el usuario. (Pressman)

Así pues será necesario contar con una serie de requerimientos previos especificados por el usuario. Este punto será tratado con más detalle en apartados posteriores.

Además se observa que nuestro producto software deberá cumplir una serie de requisitos o factores independiente de las necesidades del usuario. Podemos clasificar dichos factores en 3 grandes grupos:

1. **Características operativas**
    a. **Corrección** *(¿Hace lo que se le pide?)*: El grado en que una aplicación satisface sus especificaciones y consigue los objetivos encomendados por el cliente.
    b. **Fiabilidad** (*¿Lo hace de forma fiable todo el tiempo?)*: El grado que se puede esperar de una aplicación lleve a cabo las operaciones especificadas y con la precisión requerida.
    c. **Eficiencia** *(¿Qué recursos hardware y software necesito?)*: La cantidad de recursos hardware y software que necesita una aplicación para realizar las operaciones con los tiempos de respuesta adecuados.
    d. **Integridad** *(¿Puedo controlar su uso?)*: El grado con que puede controlarse el acceso al software o a los datos a personal no autorizado
    e. **Facilidad de uso** *(¿Es fácil y cómodo de manejar?)*: El esfuerzo requerido para aprender el manejo de una aplicación, trabajar con ella, introducir datos y conseguir resultados.
2. **Capacidad para soportar los cambios**
    a. **Facilidad de mantenimiento** *(¿Puedo localizar los fallos?)*: El esfuerzo requerido para localizar y reparar errores.
    b. **Flexibilidad** *(¿Puedo añadir nuevas opciones?)*: El esfuerzo requerido para modificar una aplicación en funcionamiento.
    c. **Facilidad de prueba** *(¿Puedo probar todas las opciones?)*: El esfuerzo requerido para probar una aplicación de forma que cumpla con lo especificado en los requisitos.
3. **Adaptabilidad a diferentes entornos**
    a. **Portabilidad** *(¿Podré usarlo en otra máquina?)*: El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo.
    b. **Reusabilidad** *(¿Podré utilizar alguna parte del software en otra aplicación?)*: Grado en que partes de una aplicación pueden utilizarse en otras aplicaciones.

c. **Interoperabilidad** (*¿Podrá comunicarse con otras aplicaciones o sistemas informáticos?)*: El esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas informáticos.

3. Ciclo de Vida Software

El desarrollo de cualquier producto sigue una serie de etapas y un producto software no es una excepción. Definiremos el ciclo de vida software como:

> *Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso (ISO 12207-1).*

Dicho marco de referencia tiene cinco etapas básicas: análisis, diseño, codificación, testeo y mantenimiento. Hay que destacar que el trabajo a realizar en cada una de las etapas estará íntimamente ligado al tipo de ciclo de vida elegido. A continuación se dará una pequeña descripción de lo que se espera de cada etapa, sin condicionar a ningún tipo en concreto.

3.1 Etapas del Ciclo de Vida Software

3.1.1 Análisis

Para desarrollar un producto software es necesario determinar qué elementos intervendrán en el sistema, como será su estructura, cuál será su funcionalidad, con que productos habrá de relacionarse, que características debe cumplir, que grado de seguridad se requiere, con qué datos trabajará, como estarán representados dichos datos…. Es decir, será necesario mediante multitud de entrevistas con el cliente, extraerle toda la información y entender que es lo que realmente quiere y/o necesita. En esta parte entraría en juego la Ingeniería de Requisitos, que será tratada con más detalle en apartados posteriores. Así mismo será necesario documentar las conclusiones extraídas, pues deberán ser consultadas por los analistas, diseñadores y programadores y deberá asegurarse su cumplimiento en la fase de testeo.

Esta fase es quizá la más importante del todo el proceso de desarrollo, ya que es la base del producto y cualquier error que se comenta en esta fase supondrá errores en las siguientes etapas, lo que puede provocar retrasos y pérdidas económicas.

3.1.2 Diseño

Después del análisis realizado en la etapa anterior se tiene una idea clara de lo que debe hacer dicho sistema. Ahora será necesario determinar cómo debe hacerlo. De esto se encarga la etapa de diseño. En esta etapa se definirá con detalle las estructuras de datos, la estructura del producto, la interfaz grafica (GUI) y los lenguajes de programación.

Así mismo se deberá documentar dichas estructuras de forma estandarizada. Para ello será necesario el uso de diferentes modelos como diagramas de clase, diagramas de flujo, etc.

3.1.3 Codificación

La etapa de codificación o implementación consiste en la programación del producto software. En esta etapa se debe implementar tanto las estructuras de datos como las relaciones existentes entre ellas, las funcionalidades del software y la integración del producto software con los diferentes sistemas con los que ha de comunicarse. El resultado de esta etapa debe ser el producto software finalizado y cumpliendo todas y cada una de las especificaciones obtenidas en la fase de análisis (a falta de comprobar las pruebas que garantizarán el correcto comportamiento de la aplicación).

3.1.4 Testeo

La finalidad de esta etapa es garantizar que el software que ha sido desarrollado funciona correctamente. Existen multitud de formas de testeo y técnicas de verificación y validación, pero quizás la más popular es la que se conoce como Software Testing.

La bibliografía que podemos encontrar sobre el Software Testing es inmensa pero podemos dar unas ideas básicas. En esta técnica de testeo se diseñarán una serie de pruebas, que no serán más que un registro de acciones y datos que habrá que proporcionarle al programa y la respuesta que cabría esperar ante dicha combinación de acciones y datos. Dichas pruebas deberán estar diseñadas con el fin de encontrar errores

y no con el fin de demostrar la corrección. Así pues estableceremos la calidad de la prueba dependiendo de la posibilidad que tiene dicha prueba de encontrar un fallo. Así mismo las pruebas a diseñar pueden ser de dos tipos:

- **Pruebas de caja blanca**: En estas pruebas se toma como base el código y se fundamentan en comprobar todos los posibles caminos que puede tomar una rutina y sus posibles comportamientos ante diferentes entradas.
- **Pruebas de caja negra**: Estas pruebas se basan en simular el comportamiento del usuario, sin atender al código. Se apoyan en la idea de que ante unas entradas determinadas el sistema debe producir una salida determinada independientemente de las rutinas que se ejecuten.

Hay que destacar que es necesario establecer de antemano la cobertura de la prueba. Para sistemas poco críticos, suele ser suficiente con una cobertura del sesenta u ochenta por ciento. Además, sería recomendable que la persona que procesará las pruebas no fuera el programador que escribió el módulo a testear. Por último, es necesario destacar que esta fase suele consumir el mayor porcentaje del ciclo de vida, por lo cual lo más interesante de este método reside en la generación automática de pruebas. Para ello disponemos de tres paradigmas:

- **Generación aleatoria**: se generan entradas aleatoriamente hasta que se encuentra una útil, según un determinado criterio. Cuanto más complejo es el programa, más difícil encontrar un caso útil.
- **Generación de test simbólicos**: estáticamente se asignan valores simbólicos a las variables.
- **Generación dinámica de pruebas**: búsqueda directa a través de la ejecución del programa a probar. Se usan técnicas de búsqueda meta heurísticas: algoritmos genéticos, recorrido simulado o búsqueda tabú.

El Software Testing es la técnica que se ha utilizado para probar la aplicación desarrollada en el presente proyecto. No obstante, como ya se ha comentado, esta técnica de testeo no es única, pues existen múltiples alternativas como podrían ser la verificación por métodos formales, el Program Slice, ModelCheking, etc.

Si con el resultado de dichas pruebas se encuentran fallos, es necesario volver a etapas anteriores para corregirlos. En cambio, de pasar las pruebas satisfactoriamente, el producto software estará preparado para entregar al cliente.

3.1.5 Mantenimiento

Como se mencionó en apartados anteriores el software no se deteriora con el tiempo. No obstante es necesario realizar un mantenimiento del mismo. Esto es debido a que se pueden producir cambios en el sistema y será necesario reajustar el software para que se adecue a la nueva configuración. Así mismo pueden aparecer errores no contemplados en las pruebas o debido a un mal uso del producto software. También es posible que el software funcione perfectamente y sean los usuarios del mismo los que necesiten algún tipo de formación o ayuda complementaria en algún momento dado.

Por todo ello es necesario realizar una etapa de mantenimiento que se encargará de dar soporte a los usuarios, así como corregir errores imprevistos y atender las peticiones de los usuarios, por si fuera necesario añadir nuevas funcionalidades y ampliar el producto software, desarrollando una versión nueva.

3.2 Tipos de Ciclo de Vida Software

Existen diferentes tipos de Ciclos de Vida Software que pueden ser utilizados en la elaboración de productos software. Algunos de estos ciclos de vida son:

3.2.1 Ciclo de vida Clásico o en Cascada

Este enfoque del desarrollo de software ha sido el más utilizado y en la actualidad, pese a la aparición de metodologías ágiles, sigue siendo la solución predominante, si bien, en cada organización o cada director de proyecto la puede llevar a cabo con ciertas variantes.

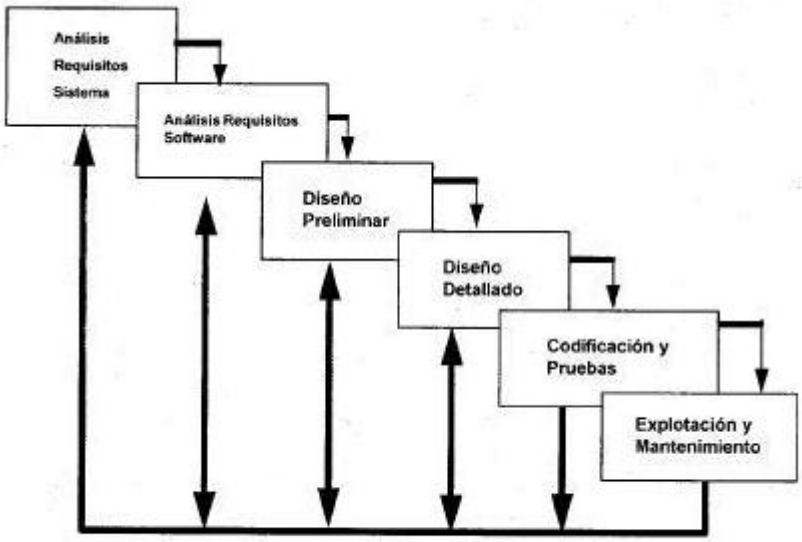


Figura 2. Modelo en Cascada

Este modelo también es conocido como modelo lineal o modelo secuencial, dado que se sigue un enfoque secuencial para el desarrollo del producto software. En este ciclo empezaríamos con la fase de análisis e iríamos pasando linealmente a la siguiente fase. Este método admite la posibilidad de iteraciones. Si en alguna fase se detecta algún error o una mala especificación o ambigüedad se puede volver a una fase anterior para corregir el fallo.

A pesar de ser el ciclo de vida más utilizado presenta una serie de inconvenientes como son:

- En la vida real, los proyectos raras veces siguen la secuencia que se ofrece en el modelo. Aunque este modelo puede acoplar interacción, lo hace indirectamente. Como resultado, los cambios pueden causar confusión cuando el equipo del proyecto ya ha comenzado.
- A menudo es difícil que el cliente exponga explícitamente todos los requisitos. Este modelo requiere disponer de todos los requisitos al inicio del proyecto, por lo que puede tener dificultades a la hora de incluir nuevos requisitos.
- Los usuarios tardan demasiado en ver los resultados, lo que hace que el tiempo transcurrido desde que se define el sistema hasta que está disponible sea lo suficientemente amplio como para que hayan ocurrido muchas cambios: desde que no estén la mayoría de las personas que participaron en la especificación, como cambios en los procesos, cambios de criterio, etc. Esto provoca el hacer un

replanteamiento de los requisitos en etapas más tardías del desarrollo, introduciendo un importante coste adicional. Todo ello puede llevar a que el producto final esté muy alejado de lo que realmente quiere el conjunto de usuarios en estos momentos, que puede ser muy distinto a lo que querían en el inicio del proyecto.

- Cualquier error de diseño detectado en la fase de prueba conduce al rediseño y una nueva programación cosa que aumenta los costes del desarrollo.

No todo es malo en este modelo, ya que describe un procedimiento racional y ordenado de desarrollo de software, la clave para su éxito o su fracaso es como se gobierne el mismo y las circunstancias que rodeen al proyecto en el momento de su ejecución. Además, existen variantes como el ciclo de vida con prototipado, explicado en el siguiente punto, que ofrecen una mayor flexibilidad al mismo y que permite reducir sus riesgos.

3.2.2 Ciclo de vida Clásico con Prototipado

Este tipo de ciclo de vida ofrece una alternativa de enfoque para el ciclo de vida clásico. En este caso, la fase de análisis consiste en capturar un conjunto inicial de requerimientos y necesidades e implementarlas rápidamente con la intención declarada de expandirlas y refinarla iterativamente al ir aumentando la compresión del sistema tienen los usuarios y quien lo desarrolla. Este prototipo puede servir como primer sistema, aunque lo recomendable una vez se ha conseguido captar lo que el cliente realmente desea, es comenzar de nuevo con la creación de software.

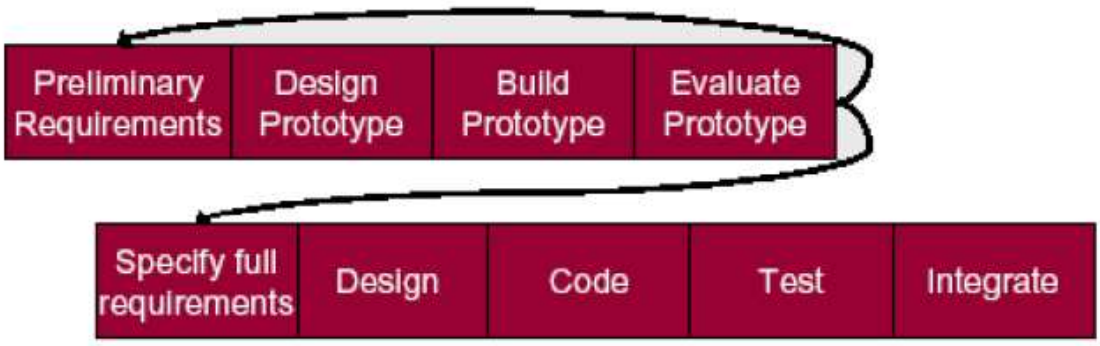


Figura 3. Ciclo de Vida con Prototipado

Las principales ventajas que ofrece este modelo es que facilita la comunicación entre el cliente y el desarrollador, reduciendo el riesgo de construir productos que no satisfagan las necesidades de los usuarios. Con ello conseguimos reducir los costes y aumentamos la probabilidad de éxito de nuestro proyecto.

A pesar de ofrecer una serie de ventajas respecto al modelo anterior, también presenta ciertos inconvenientes:

- El cliente se puede crear unas expectativas cuando ve el prototipo de cara al sistema final, o pensar que el producto software esta cerca de ser finalizado. Esto puede ser contraproducente, ya que con la intención de crear un prototipo de forma rápida, se suelen dejar de lado aspectos esenciales en la elaboración del software, lo cual obliga a que cuando se han reevaluado los requisitos con el usuario se debe volver a reconstruir desde el principio.
- Otro inconveniente es que con la intención de ofrecer un prototipo rápidamente, no se toman las decisiones adecuadas como estructuras del sistema de ficheros, diseño de la base de datos o la elección del lenguaje de programación. Estas decisiones deben ser reevaluadas al reconstruir el producto desde el principio.

3.2.3 Modelos Evolutivos

Estos modelos surgen como consecuencia de que el software evoluciona con el tiempo, que los requisitos del usuario y del producto pueden cambiar mientras se está desarrollando el mismo. Por ello se diseñan los modelos evolutivos, que permiten a los desarrolladores disponer de una evolución temporal o progresiva en la elaboración del software. En estos modelos se debe conocer desde el comienzo los requisitos centrales del producto, pero no es necesario que estén definidos con todo lujo de detalles, ya que gracias a la posibilidad de ir evolucionando el software, podrán ir adaptándolo a los requisitos más específicos.

En resumen, son modelos adaptables a requisitos cambiantes e iterativos, que permiten realizar versiones cada vez más complejas y completas hasta llegar al producto final, solicitado por el cliente. Entre los modelos evolutivos destacan el modelo incremental y el modelo en espiral.

3.2.3.1 Modelo Incremental

El Modelo Incremental combina elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos.

En una visión genérica, el proceso se divide en las cinco partes básicas: Análisis, Diseño, Codificación, Testeo y Mantenimiento. Sin embargo, para la producción del Software, se usa el principio de trabajo en cadena o "Pipeline", utilizado en muchas otras formas de programación. Con esto se mantiene al cliente en constante contacto con los resultados obtenidos en cada incremento.

Es el mismo cliente el que incluye o desecha elementos al final de cada incremento a fin de que el software se adapte mejor a sus necesidades reales. El proceso se repite hasta que se elabore el producto completo. De esta forma el tiempo de entrega se reduce considerablemente. Al igual que los otros métodos de modelado, el Modelo Incremental es de naturaleza interactiva pero se diferencia de aquellos en que al final de cada incremento se entrega un producto completamente operacional.

El Modelo Incremental es particularmente útil cuando no se cuenta con una dotación de personal suficiente. Los primeros pasos los pueden realizar un grupo reducido de personas y en cada incremento se puede añadir personal, de ser necesario. Por otro lado los incrementos se pueden planear para gestionar riesgos técnicos.

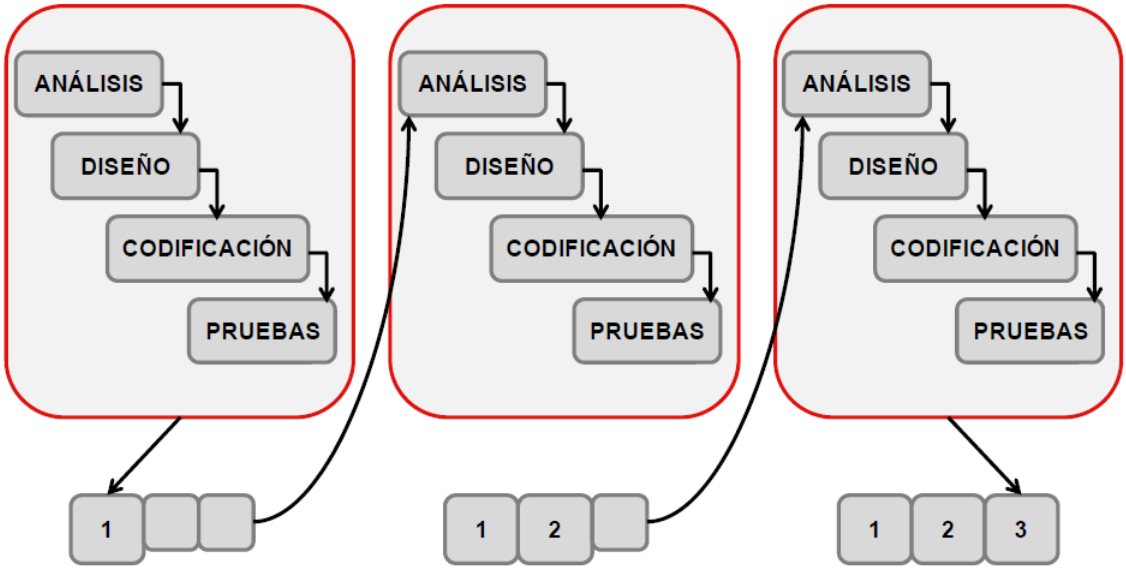


Figura 4. Modelo Incremental

Entre las ventajas que puede proporcionar un modelo de este tipo encontramos las siguientes:

- Mediante este modelo se genera software operativo de forma rápida y en etapas tempranas del ciclo de vida del software.
- Es un modelo más flexible, por lo que se reduce el coste en el cambio de alcance y requisitos.
- Es más fácil probar y depurar en una iteración más pequeña.
- Es más fácil gestionar riesgos.
- Cada iteración es un hito gestionado fácilmente
- El usuario se involucra más.

No obstante este modelo también presenta una serie de desventajas:

- Cada fase de una iteración es rígida y no se superponen con otras.
- Requiere de un cliente involucrado durante todo el curso del proyecto. Hay clientes que simplemente no estarán dispuestos a invertir el tiempo necesario.
- Requiere de mucha planeación, tanto administrativa como técnica.

Este es el tipo de ciclo de vida elegido para el desarrollo del presente proyecto.

3.2.3.2 Modelo en Espiral

El desarrollo en espiral es un modelo de ciclo de vida desarrollado por Barry Boehm en 1985, utilizado de forma generalizada en la ingeniería del software. Las actividades de este modelo se conforman en una espiral, en la que cada bucle representa un conjunto de actividades. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgos, comenzando por el bucle anterior.

Boehm, ideó y promulgó este modelo desde un enfoque distinto al tradicional en Cascada. Su modelo de ciclo de vida en espiral tiene en cuenta fuertemente el riesgo que aparece a la hora de desarrollar software. Para ello, se comienza mirando las posibles alternativas de desarrollo, se opta por la de riesgos más asumibles y se hace un ciclo en la espiral. Si el cliente quiere seguir haciendo mejoras en el software, se vuelven a evaluar las nuevas alternativas y riesgos y se realiza otra vuelta en la espiral, así hasta que llegue un momento en el que el producto software desarrollado sea aceptado y no necesite seguir mejorándose con otro nuevo ciclo.

Este modelo de desarrollo combina las características del modelo de prototipos y el modelo en cascada. El modelo en espiral está pensado para proyectos largos, caros y complicados. Esto modelo no fue el primero en tratar el desarrollo iterativo, pero fue el primer modelo en explicar las iteraciones.

Se suele interpretar como que dentro de cada ciclo de la espiral se sigue un modelo en cascada, pero no necesariamente ha de ser así. De normal existen entre tres y seis fases, en la figura se muestran seis fases, que son las que se emplean en el modelo en espiral típico.

El trabajo que se realiza en cada una de estas fases es:

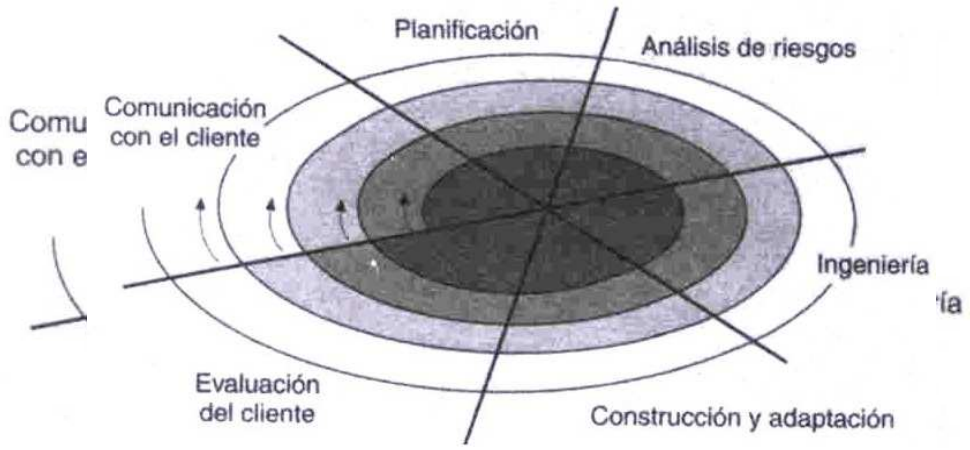


Figura 5. Modelo en espiral

Figura 5. Modelo en espiral

1. **Comunicación con el cliente**: Se centra en las tareas necesarias para el establecimiento de la comunicación entre el desarrollador y el cliente/usuario final.
2. **Planificación**: Contiene las tareas de definir información relacionada con el proyecto, como puede ser los recursos empleados para la elaboración del mismo y el tiempo necesario para ello.
3. **Análisis de riesgos**: Esta fase se centra en las tareas utilizadas para la una evaluación de los riegos técnicos y de gestión del proyecto en desarrollo.
4. **Ingeniería**: En esta fase se realizan las tareas requeridas para la construcción de una o más representaciones de la aplicación.
5. **Construcción y adaptación**: Contiene las tareas necesarias para construir, probar, instalar y proporcionar soporte al usuario.
6. **Evaluación del cliente**: Esta fase contiene las tareas requeridas para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante la fase de ingeniería e implementada durante la fase de instalación.

El análisis de riesgos se hace de forma explícita y clara. Entre las ventajas que presenta este modelo encontramos:

- Reduce riesgos del proyecto, ya que en caso de producirse un fallo grave, se puede detectar al final de la iteración, con lo que solo se han perdido los recursos y el tiempo invertidos en esa iteración.

- Incorpora objetivos de calidad
- Integra el desarrollo con el mantenimiento

Además es posible tener en cuenta mejoras y nuevos requerimientos sin romper con el modelo, ya que el ciclo de vida no es rígido ni estático. Mediante este modelo se produce software en etapas tempranas del ciclo de vida y suele ser adecuado para proyectos largos de misión crítica.

No obstante, este modelo también presenta una serie de inconvenientes:

- Al igual que en el modelo anterior, necesita la participación continua del cliente. Lo cual en ocasiones es algo complicado ya que no es fácil implicar al cliente con el desarrollo del producto.
- Modelo costoso.
- Requiere experiencia en la identificación de riesgos
- Incertidumbre en el número de iteraciones que serán necesarias.

3.2.3.3 Programación Extrema (XP)

La programación extrema (XP) es un enfoque de la ingeniería del software formulado por Kent Beck. Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto y aplicarlo de manera dinámica durante el ciclo de vida del software.

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

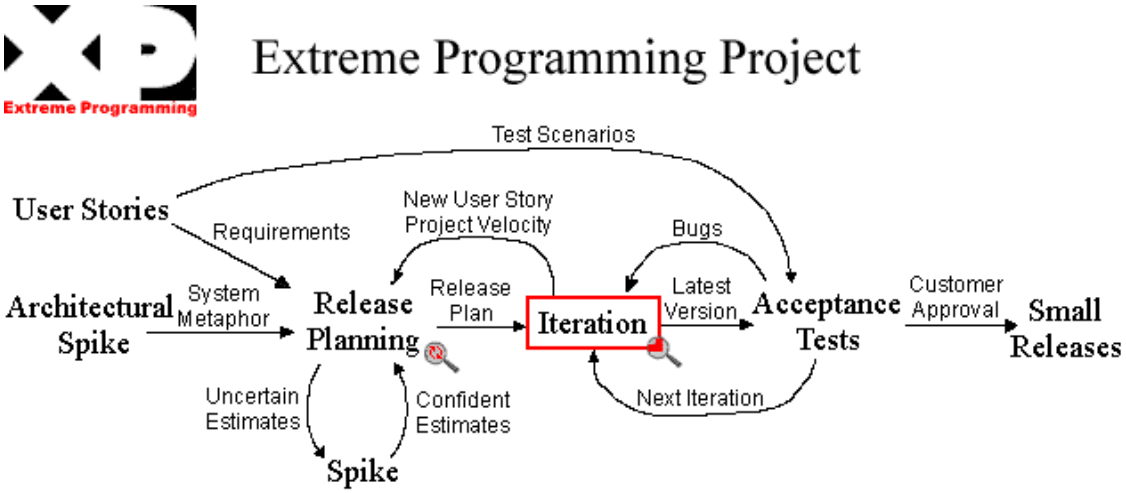


Figura 6. Extreme Programming

Las principales características de esta metodología son:

- **Desarrollo iterativo e incremental**: Como se ha dicho, Extreme Programming se basa en una metodología iterativa que desarrolla una versión o incremento del software en cada iteración.
- **Pruebas unitarias continuas**: Gracias a que se realizan versiones del producto cada poco tiempo y en el desarrollo de cada versión se ha seguido los pasos del ciclo de vida clásico, las pruebas se realizan también cada poco tiempo, con lo que es más fácil detectar y/o corregir los posibles errores cometidos.
- **Simplicidad**: En Extreme Programming se desea simplicidad en el código. Apuesta por el diseño sencillo y que el coste de modificar parte del código sea pequeño, frente a realizar un diseño complejo y que cualquier modificación sea difícil de gestionar.

- **Propiedad de código compartido**: Extreme Programming prefiere que el código sea compartido, es decir, que todo el personal tenga acceso al código y así pueda revisar, corregir y extender cualquier parte del código. Con ello se consigue mejorar el código y también aumenta la posibilidad de detectar errores.
- **Refactorización del código**: Esta tarea consiste en revisar el código y reescribir alguna parte, si es preciso, para aumentar su legibilidad y mantenibilidad. Para ello hay que tener cuidado y verificar que tras los cambios el código se comporta de la misma manera que antes. Con este mecanismo también se consigue eliminar zonas de código que realizan la misma función en distintas ocasiones y puede ser posible crear un único método que se encargue de esa labor, lo cual mejora tanto la legibilidad como la calidad del código.
- **Verificación de errores**: Antes de añadir nuevas funcionalidades verificar que no existen errores en el código.
- **Integración del cliente con el equipo de desarrollo**: Con esto se facilita y agiliza la entrega de las diferentes versiones que se van desarrollando y así la evaluación por parte del cliente se puede tener en cuenta a la hora de corregir o mejorar alguna funcionalidad del producto. En esta metodología es muy importante la opinión e intervención del usuario en el desarrollo.
- **Programación en parejas**: Extreme Programming propone un desarrollo de cada tarea por parejas de programadores. Aunque parezca que esto aumenta el coste del proyecto ya que se tienen dos personas trabajando sobre una misma tarea, se ha demostrado que los resultados son mejores ya que mientras uno de ellos está programando el otro puede estar revisando, probando o discutiendo la manera de mejorar el trabajo que se realiza.

4. Ingeniería de requisitos

4.1 ¿Qué es la Ingeniería de requisitos?

Al igual que ocurría con la Ingeniería del software, existen multitud de definiciones para la Ingeniería de requisitos (IDR). Entre las que encontramos:

- La IDR es el conjunto de actividades que incluyen la búsqueda o aprendizaje sobre el problema que necesita una solución y la especificación del comportamiento externo de un sistema que puede resolver dicho problema. El producto final de la IDR es la especificación de requisitos software. (*Davis*)
- La IDR es la rama de la Ingeniería del Software relacionada con los objetivos, servicios y restricciones de los sistemas software. (*IEEE*)

A pesar de sus diferencias, podemos extraer, al menos, que el objetivo de la Ingeniería de Requisitos es: "*Identificar, analizar, definir y especificar de los requisitos del sistema.*"

Pero, ¿Qué es un requisito? Seguin Std.610.12-1990, IEEE Standard Glossary of Software Engineering Terminology:

1. Una condición o capacidad necesaria para que un usuario resuelva un problema o alcance un objetivo.
2. Una condición o capacidad que debe reunir o poseer un sistema o un componente de un sistema para satisfacer un contrato, un estándar, una especificación u otros documentos impuestos formalmente.
3. Una **representación documentada** de una condición o capacidad como en (1) o (2).

Quizá la parte más importante sea la documentación. Un requisito no documentado es un requisito no capturado, un requisito que no podremos transmitir a los diseñadores, programadores y testers, es decir, un requisito que no estará integrado en el desarrollo del producto software y que por tanto provocará discrepancias entre lo buscado por el cliente y el producto ofrecido. Lo que podría provocar retrasos y/o pérdidas económicas.

4.2 Dominio y técnicas

Como se ha podido comprobar en el apartado anterior, a pesar de las diferencias entre los distintos tipos de ciclos de vida, todos comparten una fase de análisis. Aquí es donde entra en juego la Ingeniería de requisitos que implica todas las actividades de la fase de análisis del ciclo de vida dedicadas a:

- La educción (a veces llamada "elicitación", debido a una mala traducción del inglés "elicitation") de los requisitos de usuario. A través de entrevistas o comunicación con clientes o usuarios, para saber cuáles son sus expectativas.
- El análisis y negociación de requisitos para derivar requisitos adicionales. Detectando y corrigiendo, así mismo, las falencias comunicativas, transformando los requisitos obtenidos de entrevistas y requisitos, en condiciones apropiadas para ser tratados en el diseño.
- La documentación de los requisitos.
- Verificar los requisitos, comprobando el correcto funcionamiento de un requisito en la aplicación. (Esta actividad debe, evidentemente, hacerse tras la fase de Implementación, pudiendo provocar nuevas iteraciones en los métodos).
- La validación de los requisitos documentados contra las necesidades de usuario. Comprobando que los requisitos implementados se corresponden con lo que inicialmente se pretendía.
- Así como los procesos que apoyan estas actividades.

La ingeniería de requisitos puede ser un proceso largo y arduo para el que se requiere de habilidades psicológicas. Cabe destacar que esta búsqueda de requisitos no está acotada únicamente al cliente, es importante identificar a todos los stakeholders (cualquier persona o sistema que vaya a interactuar con la aplicación a desarrollar), considerar sus necesidades y asegurar que entienden las implicaciones de los nuevos sistemas. Los analistas pueden emplear varias técnicas para obtener los requisitos del cliente. Históricamente, esto ha incluido técnicas tales como las entrevistas, o talleres con grupos para crear listas de requisitos. Técnicas más modernas incluyen brainstorming, construcción de prototipos, etc. Cuando sea necesario, el analista empleará una combinación de estos métodos para establecer los requisitos exactos de las personas implicadas, para producir un sistema que resuelva las necesidades del negocio.

Sería interesante ver una pequeña reseña de estas técnicas y las fuentes de información que se emplea en cada una:

- **Entrevistas**: Las entrevistas son un método común. Por lo general no se entrevista a toda la gente que se relacionará con el sistema, sino a una selección de personas que represente a todos los sectores críticos de la organización, con el énfasis puesto en los sectores más afectados o que harán un uso más frecuente del nuevo sistema. Es conveniente empezar por preguntas de contexto libre, para entender el problema, contexto, necesidades, motivaciones, etc. No obstante las entrevistas presentan una serie de problemas como pueden ser malentendidos, omisión de información, mala relación de trabajo ("nosotros-ellos")…
- **Encuestas/Cuestionarios:** Las encuestas pretenden corregir las carencias de las entrevistas, estandarizando las preguntas que se le hace al usuario. Adicionalmente haciéndolas de forma anónima se consigue eliminar las malas relaciones interpersonales. No obstante el hecho de eliminar este factor humano también posibilita el perder requisitos no imaginados a la hora de construir la encuesta y que el encuestado no tiene modo de transmitir. Del mismo modo es posible que sea necesario contextualizar algunas de las respuestas para comprender realmente lo que el cliente pretende transmitir.
- **Introspección**: El analista estudia el dominio del problema y posteriormente "imagina" las características del sistema a desarrollar. Este modo de educción presenta la ventaja que es muy sencillo de realizar y es muy económico. No obstante es muy impreciso y difícilmente reflejara correctamente las necesidades de los usuarios.
- **Análisis de documentos:** Parecida a la introspección, el analista estudia el dominio del problema y posteriormente recoge documentación utilizada en la organización: formularios, facturas, contratos, información financiera, información de mercado, manuales de las aplicaciones actuales…
- **JAD (Joint Application Development):** Alternativa a la entrevista. Una técnica de grupo que requiere la participación de todos los stakeholders: analistas, diseñadores, usuarios, clientes, etc. Representa un acuerdo entre clientes y desarrolladores y minimiza los cambios posteriores en los requisitos. Se apoya en cuatro principios básicos: dinámica de grupo, uso de

técnicas de visualización para mejorar la comunicación, proceso organizado y racional, documentación del tipo WYSIWYG. En JAD se realizan cuatro actividades básicas:

- *Definición del proyecto*: El coordinador se entrevista con gerentes y clientes para determinar objetivos y alcance del proyecto, creando la "guía de definición administrativa".
- *Investigación*: entrevista con usuarios y recopilación de información del dominio, descripción de flujos de trabajo y asuntos a tratar en la reunión. Se crea la "agenda de sesión" y la "especificación preliminar".
- *Preparación*: El coordinador crea un "documento de trabajo" o primer borrador del documento final.
- *Sesión*: El coordinador guía al equipo para crear la especificación del sistema en una reunión que puede durar varios días. Se definen los flujos del trabajo, elementos de datos, pantallas, informes… Las decisiones se documentan en unos formularios.
- *Documento Final*: El coordinador prepara el "documento final" usando los "formularios" u se distribuye a los asistentes para su revisión. Se hace una reunión para discutir revisiones y finalizar el documento.

Figura 7. JAD

- **Brainstorming:** Es una técnica de grupo similar al JAD. Consiste en recoger ideas e información de todos los stakeholders. Un participante asume el rol de "moderador" pero NO debe controlar la sesión, se debe crear un ambiente para estimular ideas y evitar las críticas. Tiene 3 fases principales:
    - *Preparación*: determinar objetivos, componentes y organización.
    - *Generación*: tantas ideas como sean posibles orientadas al objetivo.
    - *Consolidación*: organización y revisión de las ideas resultantes.

Como se puede observar, la Ingeniería de requisitos dispone de un gran número de técnicas para obtener los requisitos. La elección de una u otra dependerá del sistema a desarrollar y del contexto de desarrollo. Para la realización del presente proyecto se realizaron diversas reuniones de brainstorming, en el que tanto el cliente como los desarrolladores ofrecían ideas sobre la nueva funcionalidad a desarrollar en la siguiente iteración.

4.3 Tipos de requisitos

Así mismo los requisitos capturados por la Ingeniería de requisitos pueden ser de tres tipos:

- **Requisitos del negocio**: describe el propósito de alto nivel y las necesidades del producto para aumentar ganancias, reducir gastos,…
- **Requisitos de usuario**: describe las tareas que los usuarios necesitan realizar con el software.
- **Requisitos del software**: son descripciones de las necesidades que el software debe realizar para satisfacer los requisitos de usuario y del negocio.

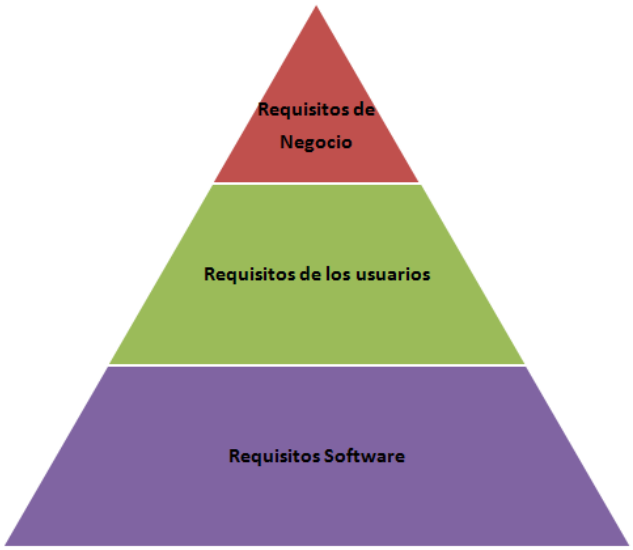


Figura 8. Tipos de Requisitos

Este último tipo, se divide a su vez en dos tipos atendiendo a la naturaliza del requisito encontrando:

- **Requisitos funcionales:** Describen la funcionalidad o los servicios que se espera que el sistema proveerá, dividiéndose a su vez en:
    - *Requisitos de Interfaz:* En este apartado aparecen reflejados todos los requisitos que definen la forma de enviar la información a procesar por los usuarios al sistema, y la forma de recibir la respuesta del sistema por el usuario. Son requisitos de Interfaz aquellos referentes a informes, medios de interacción, pantallas, formularios, libros de estilo, …
    - *Requisitos de procesamiento:* En este apartado incluiremos todos los requisitos que permiten realizar las principales funciones del sistema. Son aquellos requisitos que indican qué hacer con los datos de entrada, cómo procesarlos y generar datos de salida. Indican los requisitos que hay que aplicar a las funciones y procesos internos. Hay que definir qué datos hay que tratar y mediante qué procesos se van a tratar. Normalmente para una aplicación de gestión se recogen los requisitos que definen la lógica de negocio.
    - *Requisitos de persistencia:* En este apartado se recogen los requisitos que afectan a la información que se debe persistir en el sistema, es decir la información que se debe guardar entre diferentes ejecuciones del sistema.
    - *Requisitos de gestión:* Estos requisitos recogen todas las funciones que son necesarias para gestionar el sistema, por ejemplo la gestión de usuarios, gestión de la configuración del sistema y otras funciones del sistema que se apartan de la función principal del sistema.
- **Requisitos no funcionales**: Se refieren a las propiedades emergentes del sistema, es decir, todos los requisitos del sistema que no representan la funcionalidad principal del sistema, sino que fijan condiciones para realizar dicha funcionalidad. Entre estos requisitos encontramos:
    - *Requisitos de disponibilidad*: En este apartado se incluyen los requisitos que definen la disponibilidad del sistema, es decir, el tiempo que debe estar operativo, así como el comportamiento del sistema en caso de fallos. Entre ellos podemos enumerar: tiempo

medio entre fallos, tolerancia a fallos en el sistema o en su acceso, tolerancia a fallos de su base de datos, operativas que deben estar disponibles en caso de fallos de alguna de las partes del sistema, tiempos de respuesta, etc.

- *Requisitos de Almacenamiento*: Aquí se recogen todos los requisitos que especifican el cómo, dónde y cuándo guardar los datos persistentes del sistema, así como la capacidad del sistema de almacenamiento de los mismos, su seguridad, su fiabilidad, su protección contra fallos o intento de acceso no autorizado y su política de respaldo.
- *Requisitos de Seguridad*: Aquí se recogen todos los requisitos relativos a la seguridad del sistema, como pueden ser: control de acceso al sistema y autenticación de usuarios, políticas de usuarios y contraseñas, control y auditoría de las acciones de los usuarios, métodos de agrupación de usuarios y de permisos, esquemas de administración y almacenamiento de la seguridad, gestión de los roles de los usuarios, medidas de protección del sistema frente a ataques externos, etc.
- *Requisitos de Escalabilidad*: Aquí se recogen los requisitos de capacidad del sistema y cómo se debe poder ampliar si es necesario. Si el sistema es utilizado por múltiples usuarios simultáneos, debe disponer de un plan para redimensionar el sistema al crecer el número de usuario.

4.4 Documentación

Por último es necesario destacar que todo el tiempo invertido en la obtención de requisitos mediante el uso de cualquier técnica, así como su clasificación, sería inútil si dichos requisitos no se documentaran. Además, dicha documentación debe hacerse de forma estandarizada para que pueda ser aprovechada por todos los profesionales que participarán en el desarrollo de la aplicación. Uno de los estándares más populares actualmente para la documentación de requisitos quizás sean los documentos "Visión" de UML. La estructura de dichos documentos queda fuera del presente proyecto, pero UML posee una gran bibliografía al respecto.

5. Gestión documental

5.1 ¿Qué es la Gestión documental?

Se entiende por gestión documental el conjunto normas técnicas y prácticas usadas para administrar el flujo de documentos de todo tipo en una organización, permitir la recuperación de información desde ellos, determinar el tiempo que los documentos deben guardarse, eliminar los que ya no sirven y asegurar la conservación indefinida de los documentos más valiosos, aplicando principios de racionalización y economía.

En la actualidad, coexisten en el mundo los más diversos sistemas de gestión documental: desde el simple registro manual de la correspondencia que entra y sale, hasta los más sofisticados sistemas informáticos que manejan no sólo la documentación administrativa, sino que además controlan los flujos de trabajo del proceso de tramitación de los expedientes, capturan información desde bases de datos de producción, contabilidad y otros, enlazan con el contenido de archivos, bibliotecas, centros de documentación y permiten realizar búsquedas sofisticadas y recuperar información de cualquier lugar.

La gestión documental es una disciplina bastante extensa, pero en nuestro contexto haciendo una simplificación extrema, puede quedar relegada a la implantación y configuración de un servidor de base de datos. No obstante es necesario destacar que cualquier producto software debe trabajar con un conjunto de datos, y no solo es necesario en la fase de análisis capturar los requisitos para usar y proteger esos datos. Sino que es necesario establecer como habrán de mantenerse esos datos, cuando pueden considerarse obsoletos, cuando pueden ser descartados, cuantas copian han de tenerse de los datos, como se han de manejar esas copias, etc. Como se puede observar estos aspectos transcienden de la simple programación y están mas relacionados con las tareas administrativas que se hayan de llevar a cabo.

6. Modelos

6.1 ¿Qué es un modelo?

Para entender que es un modelo en Ingeniería del Software es necesario hacer un símil con alguna otra ingeniería. Imaginemos pues un arquitecto que desea construir un edificio. La primera tarea que nos viene a la mente sea quizás, la construcción de un plano. En este plano se hará una representación de cómo será el edificio y todas las decisiones adicionales se harán tomando este plano como punto de partida.

Adicionalmente todas las personas involucradas en el proceso de creación del edificio son capaces de interpretar, en mayor o menor medida, dicho plano, ya que para el diseño de los planos se utiliza un lenguaje estandarizado. Si cuando se proponen construir un edificio no se dispusiera de planos, tal vez sería posible construirlo, pero sería más complicada la labor y además también sería más complicado satisfacer los deseos de los compradores. El uso de planos facilita la comprensión e incluso la toma de decisiones, lo cual facilita en gran medida el desarrollo de la construcción y también aumenta las posibilidades de que el cliente obtenga lo que deseaba.

Si ahora nos centramos en el desarrollo de software, es lógico pensar que si se utiliza algo similar a estos planos, sería posible facilitar la labor de los desarrolladores. Es por ello que se decidió incorporar al desarrollo de software unos planos, llamados modelos, cuya labor es similar a la que realiza un plano en la construcción de un edificio.

Así pues los modelos son representaciones de la realidad, ya que proporcionan planos del sistema que se debe crear. Estos planos pueden ser planos generales que ofrecen una visión global del producto o pueden ser planos más detallados que traten sobre aspectos más concretos del producto final. En un buen modelo deben aparecer los elementos que tienen más relevancia y evitar aquellos que no lo son. Cada sistema puede ser descrito desde diferentes puntos de vista utilizando diferentes modelos. Los modelos pueden ser estructurales, es decir que destacan la organización del sistema, o modelos de comportamiento que tratan de la semántica del sistema.

6.2 ¿Por qué usar modelos?

Como se ha comentado en el apartado anterior un modelo es una representación de la realidad. Concretamente de la parte de la realidad queremos estudiar. Gracias a esta representación podemos comprender mejor el sistema que se ha de desarrollar, obteniendo numerosas ventajas:

- Nos ayuda a visualizar como es o esperamos que sea el sistema final.
- Nos permite especificar la estructura o comportamiento del sistema.
- Nos proporciona plantillas que nos guiarán en la construcción del sistema.
- Nos ayuda a documentar las decisiones que hemos tomado.
- Facilita la comunicación de forma no ambigua: si se utilizan modelos, al tener una notación estandarizada y tipificada, el intercambio de información es más fiable que si se hace de palabra entre los componentes del equipo, ya que una mala interpretación puede tener graves consecuencias en el desarrollo.
- Nos ofrece una solución general y reutilizable: en el mundo del desarrollo de software siempre pueden aparecer sistemas similares. Para ello, se pueden utilizar modelos de sistemas anteriores y así poder ahorrar tiempo y dinero en el desarrollo de una solución, ya que bastaría con adaptar el modelo a lo que se necesita.

Es importante destacar que el modelado no se utiliza exclusivamente en los grandes proyectos. Sin embargo, es cierto que cuanto más complejo sea un sistema más importante es el uso del modelado para el desarrollo del mismo y mayores ventajas obtendremos.

Cabe destacar que, puesto que los modelos se utilizarán como guía, han de emplearse en las etapas iniciales del proceso de desarrollo. En la actualidad los modelos han llegado a ser una herramienta de desarrollo tan potente que han surgido paradigmas de desarrollo, como MDA, en las que se puede automatizar el proceso de pasar del modelo a código. En este paradigma de desarrollo se considera que los modelos y el código son dos caras de la misma moneda, y se intenta que los cambios en los modelos queden reflejados automáticamente en el código, respetando el código adicional introducido por los programadores. Del mismo modo se pretende que al cambiar el

código estos cambios queden reflejados en el modelo. Volviendo a la analogía del arquitecto, sería como si tras dibujar el plano, pulsando un botón, se construyera automáticamente el edificio, y si posteriormente se decide crear una ventana en una habitación al considerarse necesaria, el plano se redibujara automáticamente para contemplar este hecho.

Por último cabe destacar que cuando se utilizan modelos hay que tener especial cuidado con elegir el tipo de modelo correcto. Es lógico suponer que una mala elección de modelo o un modelo erróneo creará confusión y desorientación. Para comprender esto es fácil imaginar, por ejemplo, lo que ocurriría si en el plano del arquitecto no se respetara la misma escala en todas las habitaciones.

6.3 Modelado OO y Diseño OO

La tarea de creación de modelos es una de las actividades incluidas en la fase de diseño de los ciclos de vida que fueron mostrados en el apartado tres. Para construir dichos modelos deberá utilizarse la especificación de requisitos obtenidos en la fase de análisis por parte de la Ingeniería de Requisitos.

No obstante hay que diferenciar entre el modelado y el diseño. El modelado es el proceso de construcción de un modelo o especificación detallada de un problema del mundo real al que nos enfrentamos. Este proceso está aislado de toda consideración de diseño e implementación. Es por esto, por lo que se puede afirmar que modelado y diseño son dos cosas diferentes. Cuando se habla de modelado OO (orientado a objetos), se está hablando de la creación de modelos pero pensando en un desarrollo del sistema utilizando la programación orientada a objetos.

Así pues, ¿Qué es el modelado orientado a objetos? Según Booch el modelado orientado a objetos es: “Un proceso que examina los requisitos desde la perspectiva de las clases y objetos encontrados en el vocabulario del dominio del problema”. Las actividades que se deben realizar en el modelado orientado a objetos son:

- Identificación de las clases y la estructura del dominio del problema.
- Identificar y describir el comportamiento de los objetos.
- Identificar y definir las interacciones entre objetos.

Por otro lado el diseño orientado a objetos es un proceso que se encarga de refinar, extender y reorganizar las clases resultantes de la fase de modelado, así como especificar los atributos y los métodos de cada clase. Las actividades del diseño orientado a objetos son:

- Identificar las clases de negocio.
- Modelar el comportamiento de los objetos.
- Especificar todas las clases junto con sus atributos, métodos, relaciones y restricciones.
- Especificar las interfaces de usuario.

Podemos entender el modelado orientado a objetos como un primer borrador del modelo a construir teniendo como información la especificación de los requisitos obtenidos por la Ingeniería de Requisitos en la fase de análisis, y el diseño orientado a objetos como la optimización de dicho modelo.

El modelado proporciona la comprensión de un sistema y es conveniente saber que nunca suele ser suficiente con un único modelo. Para comprender algo, es más fácil si se emplean múltiples modelos relacionados, salvo en sistemas muy básicos. Por último, es necesario destacar que para poder realizar de manera correcta el modelado orientado a objetos, se deben captar los siguientes aspectos:

- **Estructurales**: deben ofrecer información sobre las clases de objetos que componen el sistema y también sobre las relaciones que existen entre las clases, ya sean de herencia, agregación, asociación, uso…
- **Comportamiento**: deben describir la secuencia de servicios que un objeto puede ofrecer.
- **Funcionales**: Deben describir la interacción entre los objetos para llevar a cabo sus funciones.

En los dos siguientes apartados se dará una visión de los modelos más usados actualmente en la Ingeniería del Software.

6.4 UML

UML (lenguaje unificado de modelado, del inglés: *Unified Modeling Language*) es el lenguaje estándar para escribir modelos. UML puede ser utilizado para visualizar, especificar, construir y documentar los componentes de un sistema que involucre una gran cantidad de software. Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y desplegar los sistemas, estableciendo estándares de apoyo a las fases iniciales del proceso, desde la especificación de plantillas para los documentos de la Ingeniería de Requisitos a la especificación de los modelos de la fase de diseño.

Hoy en día es inimaginable encontrar un Ingeniero del Software que no tenga un profundo conocimiento de UML de la misma manera que es impensable encontrar a un arquitecto incapaz de dibujar un plano.

UML es sólo un lenguaje por lo que es tan sólo una parte del desarrollo del software. Los lenguajes proporcionan un vocabulario y unas reglas para poder combinar los elementos de ese vocabulario con la intención de hacer posible la comunicación. Un lenguaje de modelado es un lenguaje cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema. Este vocabulario, junto con las reglas del lenguaje, indica cómo crear e interpretar los modelos, pero no dicen que modelos se deben crear, ni cuándo se debe hacer.

Se dice que UML es un lenguaje para visualizar, especificar, construir y documentar por los siguientes motivos:

- **Visualizar**: Es cierto que algunas cosas se modelan mejor de manera textual, pero existen otro tipo de cosas que se deben modelar de manera gráfica. UML no es solo un conjunto de símbolos gráficos, sino que detrás de cada uno de los símbolos existe una semántica bien definida. Con ello, un desarrollador es capaz de crear un modelo en UML y luego otro desarrollador es capaz de interpretarlo sin ambigüedad.
- **Especificar**: En el contexto actual, especificar significa construir modelos precisos, no ambiguos y completos. UML cubre la especificación de las decisiones de análisis, diseño e implantación que deben realizarse al desarrollar y desplegar un sistema con gran cantidad de software.

- **Construir**: UML no es un lenguaje de programación visual, pero sus modelos pueden conectarse de forma directa con diferentes lenguajes de programación. Gracias a ello, es posible establecer correspondencias desde un modelo UML a un lenguaje de programación.
- **Documentar**: UML cubre la documentación de la arquitectura de un sistema y todos sus detalles. También proporciona un lenguaje para expresar requisitos y pruebas, además de modelar las actividades de planificación de procesos y gestión de variaciones.

Es importante hablar sobre uno de los aspectos más importantes de UML en la construcción de modelos: los bloques básicos de UML. Estos bloques se dividen en tres tipos:

1. **Elementos UML**: Conjunto de elementos básicos que tienen una relación directa con algún objeto de la realidad.
2. **Relaciones**: Interacciones entre los elementos básicos.
3. **Diagramas**: Conjunto de Elementos básicos y relaciones que modelan una parte del sistema.

6.4.1 Elementos UML

Como se ha comentado un elemento UML es un elemento gráfico que comparte una relación directa con una parte la realidad. Estos elementos se dividen a su vez en cuatro elementos principales: estructurales, de comportamiento, de agrupación y de anotación

6.4.1.1 Elementos estructurales

Los elementos estructurales en UML, es su mayoría, son las partes estáticas del modelo y representan cosas que son conceptuales o materiales. Entre los elementos más importantes encontramos:

- **Clase**: Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.). La

clase es el elemento básico de los diagramas de clase. En UML, una clase es representada por un rectángulo que posee tres divisiones:

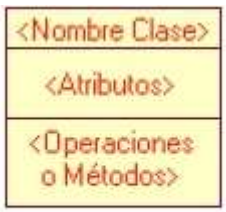


Figura 9. Clase UML

La parte superior contiene el nombre de la Clase. La parte intermedia: Contiene los atributos que caracterizan a la Clase. Estos atributos han de poseer una visibilidad indicando si dichos atributos pueden verse desde fuera del objeto (+) o no (-), un nombre y un tipo (si son números enteros, reales, secuencias de caracteres, etc.). Por último en la parte inferior contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno. Dichos métodos a su vez tienen una visibilidad, y opcionalmente atributos de entrada y/o de salida. Podemos ver un ejemplo de clase en la figura 10.

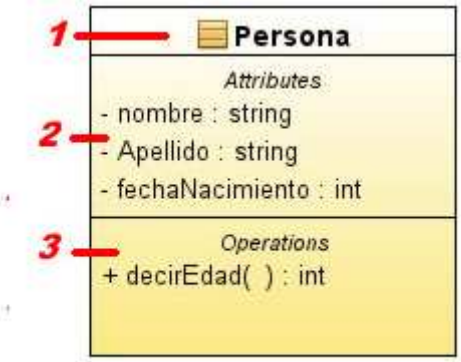


Figura 10. Ejemplo Clase UML

- **Interfaz**: es una colección de operaciones que especifican un servicio de una clase. Puede representar el comportamiento completo de una clase o sólo un parte de ese comportamiento. En la interfaz se definen una serie de especificaciones pero jamás un conjunto de las implementaciones de las operaciones. Su representación gráfica es idéntica a la de la clase, pero el nombre de la interfaz ha de estar precedido por: <<interface>>

- **Actor**: especifica un rol jugado por un usuario o cualquier otro sistema que interactúa con nuestro sistema, es decir representan roles jugados por usuarios humanos, hardware externo, aplicaciones externas, u otros sujetos. Un actor no necesariamente representa una entidad física específica, sino simplemente una faceta particular de alguna entidad. Así, una única instancia física puede jugar el rol de muchos actores diferentes y, asimismo, un actor dado puede ser interpretado por múltiples instancias diferentes. La figura 11 muestra la representación gráfica de un actor.

Figura 11. Ejemplo Actor UML

- **Caso de uso:** es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. Se utiliza para estructurar los aspectos de comportamiento en un modelo. En la figura 12 se muestra la representación gráfica de un caso de uso.

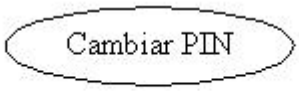


Figura 12. Ejemplo Caso de Uso UML

- **Colaboración:** define una interacción y una sociedad en la que diversos componentes o clases colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos. En la figura 13 se muestra la representación gráfica de la colaboración.

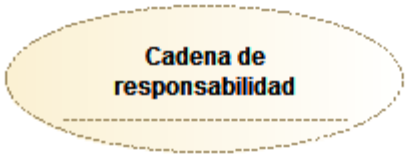


Figura 14. Ejemplo Colaboración UML

6.4.1.2 Elementos de comportamiento

Los elementos de comportamiento son las partes dinámicas de un modelo, representan el comportamiento en el tiempo y en el espacio. Estos elementos están conectados normalmente a varios elementos estructurales, principalmente clases, y colaboraciones. Los principales elementos son los dos que siguen.

- **Interacción:** Es un componente que comprende un conjunto de mensajes intercambiables entre un conjunto de objetos, dentro de un contexto particular para alcanzar un propósito específico. Se representa con una línea dirigida.

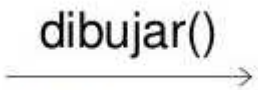


Figura 15. Interacción UML

- **Maquina de Estados**: Es un componente que especifica la secuencia de estados por las que pasa un objeto durante su vida en respuesta a eventos, junto con sus reacciones a estos eventos.

Figura 16. Estado UML

6.4.1.3 Elementos de agrupación

Son las partes organizativas de los modelos UML. Los elementos de agrupación son cajas en las que puede descomponerse un modelo. El principal tipo de estos elementos en UML se llama paquete. El paquete es un mecanismo de propósito general para organizar elementos, ya sean estructurales, elementos de comportamiento u otros paquetes.

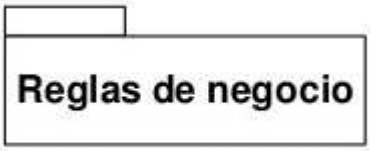


Figura 17. Paquete en UML

6.4.1.4 Elementos de anotación

Estos elementos se utilizan para las explicaciones dentro de los modelos UML. Constan de comentarios que se pueden aplicar para describir, clarificar y hacer observaciones sobre cualquier elemento de un modelo. El tipo principal de elementos de anotación se llama nota.

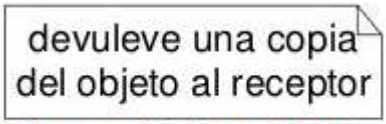


Figura 18. Nota UML

6.4.2 Relaciones UML

Existen cuatro tipos de relaciones entre los elementos de un modelo UML: dependencia, asociación, generalización y realización; estas se describen a continuación.

6.4.2.1 Dependencia

Es una relación semántica entre dos elementos en la cual un cambio a un elemento (el elemento independiente) puede afectar a la semántica del otro elemento (elemento dependiente). Se representa como una línea discontinua, posiblemente dirigida, que a veces incluye una etiqueta.

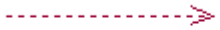


Figura 19. Dependencia UML

6.4.2.2 Asociación

Una asociación es una relación estructural entre clases que se describe mediante un conjunto de enlaces, los cuales son conexiones conceptuales entre dos o más objetos. Adicionalmente una asociación tiene una cardinalidad en cada uno de sus extremos. Esta cardinalidad indica con cuantos elementos como mínimo y como máximo puede relacionarse cada clase. Su representación en UML es una línea continua que puede ir etiquetada, puede verse en la siguiente figura:

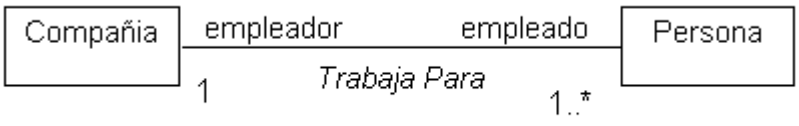


Figura 20. Asociación UML

Dentro del tipo asociación, existe dos tipos especiales que expresan una relación "*ser parte de*" en la que los objetos que representan los componentes de algo, se asocian con un objeto que representa el ensamblaje completo. Su representación en UML se diferencia de la de la asociación simple porque en un extremo aparece un rombo, indicando que la clase que lo contiene es el todo, formado a partir de elementos de la clase con la que se asocia. A estas formas de asociación más fuertemente acopladas se las conoce como:

- **Agregación**: Cuyo rombo es blanco, indicando que la destrucción del objeto que representa el ensamblaje no implica la destrucción de sus objetos componentes.
- **Composición**: Cuyo rombo es negro, indicando que la destrucción del objeto que representa el ensamblaje completo implica la destrucción de sus objetos componentes.

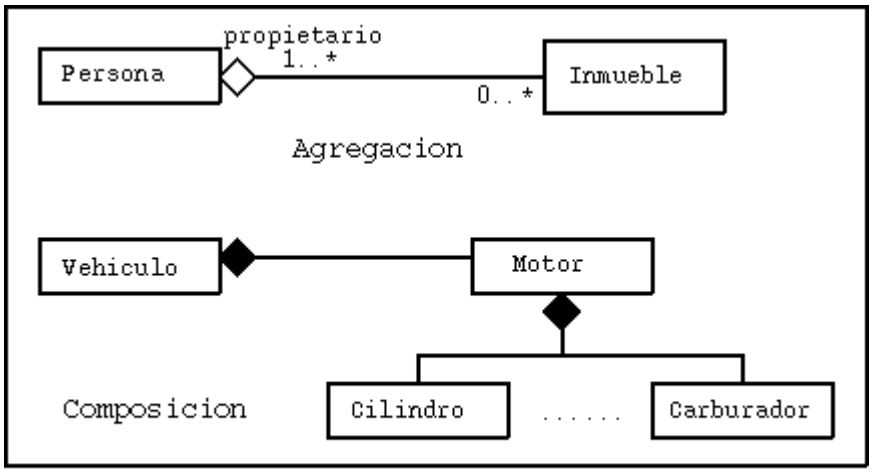


Figura 21. Agregación y Composición UML

6.4.2.3 Generalización

Es una relación de especialización/generalización, en la cual el elemento hijo (especializado) se basa en la especificación del elemento padre (generalizado). El hijo comparte la estructura y el comportamiento del elemento padre, motivo por el cual también se le conoce como herencia. Su representación en UML se hace mediante una flecha dirigida del hijo al padre.

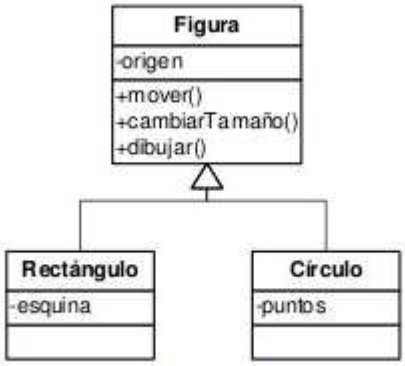


Figura 22. Generalización UML

6.4.3 Diagramas UML

El último de los bloques básicos de UML son los diagramas. Un diagrama es la representación gráfica de un conjunto de elementos, visualizado la mayoría de veces como un grafo conexo de nodos (elementos) y arcos (relaciones). Los diagramas suelen representar una visión resumida de las partes que forman un sistema. Existen diversos tipos de diagramas pero nos centraremos en los que puede que sean lo más utilizados.

6.4.3.1 Diagrama de clases

Son los diagramas más comunes en el modelado de productos orientados a objetos. Estos diagramas muestran un conjunto de clases, interfaces y colaboraciones y también todas las relaciones existentes.

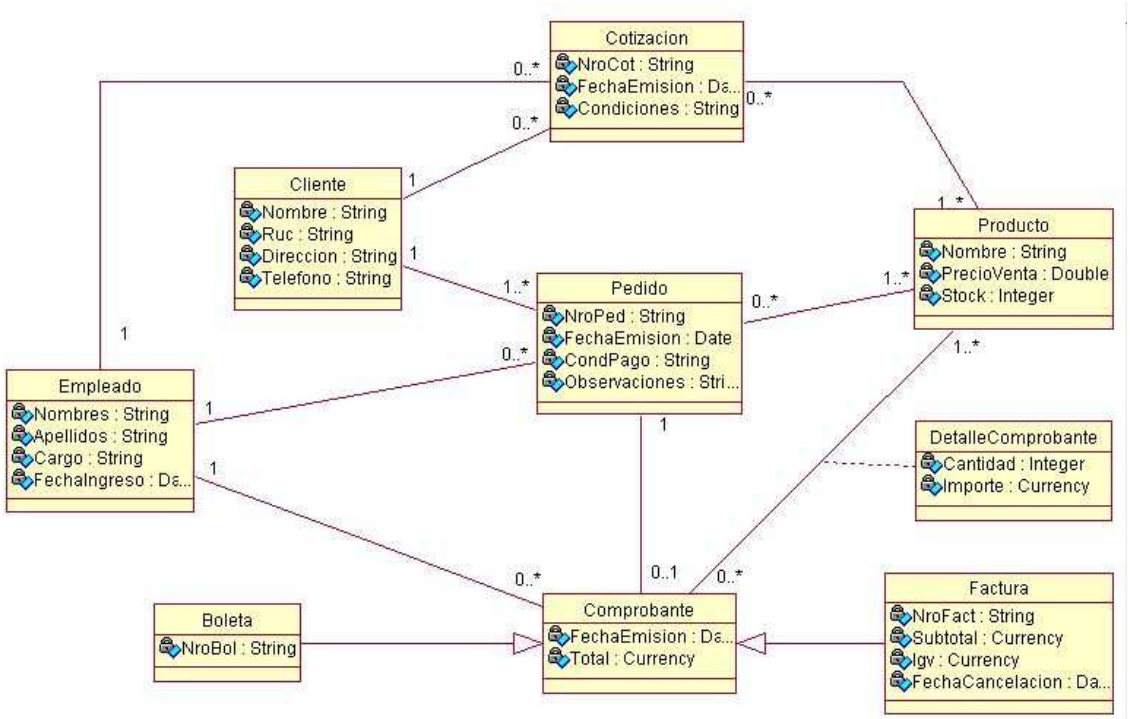


Figura 23. Diagrama de Clases

6.4.3.2 Diagrama de casos de uso

Muestra un conjunto de casos de uso y actores, y sus relaciones. Estos diagramas describen el comportamiento del sistema desde el punto de visa del usuario las funcionalidades del sistema. Son precisamente importantes en el modelado y organización del comportamiento del sistema.

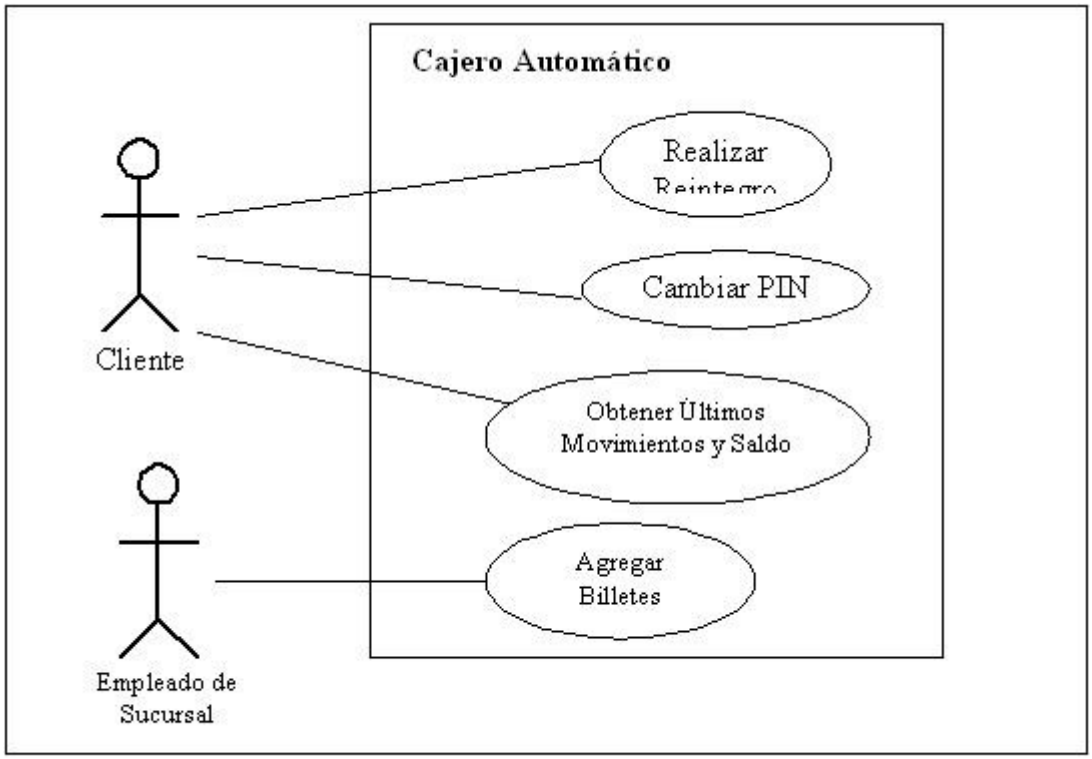


Figura 24. Diagrama de casos de uso

6.4.3.3 Diagrama de secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Mientras que el diagrama de casos de uso permite el modelado de una vista de negocio del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario y mensajes intercambiados entre los objetos.

Típicamente se examina la descripción de un caso de uso para determinar qué objetos son necesarios para la implementación del escenario. Si se dispone de la descripción de cada caso de uso como una secuencia de varios pasos, entonces se puede

"caminar sobre" esos pasos para descubrir qué objetos son necesarios para que se puedan seguir los pasos. Un diagrama de secuencia muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como flechas horizontales.

No obstante este diagrama está siendo desbancando poco a poco por los diagramas BPMN, al presentar una representación más cómoda y ágil.

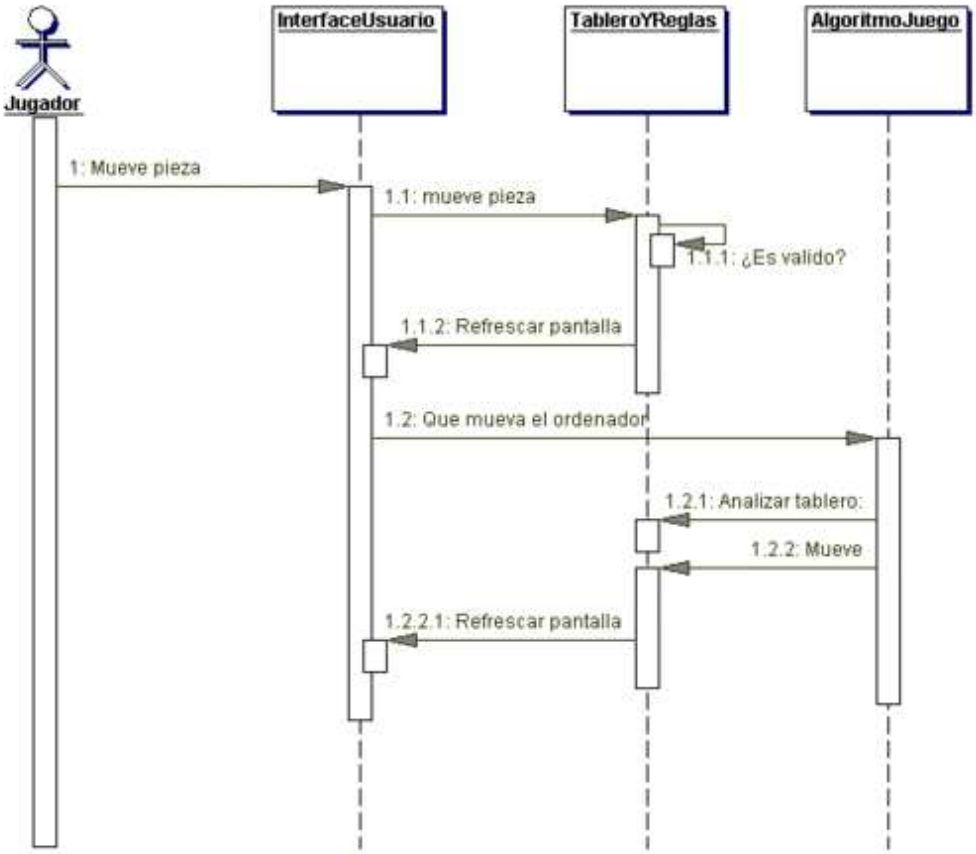


Figura 25. Diagrama de Secuencia

6.5 BPMN

Business Process Modeling Notation o BPMN (Notación para el Modelado de Procesos de Negocio) es una notación gráfica estandarizada que permite el modelado de procesos de negocio, en un formato de flujo de trabajo (workflow). BPMN representa el extremo a extremo de un proceso de negocio. La notación ha sido diseñada

específicamente para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes del mismo, con un conjunto de actividades relacionadas

Actualmente, un proceso de negocio ahora se extiende por varios participantes u organizaciones y la coordinación puede ser compleja. BPMN ha sido desarrollada para proporcionar a los usuarios una notación libre, beneficiando a los usuarios de una manera similar en que lo ha hecho el estándar UML en el mundo de la ingeniería de software. Así pues, podemos establecer el principal objetivo de BPMN como proporcionar una notación estándar, no ambigua, que sea fácilmente legible y entendible por parte de todos los involucrados e interesados del negocio (stakeholders), así como suficientemente elaborada para modelar procesos muy complejos.

La pregunta que nos hacemos a continuación es: ¿Procesos de Negocio? ¿No estábamos hablando de desarrollo de producto software? Si, efectivamente. A pesar de que BPM ha sido diseña pensando en el modelado de procesos de negocio, extrapolar BPMN para un proceso software es algo inmediato. Esto es debido a que los elementos del diagrama de BPMN tienen una correspondencia directa con los elementos en los diagramas de UML, y pueden ser usados como alternativa a los diagramas de secuencia.

El modelado en BPMN se realiza mediante diagramas muy simples, con un conjunto muy pequeño de elementos gráficos. Con esto se busca que para los usuarios del negocio y los desarrolladores técnicos sea fácil entender el flujo y el proceso. Las cuatro categorías básicas de elementos son: objetos de flujo, objetos de conexión, artefactos, pools y lanes

6.5.1 Objetos de flujo

Los objetos de flujo son, quizá, los elementos más importantes de los diagramas de BPMN. Son utilizados para modelar eventos que ocurren en el sistema, actividades a realizar o diferentes líneas de procesamiento dependiendo de una decisión. Se dividen en tres grupos principales: actividades, eventos y puertas.

6.5.1.1 Eventos

Un evento es algo que sucede durante el curso del proceso, afectando al flujo del proceso. Normalmente tienen una causa (trigger) o un impacto (resultad). Se

representan mediante círculos. Encontramos 3 tipos principales de eventos dependiendo de cuándo afectan el flujo:

- **Inicio:** Como su nombre lo indica, representa el punto de inicio de un proceso.
- **Intermedio:** Ocurren en el transcurso de una actividad y entre eventos de inicio y de fin. Afectará el proceso pero no lo iniciará o directamente finalizará.
- **Fin:** Indica cuando un proceso termina.

Los eventos pueden tener un subtipo dependiendo del impacto en el flujo del proceso, que se indica mediante un dibujo en el interior del círculo, y adicionalmente se consideran de envío o de recepción dependiendo si el dicho dibujo está, o no, coloreado. Existen multitud de subtipos pero quizás los más importantes sean:

- **Mensaje**: Un evento de Mensaje puede ser usado tanto para enviar como para recibir un mensaje. Esto causa que el proceso continúe si éste estaba esperando por el mensaje o cambia el flujo para manejo de excepciones. Para atrapar y lanzar mensajes debe tener el mismo nombre.
- **Temporizador**: Un evento temporizador puede ser usado tanto para definir retrasos dentro del proceso, como para programar acciones en el tiempo (por ejemplo, un evento se dispara en 3 min o todos los lunes a las 9:00).
- **Terminador:** Si el proceso alcanza este evento, éste será cerrado, independientemente de que tenga o no flujos paralelos.
- **Cancelación:** Este evento es similar al evento terminador, pero causa la activación de los eventos compensatorios.
- **Compensatorio:** Cuando un flujo de proceso es cancelado, es posible que sea necesario deshacer alguno de los cambios hechos por alguna actividad previa. El evento compensatorio capturará los eventos de cancelación
- **Enlace**: Sirve para desviar el flujo a una página diferente del diagrama.
- **Señal**: Utilizado para enviar señales que pueden ser capturadas por diferentes actividades simultáneamente

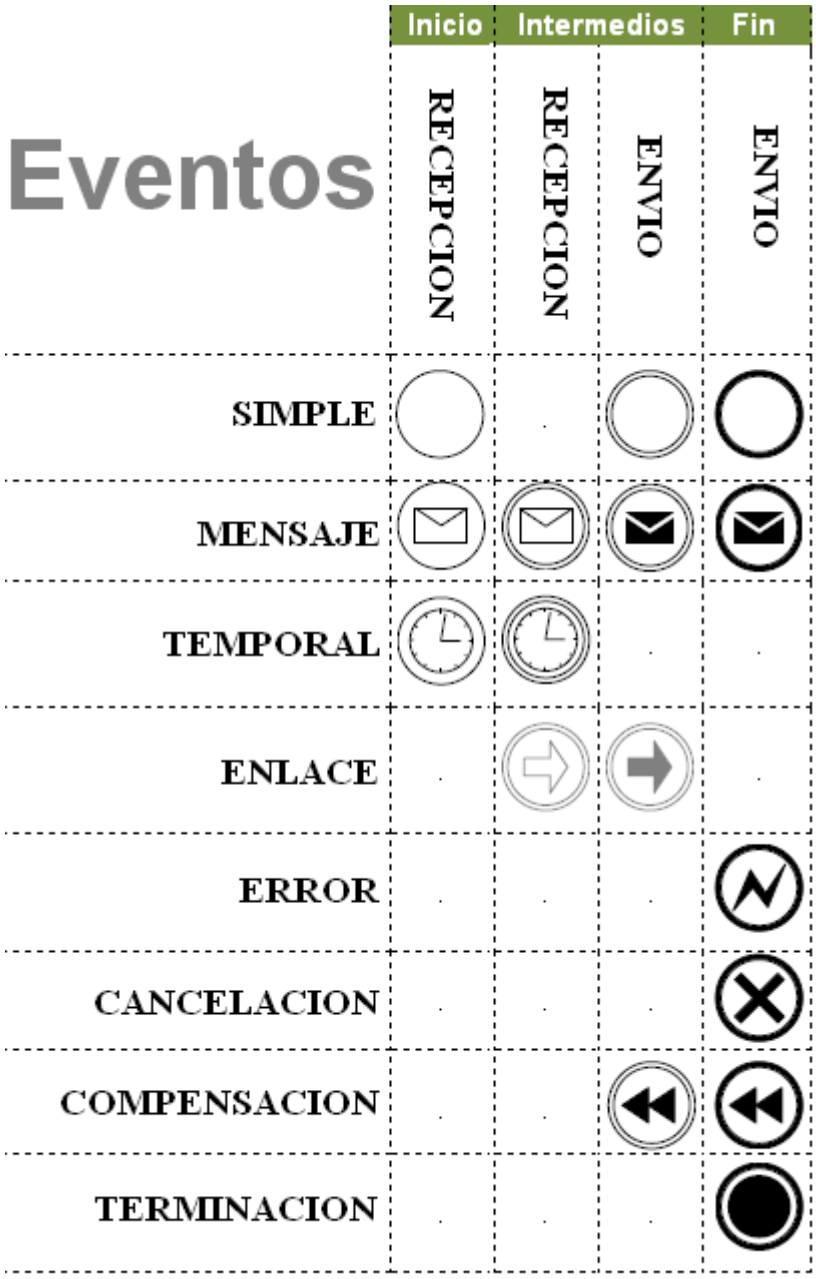


Figura 26. Resumen Evento BPMN

6.5.1.2 Actividades

Una actividad es el nombre genérico que recibe una porción del trabajo dentro un proceso. Representa una tarea o secuencia de tareas a realizar. El nombre de la actividad es muy importante, pues ha de describir que trabajo se lleva a cabo en dicha tarea. La figura 27 muestra la representación gráfica de una actividad.

Figura 27. Tarea BPMN

Adicionalmente una actividad puede llevar asociado un símbolo, llamado marcador de tarea, reflejando de qué manera ha de realizarse dicha tarea. Dicho marcadores de tarea pueden ser:

- **Subproceso**: Indica que es una actividad compleja, que realiza más de una tarea. Dicha actividad debe estar modelada en detalla en algún diagrama BPMN
- **Ciclo**: Indica que dicha actividad se ejecuta varias veces.
- **Instancias múltiples en paralelo**: Indica que la tarea se divide en subrutinas que se ejecutan en paralelo
- **Instancias múltiples en secuencia**: Indica que la tarea se divide en subrutinas que se ejecutan una detrás de otra.
- **Ad Hoc**: Indica que la actividad es muy específica. Da respuesta al problema en el que se está trabajando, sin dotar al desarrollo de la necesaria modularidad que permita reutilizar sus componentes en el futuro.
- **Compensación**: Es una tarea diseñada específicamente para cuando se producen eventos de cancelación.

La figura 28 muestra la representación grafica de los marcadores de tarea. Estos símbolos se representaran en la parte inferior del rectángulo que modela la actividad.

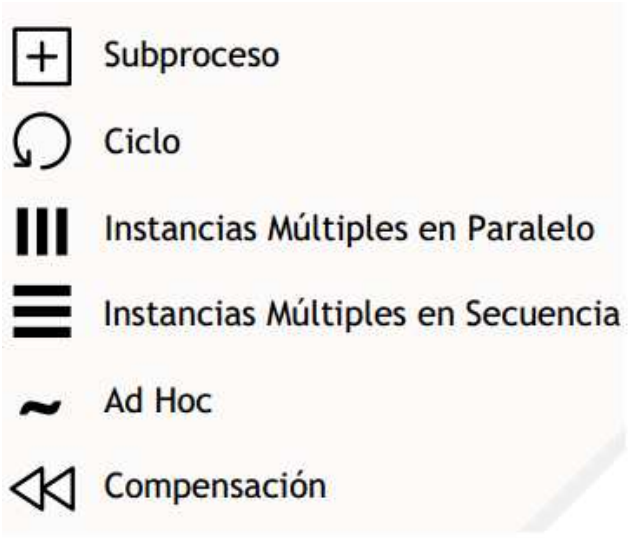


Figura 28. Marcador de actividad

6.5.1.3 Puertas

Las Decisiones son usadas para controlar la divergencia y convergencia del flujo del proceso. Éstas determinan ramificaciones, bifurcaciones y combinaciones basadas en alguna decisión discriminatoria, así como y fusiones de dichas ramificaciones. Encontramos cinco tipos de puertas:

- **Exclusiva**: En un punto de bifurcación, selecciona exactamente un flujo de secuencia de entre las alternativas existentes. En un punto de convergencia, la compuerta espera a que un flujo incidente complete para activar el flujo saliente. Esta puerta tiene dos representaciones posibles que pueden ser usados indistintamente.
- **Inclusiva**: En un punto de bifurcación, al menos un flujo es activado. En un punto de convergencia, espera a todos los flujos que fueron activados para activar al saliente.
- **Paralela**: En un punto de bifurcación, todos los caminos salientes serán activados simultáneamente. En un punto de convergencia, la compuerta espera a que todos los flujos incidentes completen antes de activar el flujo saliente.
- **Basada en eventos**: En este caso, la puerta solo podrá estar unida a eventos intermedios de recepción. El flujo de trabajo quedará a la espera en la puerta, y saldrá por aquel camino cuyo evento intermedio sea el primero en ser capturado.
- **Compleja**: En este caso la elección del camino de salida depende de una decisión compleja, no capturada por el resto de puertas y que deberá estar escrita al lado de cada camino.

La figura 29 muestra la representación gráfica de cada puerta.

Figura 29. Puertas BPMN

6.5.2 Objetos de conexión

Como su propio nombre indica los objetos de conexión tiene como objetivo conectar los diferentes elementos del diagrama. Encontramos tres tipos básicos de conectores:

- **Flujo de secuencia**: Son usados para interconectar objetos de flujo. Al ser arcos dirigidos, indican el orden en el que han de ejecutarse las actividades.
- **Flujo de mensaje**: Se usan para indicar el flujo de mensajes que se mandan dos actividades o procesos. Dichas actividades deben estar en piscinas diferentes.
- **Asociación**: Usados para asociar artefactos con objetos de flujo.

La figura muestra la representación de los diferentes objetos de conexión.

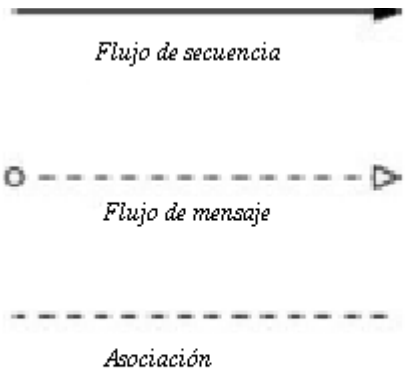


Figura 30. Objetos de Conexión

6.5.3 Pools y Lane

Las Pools o piscinas, representan los principales participantes de un proceso, por lo general, separados por las diferentes organizaciones. Una piscina puede ser abierta mostrando cada de sus carriles y procesos internos, o cerrada escondiendo el detalle interno, representándose como un rectángulo vacío que se extiende a lo ancho o alto del diagrama.

A su vez, una piscina contiene uno o más Lanes (carriles). Los carriles son usados para organizar y categorizar las actividades dentro de una piscina de acuerdo a su función o rol. La figura 31 muestra la representación gráficas de los carriles y piscinas.

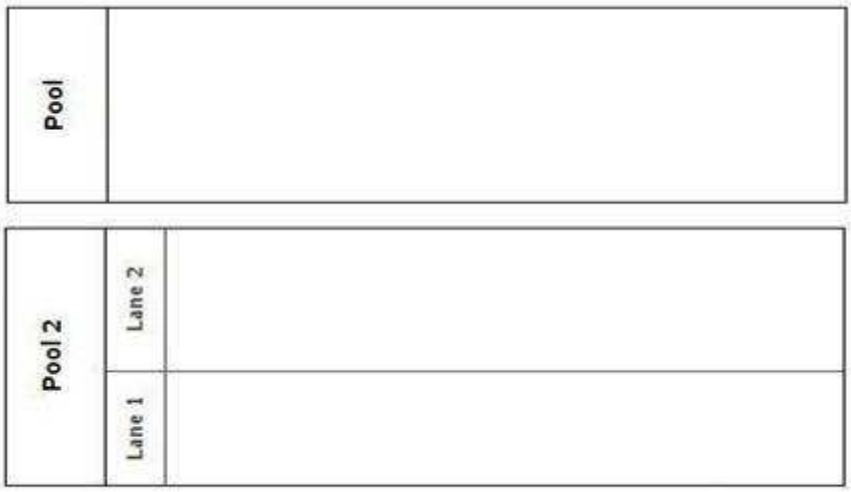


Figura 31. Piscina y carril

6.5.4 Artefactos

El último elemento gráfico en los diagramas BPMN son los artefactos. Los Artefactos permiten a los desarrolladores llevar algo más de información al modelo o diagrama. De esta manera, el modelo o diagrama se hace más legible. Básicamente hay tres tipos de artefactos:

- **Objetos de Datos:** Muestra al lector cual es el dato que deberá ser requerido o producido en una actividad.
- **Grupos:** Se representan por un rectángulo de líneas discontinuas y vértices redondeados. El Grupo se utiliza para agrupar diferentes actividades pero no afecta al flujo dentro de un diagrama.
- **Anotación:** Se utiliza para darle al lector una descripción textual del modelo o diagrama.

La figura 32 muestra la representación gráfica de los artefactos.

.

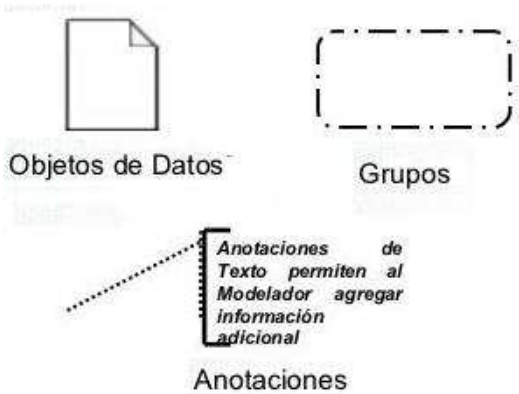


Figura 32. Artefactos BPMN

6.5.5 Diagrama BPMN

Por último, no podríamos acabar este apartado sin dar una visión general de cómo se conectan todos estos elementos entre sí. La figura 33 muestra un ejemplo de diagrama BPMN completo.

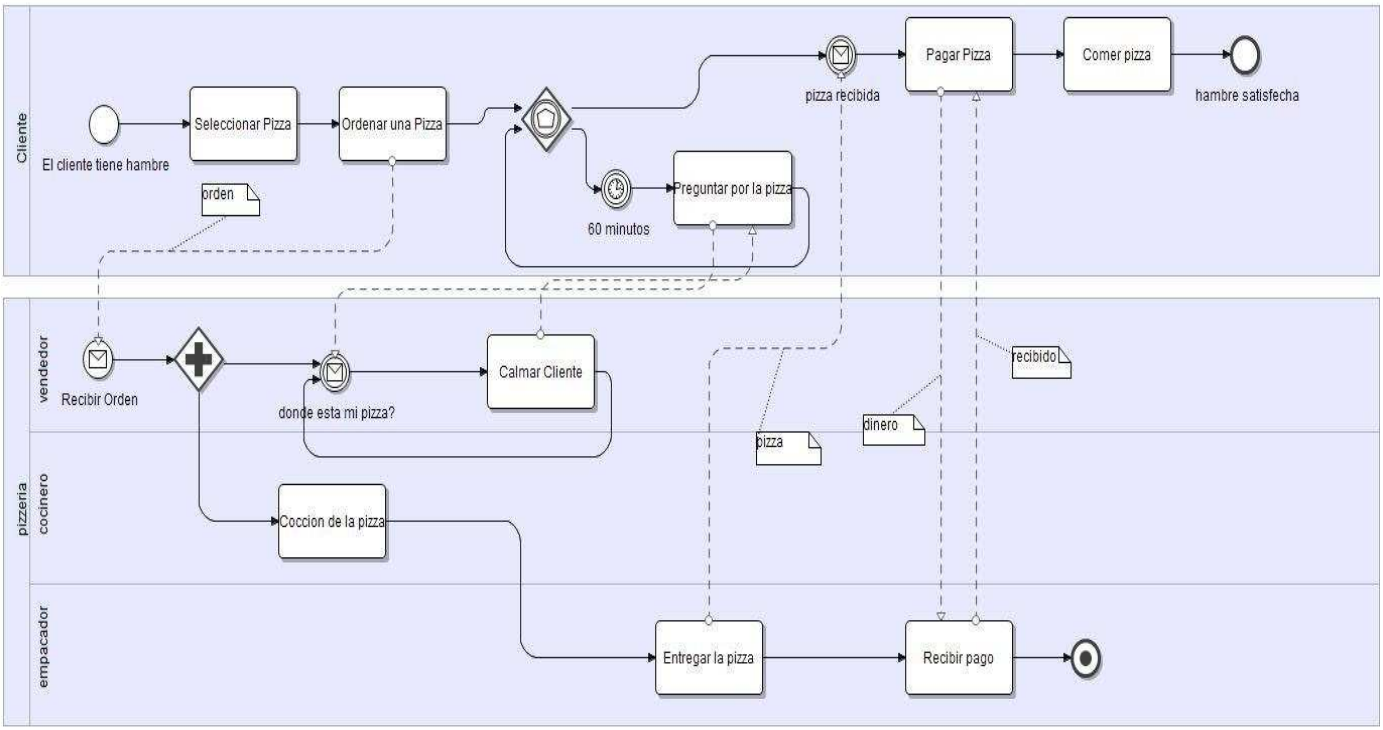


Figura 33. Ejemplo BPMN

7. Desarrollo de la Aplicación

A continuación se aplicarán los conceptos explicados en los apartados anteriores enfocados a desarrollar la aplicación del presente proyecto. Como se comento al inicio del documento, dicha aplicación pretende ser una herramienta software para gestionar y catalogar las quejas presentadas por los ciudadanos al ayuntamiento de Torrent.

Cabe destacar que el ciclo de vida elegido para desarrollar la aplicación ha sido el modelo evolutivo incremental. Por ello, durante todas las iteraciones realizadas han ido apareciendo nuevos requisitos capturados en sesiones de brainstorming, se han eliminado otros, se han generado nuevos documentos visión resultantes de dichas reuniones, se han desarrollado partes de la aplicación nuevas y se han descartado otras por considerarse innecesarias. En los siguientes apartados se da una visión global del producto final y no el detalle de la evolución del proyecto, en el que aparecerían cada una de las versiones desarrolladas en las diferentes iteraciones.

7.1 Requisitos funcionales

Los requisitos funcionales capturados en las diversas reuniones mantenidas con el cliente se detallan en los siguientes apartados, agrupándolos por cada uno de los módulos de la aplicación.

7.1.1 Gestión de quejas

- La aplicación deberá capturar las quejas que los ciudadanos presenten vía Web.
- La aplicación deberá capturar las quejas que los ciudadanos presenten por teléfono y/o personándose en el ayuntamiento. Entre otras tareas, dichas quejas serán capturadas por la aplicación “010” y será necesario que la aplicación a desarrollar puede trabajar coordinadamente con dicha aplicación.
- Las quejas a tratar serán cualquier tipo de queja ciudadana, las solicitudes de los ciudadanos y las comunicaciones de los ciudadanos hacia la alcaldesa.
- Se debe poder contestar las quejas por correo sin salir de la aplicación.
- Debe poderse obtener información del ciudadano que presento las queja.
- Deben poder consultarse quejas antiguas.

- La aplicación debe ofrecer alguna funcionalidad para filtrar las quejas presentadas. Se contemplan los filtros por: departamento, estado actual, email, distrito, fecha, dirección, y nombre.
- Las quejas deben poder ser redirigidas al organismo del ayuntamiento adecuado.
- Las quejas han de clasificarse según el área de gobierno al que pertenecen.
- Las quejas han de clasificarse según el distrito del ciudadano que presento la queja.
- Ha de poder recuperarse todo el proceso que siguió la queja, desde su entrada en el sistema hasta la contestación al ciudadano.
- La aplicación ha de ofrecer en el momento de capturar una queja nueva, información sobre todas las quejas previas presentadas por el mismo ciudadano, con el objetivo de ofrecer un trato personalizado.
- Las quejan pueden ser archivadas sin necesidad de contestar al ciudadano por correo electrónico.
- Los usuarios solo deben poder ver las quejas atendiendo a su tipo.
- Solo el personal del departamento de atención ciudadana y la alcaldesa pueden contestar al ciudadano.

7.1.2 Gestión de Información

- El administrador debe poder crear, modificar y borrar usuarios.
- El administrador debe poder cambiar los permisos de un usuario.
- El administrador debe poder crear y modificar departamentos.

7.1.3 Gestión de Avisos

- Se debe poder mandar un email a cada departamento con las quejas que les fueron reenviadas solicitando información y que han transcurrido siete días desde la solicitud.
- Se debe poder generar un correo a la alcaldesa informándole de los retrasos en la respuesta de dichos departamentos.

7.1.4 Gestión de Informes

- Se deben poder generar los siguientes informes:

- Los veinte ciudadanos que más quejas han presentado.
- Número de quejas presentadas por cada distrito en un rango de fechas específico.
- Número de quejas por mes en un año especifico.
- Quejas que llevan más de una semana en el buzón de entrada.
- Quejas que llevan más de una semana esperando la respuesta de un departamento a una solicitud de información.
- Tiempos de respuesta por los departamentos en un rango de fechas específico.
- Tiempo de respuesta de cada queja ciudadana en un rango de fechas específico.

7.1.5 Gestión de Usuarios

- Un empleado del ayuntamiento debe poder loguearse en la aplicación.

7.2 Requisitos no funcionales

A continuación se detallan los requisitos no funcionales capturados en las diversas reuniones mantenidas por el equipo de desarrollo:

- Solo el personal del ayuntamiento debe poder utilizar la aplicación.
- La comprobación de usuario y contraseña en el loggin debe estar integrado con el directorio activo de Windows.
- El envío de correo ha de estar integrado con la aplicación de Microsoft Outlook.
- Las Interfaces han de desarrollarse en Microsoft Access 2003.
- Las tablas locales han de implementarse usando una base de daos de Microsoft Access 2003.
- La conexión a bases de datos externas ha de hacerse a través de un ODBC.
- La lógica del programa ha de ser implementada en el lenguaje Visual Basic.
- Ha de asignarse a cada usuario de la aplicación un permiso de entre:
  - **Invitado**: El rol por defecto para cualquier personal del ayuntamiento. Los usuarios con estos permisos podrán:
    - Ver Informes.
  - **Queja Ciudadana**: Los usuarios con estos permisos podrán:

        - Consultar, archivar i cerrar quejas o sugerencias clasificadas como quejas ciudadanas.
        - Solicitar información a otros departamentos.
        - Ver informes.
    - **Alcaldía**: Los usuarios con estos permisos podrán:
        - Consultar, archivar i cerrar quejas o sugerencias clasificadas como escritos a la alcaldesa.
        - Solicitar información a otros departamentos.
        - Contestar al ciudadano.
        - Ver informes.
    - **Atención ciudadana**: Los usuarios con estos permisos podrán:
        - Consultar, archivar i cerrar cualquier tipo de quejas o sugerencias.
        - Solicitar información a otros departamentos.
        - Contestar al ciudadano.
        - Ver informes.
        - Generar emails de aviso.
    - **Administrador**: Ha de existir como mínimo un usuario Administrador. Los usuarios con estos permisos podrán:
        - Consultar, archivar i cerrar cualquier tipo de quejas o sugerencias.
        - Solicitar información a otros departamentos.
        - Contestar al ciudadano.
        - Ver informes.
        - Generar emails de aviso.
        - Crear, modificar y borrar usuarios.
        - Asignar permisos a usuarios.
        - Crear o modificar departamentos.

7.3 Diagrama de clases

Dichos requisitos funcionales eran posteriormente analizados en la fase de diseño. Gracias a dicho estudio se pudo identificar los objetos que era necesario modelar en la aplicación. En la figura siguiente se detallan los objetos mediante un diagrama de clases.

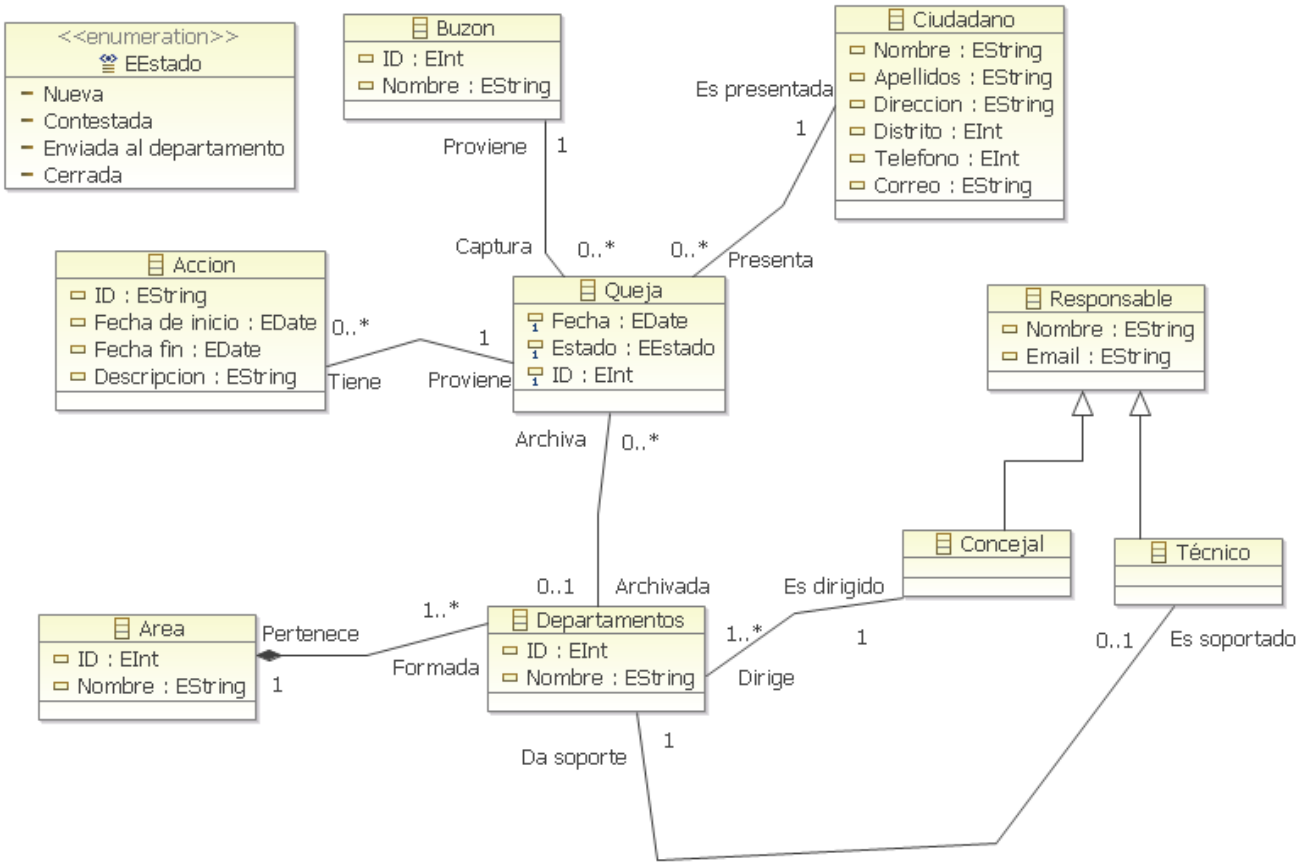


Figura 34. Diagrama de Clases de la aplicación

7.4 Diagrama BPMN

En las diferentes fases de diseño también se generaba un diagrama BPMN esquematizando la función a desarrollar en esa iteración, mostrando las tareas a realizar y como debía ser el intercambio de mensajes. A continuación se muestra únicamente el diagrama BPMN con el proceso de catalogar las quejas. Dicho proceso es la parte central de la aplicación.

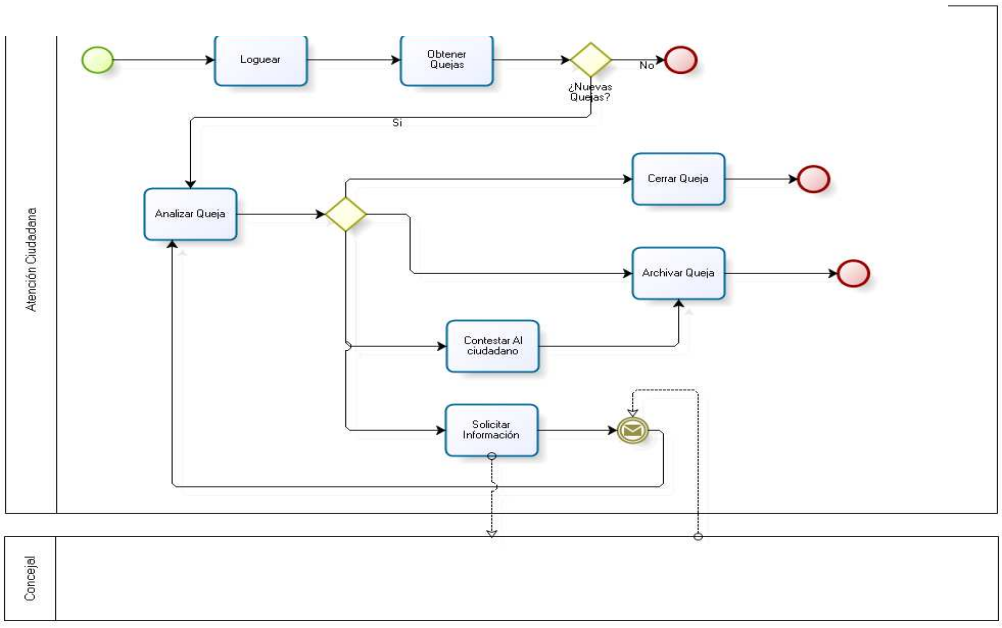


Figura 35. BPMN Catalogar Queja

7.5 Diagrama de casos de uso

El último documento generado en la fase de análisis fue un diagrama de casos de uso, con una explicación más detalla de cada funcionalidad, que complementaba al diagrama BPMN generado.

Los diferentes casos y una breve explicación de cada uno:

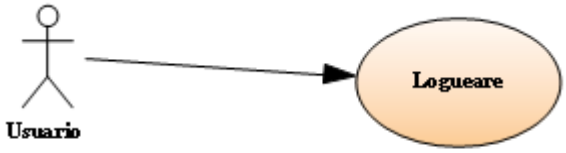


Figura 36. 1º Diagrama de casos de uso

- **Logarse:** Un empleado del ayuntamiento deberá logarse al iniciar la aplicación. Para ello, debe usar su usuario y contraseña en el directorio activo de Windows. De existir dicho usuario en el directorio activo, se comprobará en una tabla local los permisos que tiene asociados.

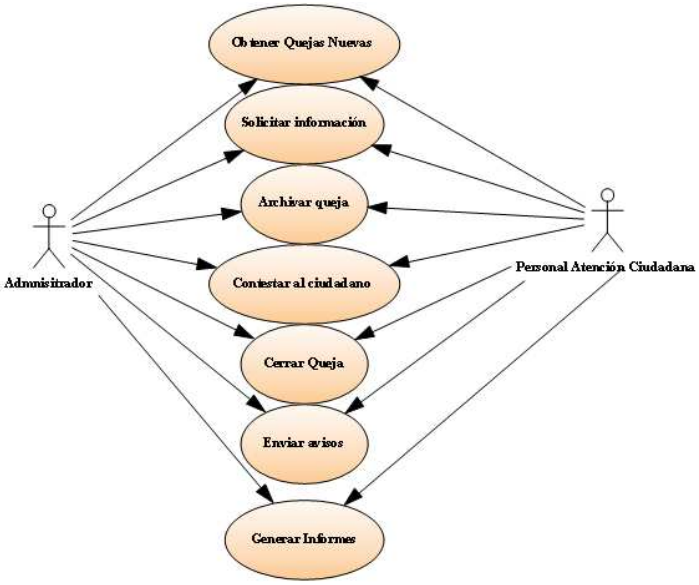


Figura 38. 3º Diagrama de casos de uso

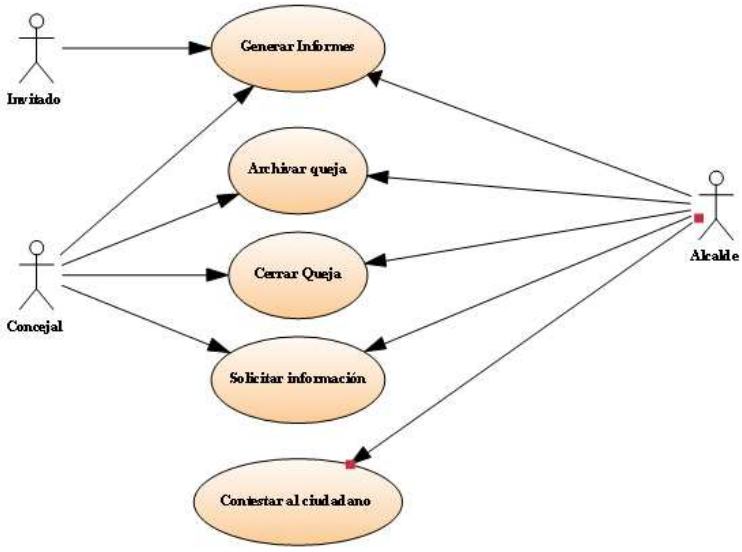


Figura 37. 2º Diagrama de casos de uso

En los diagramas 38 y 39 se han repetido los casos de uso para poder presentar los diagramas de forma más clara.

- **Generar Informes:** Cualquier usuario logado podrá generar informes. Para ello simplemente habrá de pulsar el botón Informes.
- **Obtener Quejas Nuevas:** Esta funcionalidad puede ser usada por el administrador, por el personal de Atención Ciudadana y por Alcaldía. Pulsando el botón *Catalogar nuevas quejas*, se conectará a las bases de datos externas y se copiarán a las tablas locales las nuevas quejas enviadas por los ciudadanos.
- **Solicitar información:** Esta funcionalidad puede ser usada por cualquier usuario logado a excepción de los invitados. Se elegirá un área de un menú desplegable y a continuación un departamento de dicha área. Tras escribir el mensaje a enviar se pulsará el botón *Vista previa del correo*, trasladándose la información a Microsoft Outlook.
- **Archivar queja:** Esta funcionalidad puede ser usada por cualquier usuario logado a excepción de los invitados. Tras elegir el área y el departamento, deberá introducirse la contestación que se le dio al ciudadano por teléfono y pulsar el botón *archivar queja*.
- **Contestar al ciudadano:** Esta funcionalidad puede ser usada por el administrador, por el personal de Atención Ciudadana y por Alcaldía. Se elegirá un área de un menú desplegable y a continuación un departamento de dicha área. Tras escribir el mensaje a enviar se pulsará el botón *Vista previa del correo*, trasladándose la información a Microsoft Outlook.
- **Cerrar Queja:** Esta funcionalidad puede ser usada por cualquier usuario logado a excepción de los invitados. Tras elegir el área y el departamento, deberá introducirse el motivo por el que se desestima la queja y pulsar el botón *desestimar queja*.
- **Enviar Avisos:** Únicamente el administrador o el personal de atención ciudadana puede utilizar esta funcionalidad. Pulsado el botón Enviar quejas, se generarán un correo por departamento incluyendo las quejas retrasadas de dicho departamento. Adicionalmente se generará un correo a la alcaldesa con un resumen de los correos mandados. Estos correos se mandarán utilizando la cuenta de Microsoft Outlook del usuario.

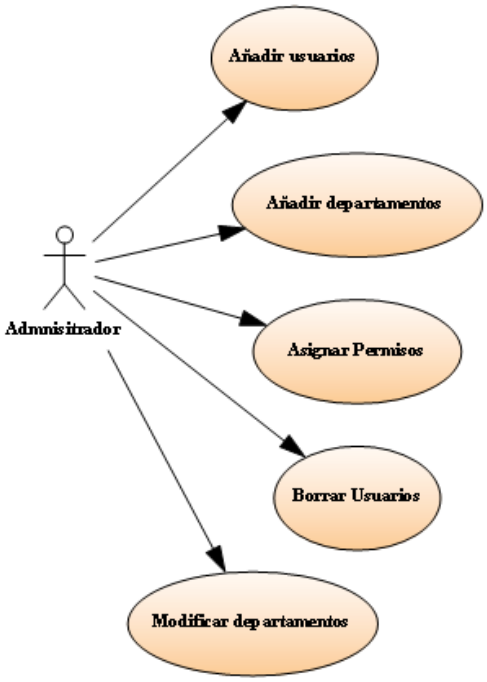


Figura 39. 4º Diagrama de casos de uso

- **Añadir usuario:** El administrador del sistema introducirá el loggin del usuario y seleccionará unos permisos del menú desplegable. Pulsando *Añadir usuario* se creará el usuario.
- **Añadir departamento:** El administrador introducirá el nombre del departamento, el área al que pertenece, el nombre y correo del concejal y, opcionalmente, el nombre y correo del técnico. Pulsando en el botón. *Añadir departamento* se creará el departamento.
- **Asignar permisos:** El administrador seleccionará un usuario del menú desplegable y elegirá un permiso disponible de otro menú desplegable. Pulsando en *Modificar*, se cambiarán los permisos del usuario.
- **Borrar Usuario.** El administrador seleccionará un usuario del menú desplegable. Pulsando en *Borrar* se eliminará dicho usuario del sistema.
- **Modificar departamento:** El administrador seleccionará un departamento del menú desplegable. Podrá modificar el nombre del departamento, el área al que pertenece, el nombre y correo del concejal y el nombre y correo del técnico. Pulsando en el botón *Modificar departamento* se guardarán los cambios.

7.6 Codificación

Una vez terminada la fase de diseño, los diferentes documentos generados fueron utilizados como guía para la creación del código. En esta fase era implementada la nueva funcionalidad y, una vez testeada, se volvía a concertar una reunión para mostrar la nueva versión. En dicha reunión el cliente debía aprobar o rechazar los cambios, y adicionalmente se obtenían nuevos requisitos para la nueva iteración del método incremental. Algunas de las decisiones técnicas más relevantes que se tomaron referentes al código fueron:

- La interfaz de la aplicación debía desarrollarse mediante formularios de Microsoft Access.
- El lenguaje de programación elegido habría de ser Visual Basic.
- La aplicación debía trabajar haciendo uso de tablas locales, y no directamente con las tablas y servidores desde los que obtiene los datos.
- Para conectarse a tablas externas se haría uso de la herramienta de ODBC.

7.7 Conclusiones

Uno de los objetivos perseguidos al inicio del proyecto era estandarizar el proceso de gestión de las quejas. Objetivo que ha sido alcanzado satisfactoriamente. Tras unos meses de uso de la aplicación se ha podido comprobar que ha tenido una gran aceptación en el personal y se ha integrado satisfactoriamente en el trabajo administrativo. Así mismo el tiempo de respuesta al ciudadano ha sido reducido considerablemente y ahora las respuestas se ofrecen siguiendo un modelo estándar.

La parte más positiva que se ha extraído durante la realización del proyecto ha sido comprobar como se aplican los conocimientos adquiridos durante la carrera a un entorno de producción real. A lo largo de los años de estudio hay una visión muy importante que se pierde en el proceso de adquirir conocimiento, y es el factor humano. Al desarrollar una aplicación en el entorno universitario estamos acostumbrados a trabajar sobre documentos muy bien tipificados y nuestros clientes son a la vez ingenieros que adaptan sus demandas a los problemas que saben que nos encontraremos.

No obstante, no hay que olvidar que la tarea de un ingeniero de software se verá reflejada en última instancia en un cliente y unos usuarios. El principal reto de este proyecto residía en ese hecho, trasladar las destrezas adquiridas en la carrera a un entorno real, en la que los usuarios pueden presentar ideas contradictorias o irrealizables, hay fechas de entrega, no todos los usuarios desean lo mismo de la aplicación o hay información relevante que no ha sido extraída en la fase de análisis.

Para finalizar este documento, puedo afirmar que dicho reto lo he superado satisfactoriamente, dado que la aplicación es utilizada diariamente por los responsables de la Gestión de Quejas y Atención al Ciudadano del Ayuntamiento de Torrent.

8. Bibliografía

La bibliografía consultada durante la realización del presente documento ha sido:

1. Transparencias de la asignatura Ingeniería de la Programación.
2. Transparencias de la asignatura Ingeniería de requisitos.
3. www.wikipedia.org
4. http://www.ieee.org
5. PRESSMAN, Roger S. Ingeniería del Software: un enfoque práctico. Editorial McGrawHill. Ediciones 5º (2001) y 6º (2005). ISBN: 8448132149.
6. http://www.alegsa.com.ar/Dic/software.php
7. http://www.mitecnologico.com/Main/DefinicionIngenieriaDeSoftware
8. http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node10.html
9. http://www.es.scribd.com/doc/18286706/El-Ciclode-Vida-Del-Software
10. www.bpmn.org
11. www.uml.org

9. Anexo I

MANUAL DE USO

Quejas y Sugerencias

Índice

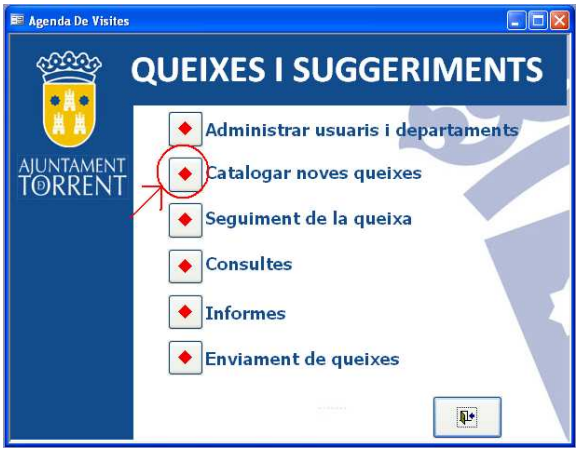
1. Login

Esta es la ventana que nos aparecerá al arrancar la aplicación. En ella deberemos introducir nuestro usuario y contraseña de Windows y pulsaremos ***Entrar***. Si hemos introducido bien los datos accederemos a la aplicación y dependiendo de los permisos que nos hayan otorgado podremos ver unos apartados u otros.

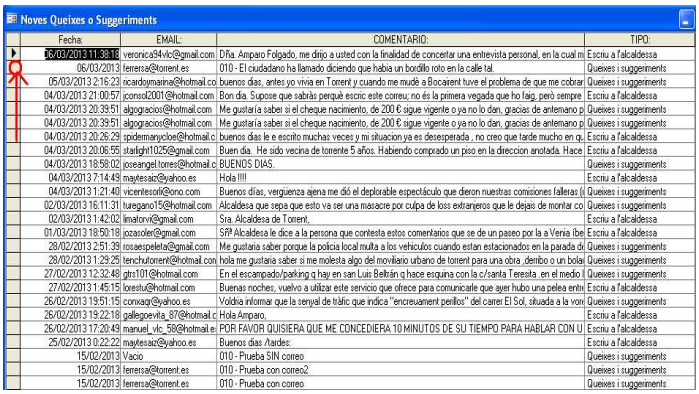
2. Catalogar nuevas quejas

En esta parte de la aplicación podremos catalogar las nuevas quejas que hayamos recibido. Dependiendo de los permisos de los que disponga nuestro usuario veremos las quejas o sugerencias normales o las quejas y sugerencias de alcaldía.

Pincharemos en el botón mostrado en la imagen para acceder al catalogo de nuevas quejas.



Nos aparecerá un listado con las nuevas quejas que hemos recibido. Dependiendo de nuestros permisos podremos ver cada uno de los distintos tipos de quejas:



- Tipo 1: Quejas de alcaldía.
- Tipo 2: Quejas normales.

Seleccionado una la abriremos para catalogarla.

En esta ventana veremos toda la información de la queja. En el listado superior veremos quejas que hayan sido mandadas por la misma persona. En el listado inferior veremos toda la información del seguimiento de la queja actual (que en este momento esta vacío al ser una queja nueva): si fue enviada al departamento, si esta contestada…

Antes de poder catalogar la queja deberemos seleccionar el distrito desde el que se mandó la queja. Una vez seleccionado deberemos pinchar en uno de los botones inferiores dependiendo de lo que vayamos a hacer con la queja:

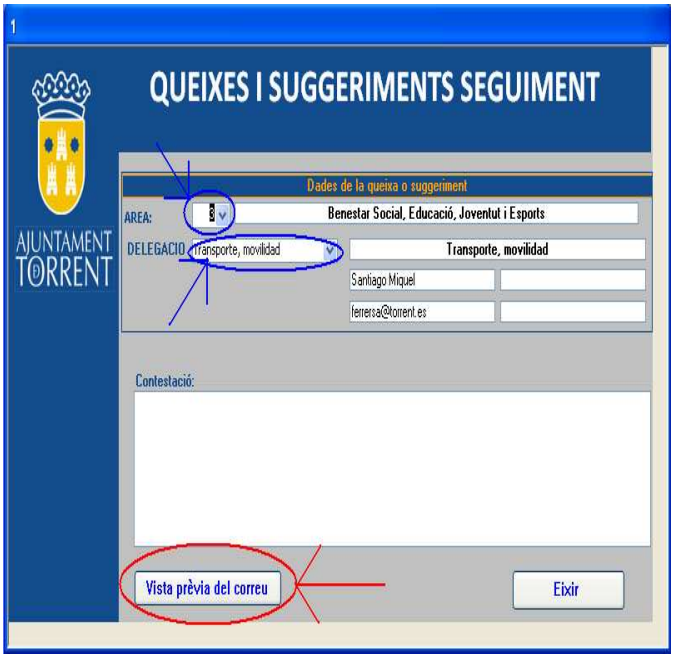
Contestar al ciutadà: En este apartado contestaremos directamente al ciudadano enviándole un correo a través de la aplicación. La queja quedará archivada como contestada.

Demanar informació: En este apartado enviaremos un correo a un departamento solicitándole información y dejaremos la queja a la espera de respuesta.

Archivar la queja: Si hemos contestado al ciudadano por otros medio, por teléfono por ejemplo, usaremos esta parte de la aplicación. Su funcionamiento es igual que el apartado "Contestar al ciutadà", pero no se enviará ningún correo.

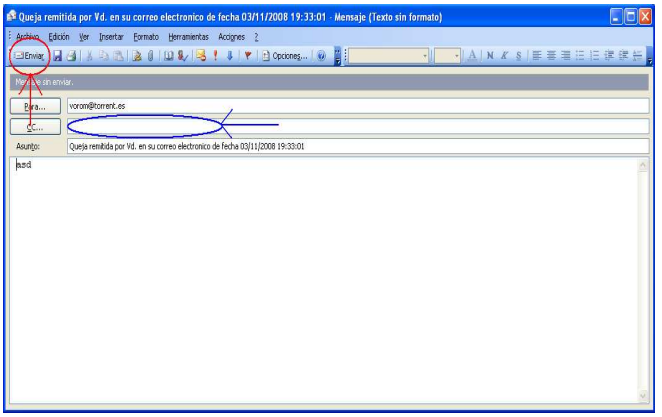
Desestimar queixa: Si la queja se ha cerrado por cualquier motivo sin contestar al ciudadano utilizaremos esta parte de la aplicación.

2.1 Contestar al ciudadano



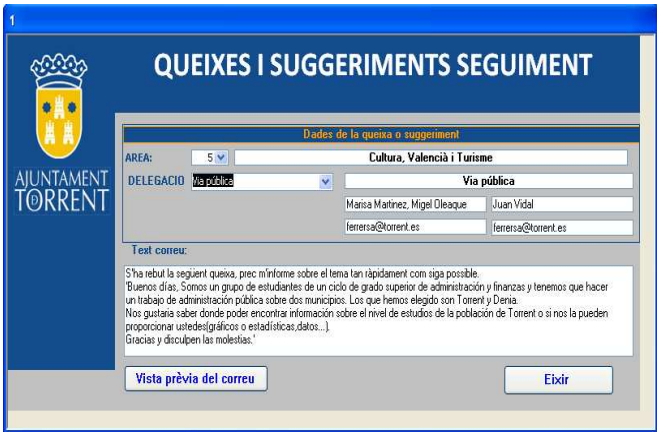
Primero habremos de catalogar de qué tipo es la queja. Seleccionaremos el área en el primer desplegable y el departamento concreto en el segundo. En los campos de la derecha podremos ir viendo la información del departamento: responsable, técnico, nombre…

Escribiremos la contestación que queremos darle al ciudadano en el recuadro ***Contestació*** y pulsaremos el botón ***Vista prèvia del correu***.



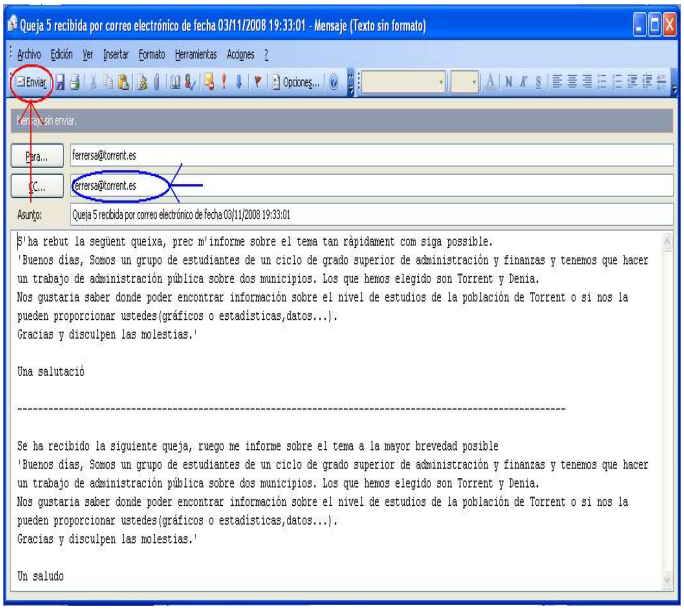
Se nos abrirá esta ventana en la que podremos añadir archivos adjuntos al correo, o poner varios destinatarios. Si queremos modificar el texto del mensaje deberemos cerrar esta ventana, volviendo a la ventana anterior y modificando el mensaje. En caso contrario la contestación archivada no se corresponderá a la contestación dada al ciudadano. Pulsaremos ***Enviar*** para contestar al ciudadano.

2.2 Pedir información



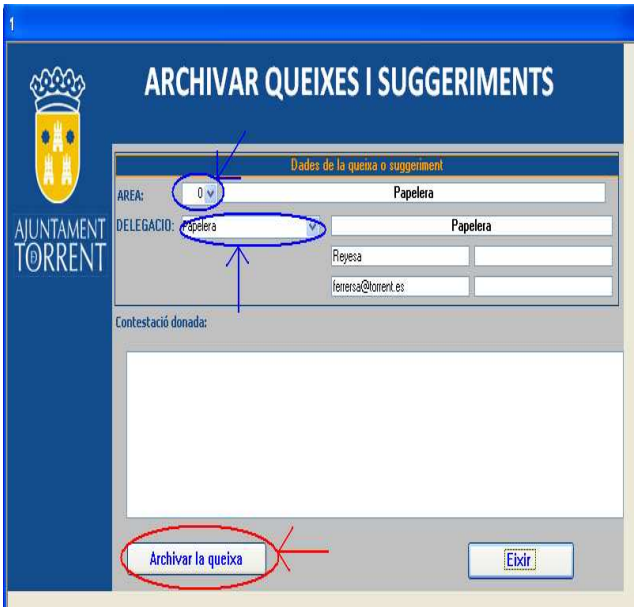
Primero habremos de catalogar de qué tipo es la queja. Seleccionaremos el área en el primer desplegable y el departamento concreto en el segundo. En los campos de la derecha podremos ir viendo la información del departamento: responsable, técnico, nombre…

Nos aparecerá un texto por defecto en valenciano y castellano para el departamento, en el que se incluirá la queja remitida por el ciudadano. Pulsaremos el botón ***Vista prèvia del correu***.



Se nos abrirá esta ventana en la que podremos añadir archivos adjuntos al correo, o poner varios destinatarios. Si queremos modificar el texto del mensaje deberemos cerrar esta ventana, volviendo a la ventana anterior y modificando el mensaje. En caso contrario la contestación archivada no se corresponderá a la contestación dada al ciudadano. Pulsaremos ***Enviar*** para contestar al ciudadano.

2.3 Archivar la queja



Primero habremos de catalogar de qué tipo es la queja. Seleccionaremos el área en el primer desplegable y el departamento concreto en el segundo. En los campos de la derecha podremos ir viendo la información del departamento: responsable, técnico, nombre…

Escribiremos la contestación que le dimos al ciudadano por teléfono o por otros medios y pulsaremos el botón ***Archivar la queixa***.

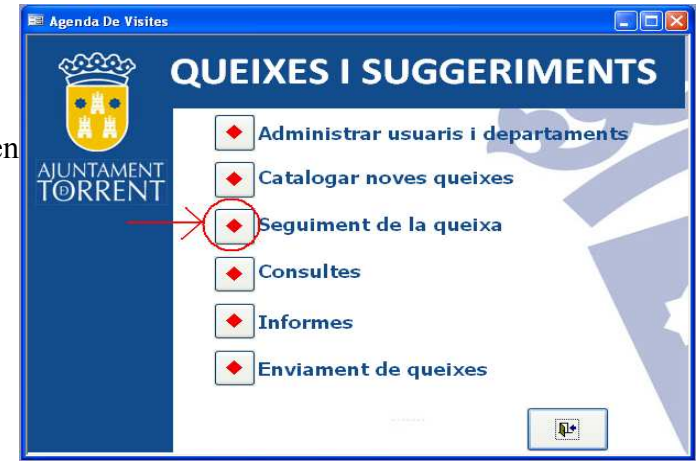
2.4 Desestimar queja

Primero habremos de catalogar de qué tipo es la queja. Seleccionaremos el área en el primer desplegable y el departamento concreto en el segundo. Para aquellas quejas que no tengan sentido y no pertenezcan a ningún departamento usaremos el área 0, el departamento Papelera. En los campos de la derecha podremos ir viendo la información del departamento: responsable, técnico, nombre…

Escribiremos el motivo por el que se desestima la queja y pulsaremos ***Tancar la queixa***

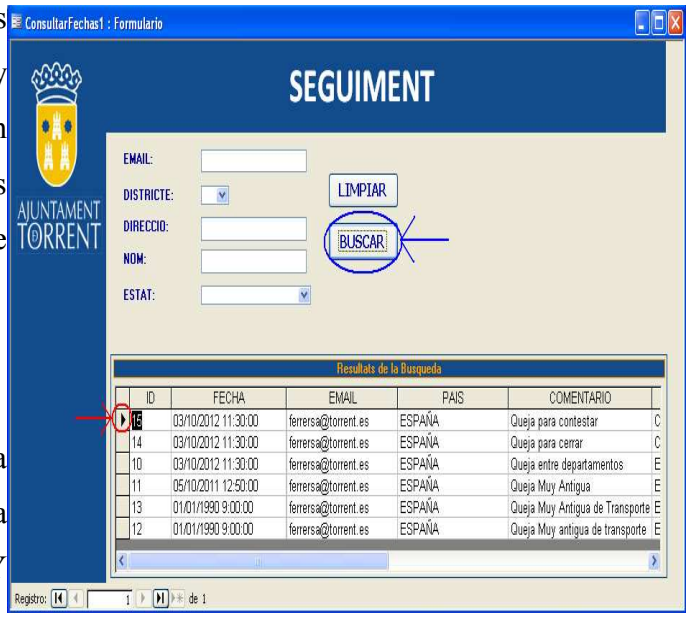
3. Seguimiento de quejas

En esta parte de la aplicación podremos hacer el seguimiento de las quejas. Podremos ver aquellas quejas que fueron enviadas a un departamento y reabrir quejas cerradas o contestadas. Dependiendo de los permisos de los que disponga nuestro usuario veremos las quejas o sugerencias normales o las quejas y sugerencias de alcaldía.



Pincharemos en el botón mostrado en la imagen para acceder al seguimiento de las quejas.

En esta ventana buscaremos la queja que nos interese. Para ello rellenaremos los campos y pulsaremos Buscar. Si todos los campos están vacíos se nos mostrará todas las quejas contestadas, cerradas o que estén actualmente enviadas a un departamento.

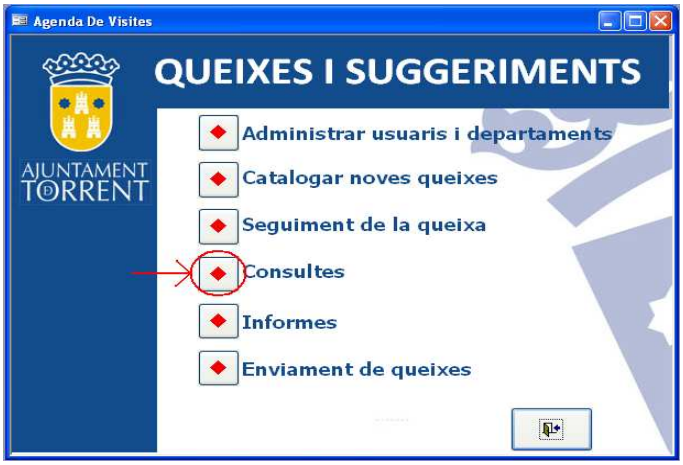


Seleccionado una de las quejas se nos abrirá la ventana de catalogó de quejas con la misma funcionalidad que en el apartado anterior. Y podremos hacer exactamente lo mismo que en los apartados 2.1, 2.2, 2.3 y 2.4.

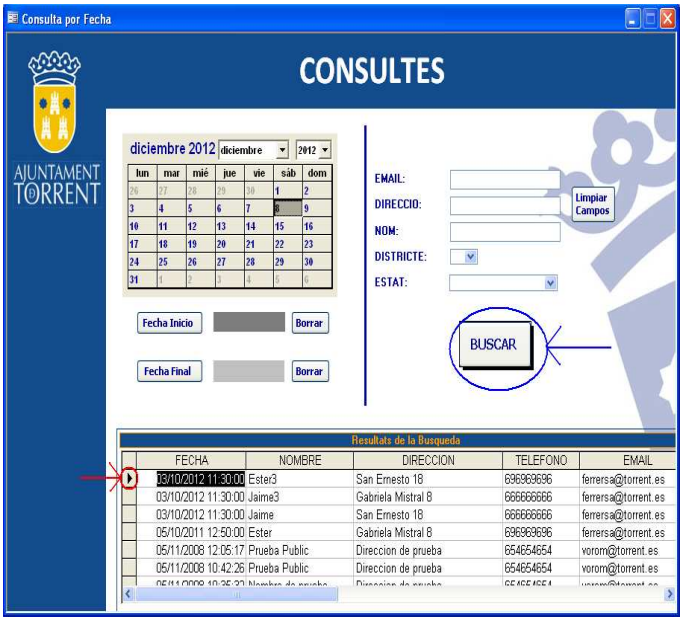
4. Consultas

Con esta parte de la aplicación podremos consultar la información de las diferentes quejas. Como siempre solo podremos haber aquellas quejas permitidas según nuestros permisos.

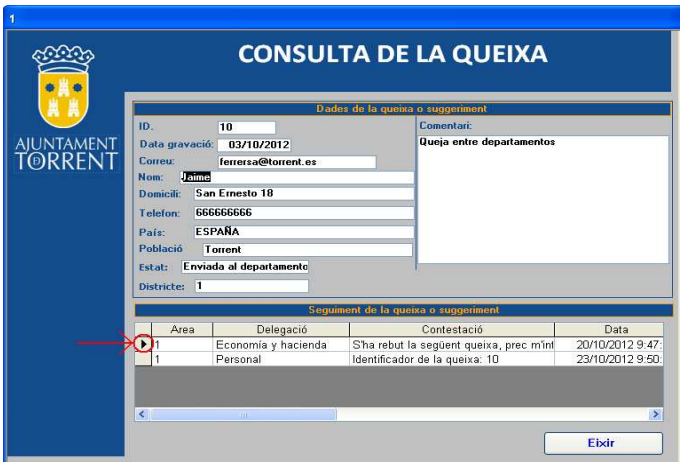
Pincharemos en el botón mostrado en la imagen para acceder al apartado de consultas.



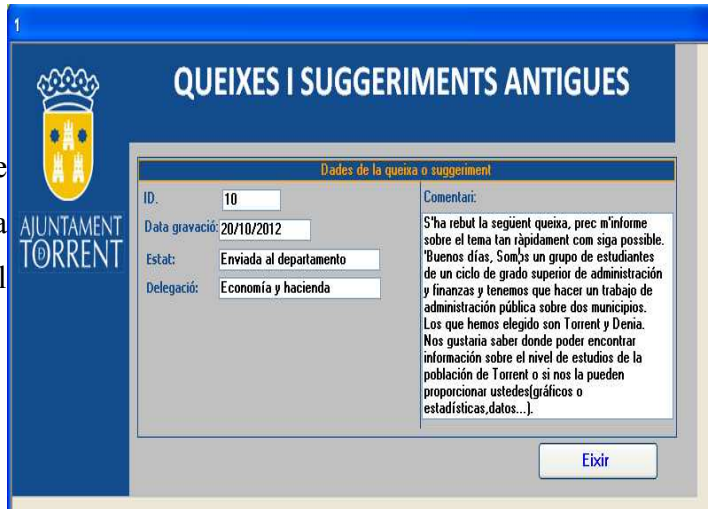
En esta ventana podremos buscar las quejas. Para ello rellenaremos los campos que necesitemos y pulsaremos el botón ***Buscar***. Nos aparecerán en la lista aquellas quejas que cumplas los criterios marcados. Para ver sus datos seleccionaremos una.



En esta ventana veremos los datos de la queja seleccionado, así como todas las etapas que ha pasado. Seleccionando una de las etapas podremos ver su información.



En esta ventana podremos ver el correo que le mandamos al ciudadano o al departamento. Si la fase que abrimos es “Cerrada” nos aparecerá el motivo por el que se cerró la queja.

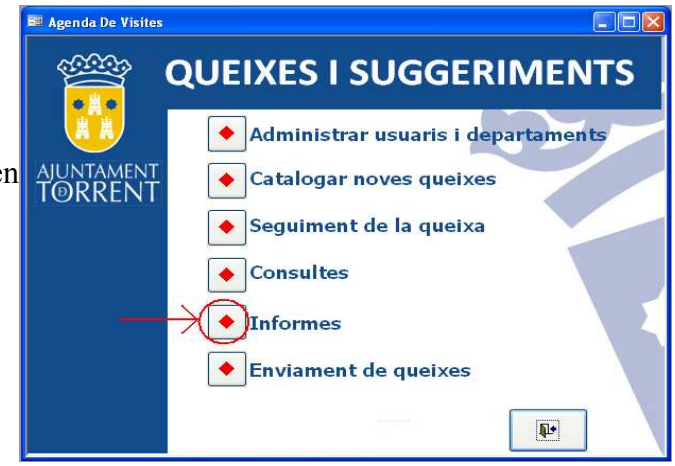


5. Informes

Con esta parte de la aplicación podremos generar informes. Una vez más solo se tendrán en cuenta para los informes aquellas quejas que podamos ver según nuestros permisos.

.

Pincharemos en el botón mostrado en la imagen para acceder al apartado de informes.



Pulsando en los botones adecuados se irán generados los distintos informes:

Nombre de queixes per usuari: Se generará un informe con los 20 usuarios que tienen más quejas.

Nombre de queixes per districte: Se nos abrirá una ventana en la que deberemos seleccionar un rango de fechas, y nos mostrara el número de quejas que se presentaron provenientes de cada distrito.

Nombre de queixes per mes: Se nos abrirá una ventana en la que deberemos seleccionar el año, y se generará un informe con el número de quejas que se presentaron cada mes.

Queixes urgents sense catalogar: Se generará un informe en el que veremos las quejas nuevas, que llevan más de una semana esperando a ser catalogadas.

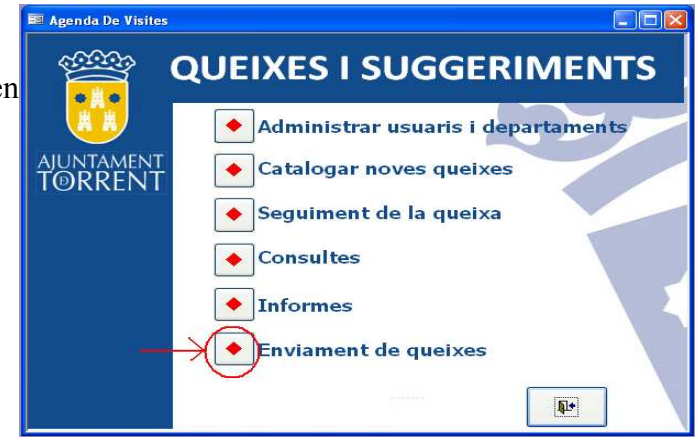
Queixes per departament pendents: Se generará un informe con aquellas quejas que fueron enviadas a cada departamento y llevan más de una semana esperando la respuesta de dicho departamento. Se presentará en un formato en el que cada departamento se generará en una hoja distinta.

Temps de desposta: Se abrirá una ventana en la que se nos pedirá un rango de fechas. Se generará un informe en el que se nos mostrará cuantos días tardó cada departamento en contestar las quejas que le fueron remitidas.

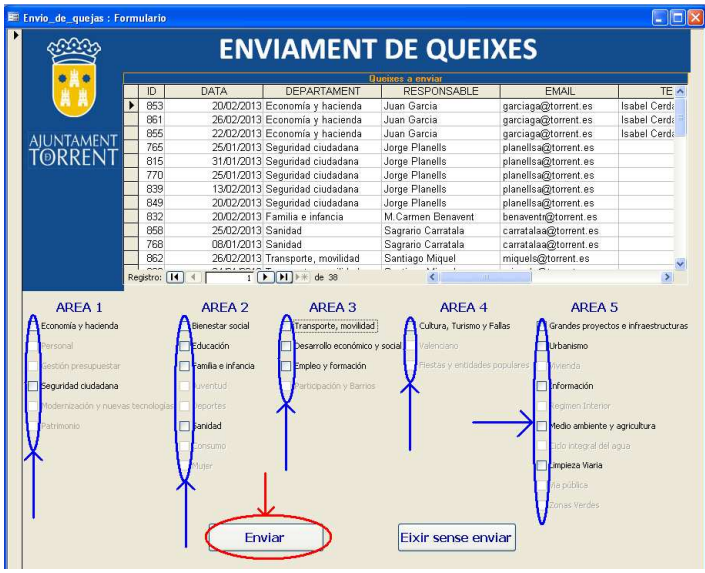
6. Envío de quejas

Únicamente el departamento de quejas puede hacer uso de esta parte de la aplicación. Usaremos esta funcionalidad para reenviar un correo aquellos departamentos que tienen quejas pendientes desde hace más de una semana. Adicionalmente se enviará un correo a la alcaldesa con un resumen de las quejas pendientes de cada departamento. El cambio de destinatario de este correo resumen esta explicado en el apartado 7.

Pincharemos en el botón mostrado en la imagen para acceder al apartado de envío de quejas.



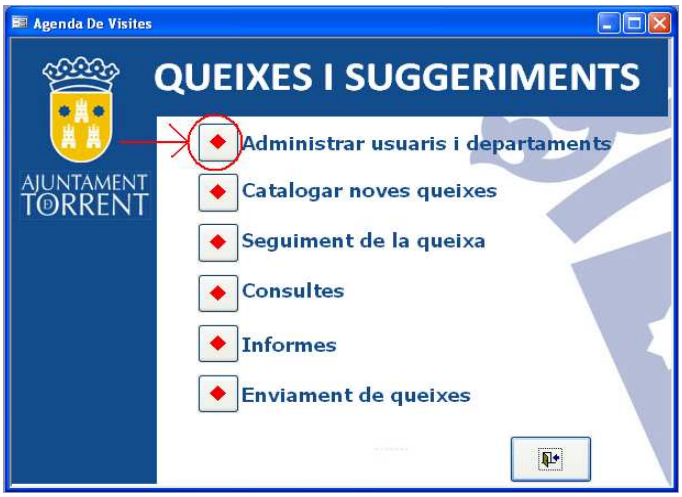
Aquí podremos ver las quejas que se enviarán a cada departamento. Estas quejas serán las mismas que aparecerán en el informe de "Queixes per departament pendents". Al pulsar en Enviar se nos mostrará los correos a enviar, por si queremos editarlos



7. Administración de usuarios y departamentos

Únicamente los usuarios encargados de administrar la aplicación pueden acceder a esta parte de la aplicación. En ella creamos y modificaremos los usuarios y departamentos de la aplicación.

Pincharemos en el botón mostrado en la imagen para acceder al apartado de administración.



7.1 Añadir usuario

Pincharemos en el botón mostrado para acceder al apartado para añadir usuarios. Pinchando en el botón marcado en azul volveremos a la ventana principal frontal de la aplicación.

En esta ventana introduciremos usuarios nuevos. Es INDISPENSABLE que el nombre del usuario sea IDENTICO al usuario de Windows. Posteriormente seleccionaremos los permisos que le queremos dar al usuario en el menú desplegable. En el cuadro de texto podremos ver una descripción de cada permiso. Pulsando en ***Afegir*** crearemos el usuario nuevo.

7.2 Modificar usuario

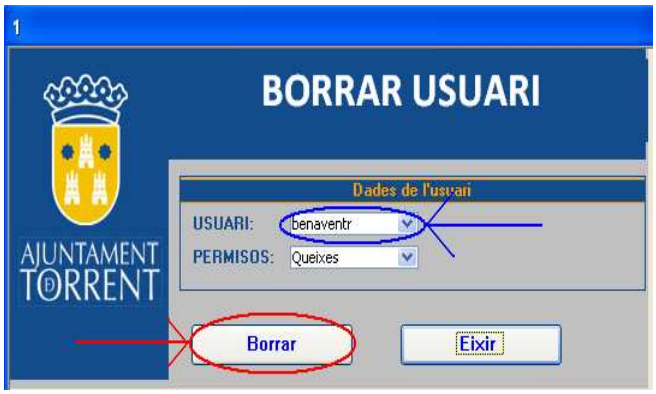
Pincharemos en el botón mostrado para acceder al apartado para modificar usuarios. Pinchando en el botón marcado en azul volveremos a la ventana principal frontal de la aplicación.

En el primer menú desplegable seleccionaremos el usuario al que queremos modificar los permisos. En el segundo menú seleccionaremos los nuevos permisos. Dándole a ***Modificar*** guardaremos los cambios.

7.3 Borrar usuario

Pincharemos en el botón mostrado para acceder al apartado para borrar usuarios. Pinchando en el botón marcado en azul volveremos a la ventana principal frontal de la aplicación.

En el primer menú desplegable seleccionaremos el usuario que queremos eliminar. Dándole a ***Borrar*** eliminaremos al usuario.



7.4 Añadir departamento

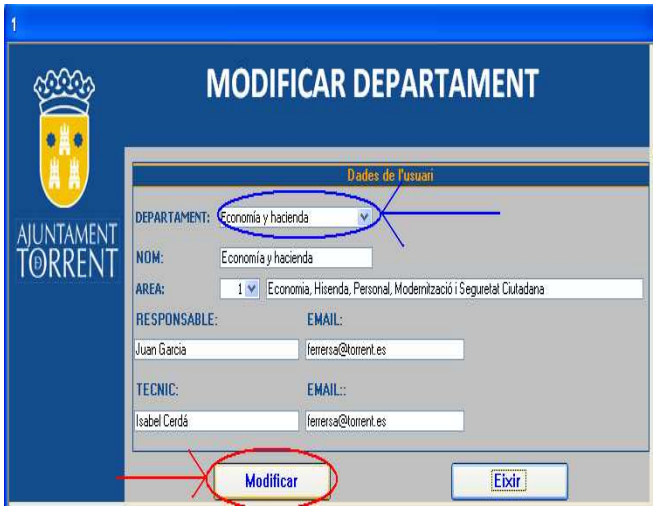
Pincharemos en el botón mostrado para acceder al apartado para añadir departamentos. Pinchando en el botón marcado en azul volveremos a la ventana principal frontal de la aplicación.

En esta ventana deberemos introducir los datos del nuevo departamento. Es muy importante que introduzcamos adecuadamente el nombre del responsable, respetando mayúsculas y acentos, así como el correo, pues estos datos se utilizarán en la generación del correo. Pulsando en *Afegir* añadiremos el nuevo departamento.

7.5 Modificar departamento

Pincharemos en el botón mostrado para acceder al apartado para modificar departamentos. Pinchando en el botón marcado en azul volveremos a la ventana principal frontal de la aplicación.

Seleccionaremos un departamento del menú desplegable y se nos mostrarán los datos actuales de dicho departamento, permitiéndonos modificarlos. Al pulsar en Modificar se guardarán los cambios.



7.6 Modificar destinatario correo resumen

Pincharemos en el botón mostrado para acceder al apartado para modificar los destinatarios de los correos resumen. Pinchando en el botón marcado en azul volveremos a la ventana principal frontal de la aplicación

Introduciremos el los destinatarios del correo en los campos marcados y pulsaremos ***Modificar***.