UNIVERSIDAD DE CÓRDOBA

Ingeniería del Software, Conocimiento y Bases de Datos

GRADO DE INGENIERÍA INFORMÁTICA

INGENIERÍA DEL SOFTWARE

SISTEMA DE GESTIÓN DE ALUMNOS

Autor/es:

Sergio Gómez Fernández

Eloy Gómez García

Ventura Lucena Martínez

Fecha:

23 de diciembre de 2018

Índice de contenidos

Índice de figura

1. Introducción

1.1. Definición del problema

El profesorado de la asignatura de Ingeniería del Software quiere informatizar los datos de contacto de los alumnos en un programa informático que guarde esta información. El profesorado, que dispondrá de una lista con un número variable de alumnos, dispondrá además de los datos necesarios para la introducción de un alumno en el sistema: Documento Nacional de Identidad (único), nombre, apellidos, dirección de residencia (único), correo asociado a la Universidad de Córdoba (único), curso más alto matriculado, teléfono de contacto, número de equipo al que pertenece y, por último, si es líder del grupo al cual pertenece.

Los alumnos almacenados en el sistema estarán especificados en una lista de alumnos, representados por los datos identificativos mencionados anteriormente.

1.2. Identificación del problema técnico

Una vez identificado el problema, se procederá a acotar los puntos que se deben resolver en la solución.

- **Clientes**
    - Persona que hará uso de los servicios del sistema para la gestión de alumnos de la asignatura Ingeniería del Software. Dispondrá de un nombre de usuario y una clave de autenticación que le permitirán el acceso al sistema.
- **Acceso a contenidos**
    - En el sistema se destacarán los contenidos accesibles por el cliente, reducidos a la información de los alumnos introducidos en el sistema.

    - En dicho apartado, el cliente será capaz tanto de añadir a cualquier nuevo alumno que se presente, como su modificación, muestra, búsqueda o eliminación; y en ámbitos del sistema, la carga del documento de información, como su guardado.

- **Registro de credenciales de usuario**
    - Datos que hacen referencia a la autentificación de acceso del cliente al sistema.
    - Los usuarios se registrarán mediante una petición de datos, en la cual se destacan el nombre de usuario y la contraseña necesaria para acceder al sistema con dicho usuario.
    - Estos datos se guardarán en un fichero binario.
- **Servicios**
    - Carga del fichero de almacenamiento de información.
    - Capacidad de añadir a un nuevo alumno al sistema.
    - Búsqueda de un alumno.
    - Modificación de los datos de un alumno.
    - Muestra de un alumno. Será posible la muestra de todos los alumnos almacenados en el sistema.
    - Eliminación de un alumno. Será posible, al igual que el servicio anterior, la eliminación de todos los alumnos almacenados en el sistema.
    - Guardado de los datos en el fichero de almacenamiento de información.

1.3. Recursos

1.3.1. Recursos hardware

Equipos proporcionados por la Escuela Politécnica Superior de Córdoba y equipos personales.

1.3.2. Recursos software

En esta sección se detallarán los recursos software utilizados tanto para el desarrollo del sistema como para su ejecución:

- En la ejecución se utilizará un sistema operativo Ubuntu versión 18.04 o cualquier derivado. Las versiones anteriores son aptas para la ejecución del sistema.
- En el desarrollo se han utilizado tanto los equipos proporcionados por la Escuela Politécnica Superior de Córdoba como equipos personales con las siguientes herramientas:
    - Sublime Text 3
    - Visual Paradigm v15.1
    - GitHub

2. Especificación de requisitos

En esta fase se desarrollarán los requisitos del sistema gracias a la formalización del problema realizada mediante entrevistas con el cliente para el desarrollo del sistema para gestionar los datos de contacto de alumnos, con el fin de que este sea óptimo para su uso.

2.1. Requisitos funcionales

- **RF-1**: El sistema permite agregar al sistema un nuevo alumno mediante la función "Añadir". Dicha función añadirá a partir de un parámetro principal, el cual será el Documento Nacional de Identidad del alumno que se quiere añadir.
- **RF-1.2**: El sistema no permitirá añadir alumnos cuando se llegue a 150 alumnos.
- **RF-1.3**: El sistema no permitirá añadir alumnos con Documentos Nacionales de Identidad idénticos.
- **RF-2**: El sistema tiene una funcionalidad que permite buscar a un alumno concreto en el sistema, identificado por su Documento Nacional de Identidad o por su apellido.
- **RF-3**: El sistema tiene una funcionalidad que permite modificar a un alumno a través de su Documento Nacional de Identidad o su apellido.
- **RF-3.1**: Si se hace uso de la funcionalidad de modificación a través del apellido y este está repetido, el sistema pedirá el apellido del alumno.
- **RF-3.2**: El sistema no permitirá hacer uso de la funcionalidad de modificación a través del Documento Nacional de Identidad si este ya se encuentra en el sistema.

- **RF-4**: El sistema tiene una funcionalidad que permite la eliminación de un alumno a través de su Documento Nacional de Identidad; o la eliminación de todos los alumnos del sistema.
- **RF-5**: El sistema tiene una funcionalidad que permite mostrar la información de los alumnos de manera ascendente.
- **RF-6**: El sistema tiene una funcionalidad que permite cargar el fichero en formato binario donde se almacena la información de cualquier lista de alumnos.
- **RF-7**: El sistema tiene una funcionalidad que permite guardar la información introducida en un fichero en formato binario.

2.2. Requisitos no funcionales

- **RNF-1**: El identificador principal del alumno será el Documento Nacional de Identidad.
- **RNF-2**: Un grupo no puede tener más de un líder.
- **RNF-3**: Se podrá buscar a un alumno tanto por su Documento Nacional de Identidad como por apellido; y se podrá modificar por su Documento Nacional de Identidad.
- **RNF-4**: El Documento Nacional de Identidad y el correo son únicos para cada alumno.
- **RNF-5**: El listado de alumnos debe mostrarse ordenado alfabéticamente en el caso de ser por apellidos o nombre, y por orden numérico en el caso del Documento Nacional de Identidad, ya sea de forma ascendente o descendente.
- **RNF-6**: El número máximo de alumnos es 150.
- **RNF-7**: Los datos guardados y cargados deben mostrarse en un fichero binario.

- **RNF-8**: Todos los campos a rellenar en la función "Añadir" son obligatorios a excepción del grupo y el líder.
- **RNF-9**: El sistema guardará cuando se termine una operación en el fichero binario.
- **RNF-10**: Interfaz a través de línea de comandos.
- **RNF-11**: Sistema operativo:
    - Principal: Linux.
    - Opcional: Multiplataforma.
- **RNF-12**: Lenguaje de programación: C++.
- **RNF-13**: Lenguaje de documentación: Markdown.
- **RNF-14**: En caso de hacer uso de un IDE, será Eclipse.

2.3. Historias de usuario

ANVERSO

ID: 001 **Añadir alumnos.**

Como usuario, quiero poder añadir un alumno en el sistema.

Prioridad: 3

REVERSO

- Quiero poder introducir los datos de un nuevo alumno.
- Quiero que me indique si se ha producido algún error al introducir el usuario.
- Quiero que me guarde los datos del alumno en el fichero binario.

ANVERSO

ID: 002 **Buscar.**

Como usuario, quiero buscar a los alumnos, ya sea a través del DNI o del apellido.

Prioridad: 2

REVERSO

- Quiero poder buscar a cada alumno por DNI, apellidos o grupo.

ANVERSO

ID: 003 **Modificar alumno.**

Como usuario, quiero poder modificar los datos que aparezcan en el sistema.

Prioridad: 2

REVERSO

- Los datos a modificar podré buscarlos por DNI lo cual me dejará modificarlos directamente; o por apellidos, pero si está repetido pedirá el DNI.

ANVERSO

ID: 004 **Eliminar alumno.**

Como usuario quiero poder eliminar alumnos.

Prioridad: 3

REVERSO

- Quiero poder eliminar los datos de un alumno de mi sistema.
- Se debe mostrar si el alumno introducido era líder del grupo y notificar que si grupo ya no tiene líder.
- Se debe actualizar el fichero binario.

ANVERSO

ID: 005 **Mostrar alumno.**

Como usuario quiero visualizar el contenido.

Prioridad: 1

REVERSO

- Podrá mostrar a un alumno, ya sea por apellido o DNI.
- Podrá mostrar todos alfabéticamente por apellido.
- Podrá ordenarlos de forma ascendente por DNI.

ANVERSO

ID: 006 **Cargar.**

Como usuario, quiero poder cargar los datos del sistema de la base de datos.

Prioridad: 1

REVERSO

- Quiero recuperar los datos guardados con anterioridad.

ANVERSO

ID: 007 **Guardar.**

Como usuario, quiero poder guardar los datos de la base de datos tras haber realizado alguna modificación. Dichos datos se guardarán en un fichero binario.

Prioridad: 1

REVERSO

- Quiero poder guardar todos los datos.
- Los datos quiero guardarlos en un fichero binario.

ANVERSO

ID: 008 **Login.**

Como usuario, quiero poder acceder al sistema.

Prioridad: 1

REVERSO

- Quiero poder tener acceso al sistema.
- En caso de que me equivoque al introducir los datos, quiero que me los pida nuevamente.

3. Análisis

3.1. Análisis funcional

Una vez se conoce el problema real que se va a resolver, es necesario llevar a cabo una especificación más técnica del problema.

3.1.1. Caso de Uso general

En la figura 3.1 se muestra el caso de uso general del sistema de gestión de alumnos en que identificamos diferentes subsistemas con los que interaccionará el actor en los diferentes casos de uso.

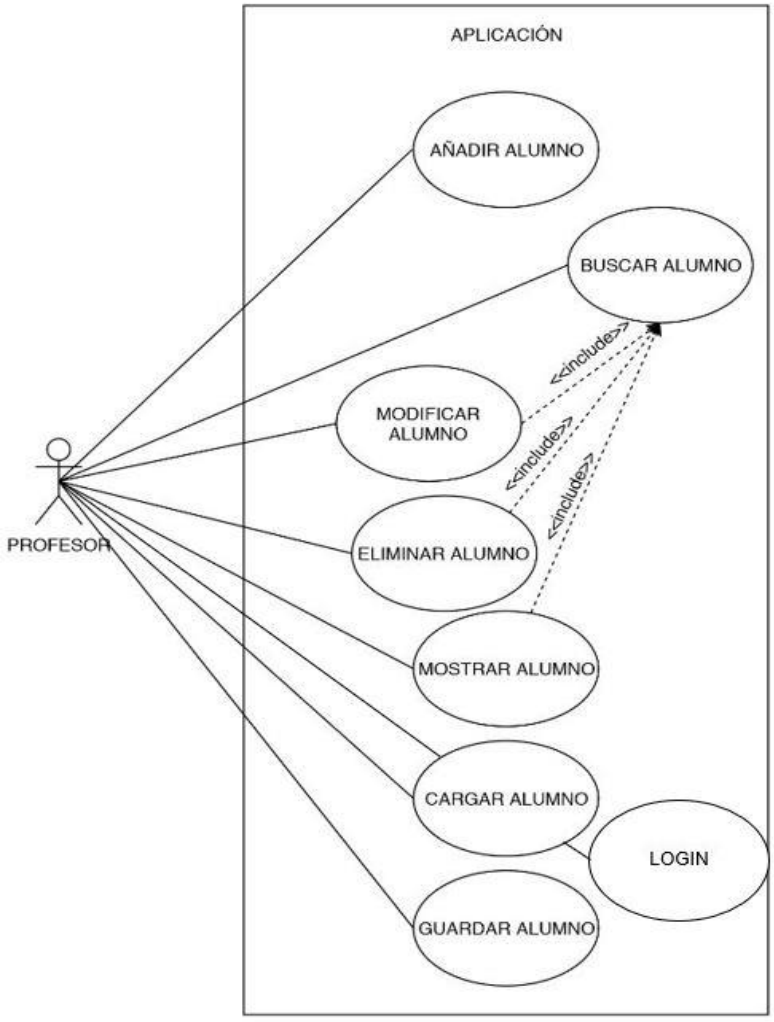

Figura 3.1

En este diagrama destacamos a un único actor[1], "Profesor", que realiza la función de cliente, que es un tipo de usuario que accede al sistema con credenciales de usuario y contraseña previamente almacenados en el sistema, el cual tiene ciertos privilegios dentro del mismo.

[1] Un actor representa un rol que interpreta una persona o periférico en nuestro sistema. El actor se relacionará con los casos de uso mediante mensajes.

3.1.2. Casos de Uso

Añadir alumno

ID: 001

Breve descripción: El sistema permite añadir a un alumno.

Actores principales: Usuario/Profesor.

Actores secundarios: Alumnos.

Precondiciones:

1. El usuario debe tener datos que añadir de un alumno.

Flujo principal:

1. El caso de uso empieza cuando el usuario quiere introducir un nuevo alumno.
2. El sistema guarda los datos del nuevo alumno en el fichero binario.

Postcondiciones:

- Queda almacenada la información del nuevo alumno.

Flujos alternativos:

1.a. Si introducimos un alumno categorizado como "líder" en un grupo que ya tiene un líder, el sistema mostrará un mensaje de error y redireccionará al usuario al menú principal o de nuevo a la función si se desea.

1.b. Si introducimos un nuevo alumno cuando ya hay 150 alumnos, el sistema mostrará un mensaje de error, y redireccionará al usuario al menú principal.

2.a. Si el usuario no introduce nada en los campos equipo y líder, el sistema no mostrará ningún error, ya que esos campos son opcionales, por lo que redireccionará al usuario al menú principal o de nuevo a la función si se desea.

Buscar

ID: 002

Breve descripción: El sistema buscará datos.

Actores principales: Usuario/Profesor.

Actores secundarios: Alumnos.

Precondiciones:

1. El sistema debe tener datos en los que buscar.
2. El alumno debe existir en el sistema.

Flujo principal:

1. El caso de uso empieza cuando el usuario vaya a buscar a un alumno.
2. El sistema busca los datos del alumno.

Postcondiciones:

- El sistema muestra un mensaje de éxito al encontrar en el sistema al alumno.

Flujos alternativos:

1.a. Si no hay datos en el sistema, el mismo mostrará un mensaje de error, y redireccionará al usuario al menú principal.

2.a. Si no existe el alumno, el sistema mostrará un mensaje de error, y preguntará al usuario si quiere volver al menú principal o quiere buscar a otro alumno.

Modificar alumno

ID: 003

Breve descripción: El sistema permite modificar los alumnos.

Actores principales: Usuario/Profesor.

Actores secundarios: Alumnos.

Precondiciones:

1. El usuario tiene que tener datos que modificar.

Flujo principal:

1. El caso de uso empieza cuando el usuario quiere modificar un alumno.
2. El sistema muestra los datos actuales del alumno.
3. El usuario modifica los datos.

Postcondiciones:

- El sistema muestra un mensaje de éxito con los nuevos datos si se han modificado correctamente.

Flujos alternativos:

3.a. Si se quiere buscar por apellidos, y los apellidos introducidos son únicos, el sistema permitirá modificar los datos. En caso de que coincidan, se pedirá el DNI.

3.b. Si el DNI que se desea modificar coincide con el de otro alumno, el sistema impedirá que se guarden esos datos mostrando un mensaje de error y pedirá unos datos nuevos.

3.c. Si el alumno que se desea modificar es el líder, el sistema mostrará un mensaje detallando que el alumno que se está modificando es el líder.

3.d. Si un alumno que no era líder se modifica para que lo sea, y en ese grupo ya hay un líder, se mostrará un mensaje indicando si quiere que se sustituya el líder actual.

Eliminar alumno

ID: 004

Breve descripción: El sistema permite eliminar a un alumno.

Actores principales: Usuario/Profesor.

Actores secundarios: Alumnos.

Precondiciones:

1. El usuario tiene que tener a un alumno que eliminar.

Flujo principal:

1. El caso de uso empieza cuando el usuario quiere eliminar un alumno.
2. El sistema elimina al usuario deseado de la base de datos.

Postcondiciones:

- El sistema muestra un mensaje de éxito al eliminar del sistema al alumno.

Flujos alternativos:

1.a. Si se desea eliminar por apellido y hay varios alumnos con el mismo apellido, el sistema automáticamente pedirá el DNI del alumno que se desea eliminar.

2.a. Si el alumno que se desea eliminar es líder de un grupo, el sistema mostrará un mensaje de advertencia indicando que el grupo al que pertenecía ya no tiene líder, pero no dará ningún error.

Mostrar alumno

ID: 005

Breve descripción: El sistema permite mostrar los alumnos registrados.

Actores principales: Usuario/Profesor.

Actores secundarios: Alumnos.

Precondiciones:

1. El usuario debe tener alumnos que mostrar.

Flujo principal:

1. El caso de uso empieza cuando el usuario quiera mostrarlos alumnos.
2. El sistema muestra los datos de los alumnos.

Postcondiciones:

- El sistema muestra una lista con los datos de los alumnos.

Flujos alternativos:

1.a. Si los datos que se muestran corresponden a todos los alumnos, estos podrán ser ordenados de forma ascendente o descendente, ya sea por apellidos o por DNI.

Cargar

ID: 006

Breve descripción: El sistema cargará los datos.

Actores principales: Usuario/Profesor.

Actores secundarios: Alumnos.

Precondiciones:

1. El sistema debe tener datos que cargar.
2. Los datos del sistema se cargarán a partir de un fichero binario.

Flujo principal:

1. El caso de uso empieza cuando el usuario vaya a cargar los datos de la agenda.
2. El sistema carga los datos de la agenda.

Postcondiciones:

- El sistema muestra un mensaje de éxito.

Flujos alternativos:

1.a. Si no hay datos en el sistema, el mismo mostrará un mensaje de error, y redireccionará al usuario al menú principal.

Guardar

ID: 007

Breve descripción: El sistema guardará datos.

Actores principales: Usuario/Profesor.

Actores secundarios: Alumnos.

Precondiciones:

1. El sistema debe tener datos que guardar.

Flujo principal:

1. El caso de uso empieza cuando el usuario vaya a guardar datos.

Postcondiciones:

- El sistema almacena los datos guardados en un fichero binario y muestra un mensaje de éxito.

Flujos alternativos:

1.a. Si no hay datos que guardar en el sistema, el mismo creará un fichero vacío con el nombre que indique el usuario.

1.b. Si el nombre que se ha introducido para el fichero coincide con uno ya existente, se sobrescribirán los datos.

Login

ID: 008

Breve descripción: El sistema permite o no la entrada al mismo.

Actores principales: Usuario/Profesor.

Actores secundarios: -

Precondiciones:

1. El sistema debe tener cargados los datos de credenciales del usuario.

Flujo principal:

1. El caso de uso empieza cuando el usuario vaya a acceder al sistema.

Postcondiciones:

- El sistema permite el acceso en caso de que las credenciales introducidas sean las correctas.

Flujos alternativos:

1.a. Si las credenciales no coinciden, el sistema pide nuevamente que se introduzcan las mismas.

3.2. Análisis de la información

3.2.1. Especificación de clases

A continuación, se presenta el modelo de clases que define el comportamiento del sistema, describiendo el conjunto de clases que conforman el análisis del sistema, los atributos que forman parte de cada una de ellas y los métodos de los que disponen.

3.2.1.1. Clase Profesor

Nombre: Profesor.

Descripción: clase que representa al usuario, que en este caso siempre será un profesor, perteneciente al sistema.

Atributos:

- rol: cadena de caracteres que especifica el tipo de usuario dentro del sistema. En este caso siempre será un profesor.
- nombre: cadena de caracteres que almacena el nombre de usuario dentro del sistema.
- Puntero a agenda*: toda función que realiza el profesor recae sobre la clase Agenda.

Métodos:

- cargarBD(); *boolean*: carga los datos en el sistema.
- guardarDB(); *boolean*: guarda la información.
- cargarSC(); *boolean*: carga una copia de seguridad.
- guardarSC(); *boolean*: guarda una copia de seguridad.
- setRol(); *void*: establece el rol del usuario.
- login(); *boolean*: permite el acceso al sistema.

La representación gráfica en UML de la clase Profesor se muestra en la figura 3.2.

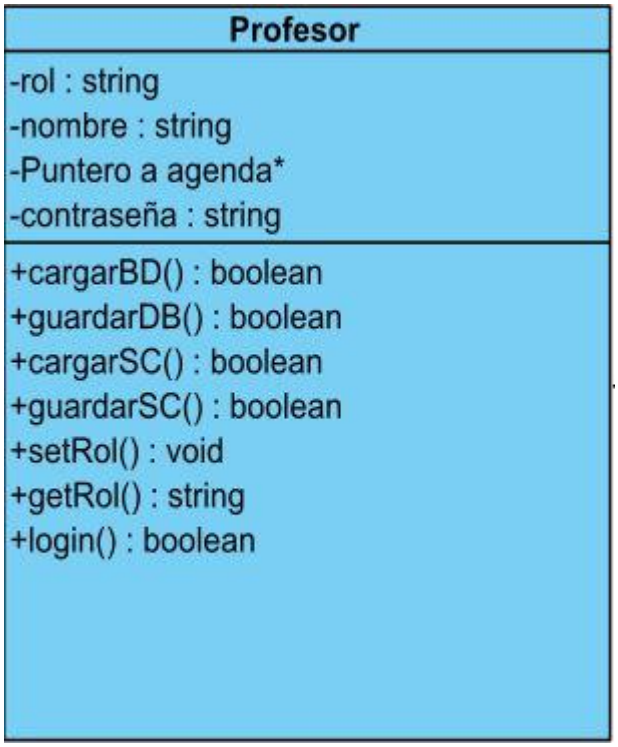

Figura 3.2

3.2.1.2. Clase Agenda

Nombre: Agenda.

Descripción: clase que representa el acceso al "almacén" de información.

Atributos:

- Lista alumnos: objeto del tipo *lista* donde se almacenan todos los alumnos.

Métodos:

- añadir(); *boolean*: permite añadir un alumno al sistema.

- buscar(string DNI/apellidos); *boolean*: permite buscar alumnos.
- modificar(); *boolean*: permite modificar un alumno.
- eliminar(); *boolean*: permite eliminar alumnos.
- mostrar(); *void*: muestra alumnos.

La representación gráfica en UML de la clase Agenda se muestra en la figura 3.3.

Figura 3.3

3.2.1.3. Clase Alumno

Nombre: Alumno

Descripción: clase que representa a un alumno cualquiera.

Atributos:

- nombre: cadena de caracteres que especifica el nombre del alumno.
- apellidos: cadena de caracteres que especifica los apellidos del alumno.
- dni: cadena de caracteres que especifica el Documento Nacional de Identidad del alumno.

- telefono: cadena de caracteres que especifica el teléfono del alumno.
- dirección: cadena de caracteres que especifica la dirección del alumno.
- email: cadena de caracteres que especifica la dirección de e-mail del alumno.
- curso: cadena de caracteres que especifica el curso más alto matriculado del alumno.
- equipo: variable de tipo entero que indica el equipo del alumno.
- líder: booleano que especifica si es o no líder.

Métodos:

- setNombre(string nombre); *void*: función que establece el nombre del alumno.
- getNombre(); *string*: función que devuelve el nombre del alumno.
- setApellidos(string apellidos); *void*: función que establece los apellidos del alumno.
- getApellidos(); *string*: función que devuelve los apellidos del alumno.
- setDNI(string DNI); *void*: función que establece el DNI del alumno.
- getDNI(); *string*: función que devuelve el DNI del alumno.
- setDireccion(string direccion); *void*: función que establece la dirección del alumno.
- getDireccion(); *string*: función que devuelve la dirección del alumno.

- setCorreo(string correo); *void*: función que establece el correo del alumno.
- getCorreo(); *string*: función que devuelve el correo del alumno.
- setTelefono(string telefono); *void*: función que establece el teléfono del alumno.
- getTelefono(); *string*: función que devuelve el teléfono del alumno.
- setEquipo(int equipo); *void*: función que establece el equipo del alumno.
- getEquipo(); *int*: función que devuelve el equipo en que el alumno se encuentra.
- setCurso(string curso); *void*: función que establece el curso del alumno.
- getCurso(); string: función que devuelve el curso en que el alumno se encuentra matriculado.
- setLider(bool lider); *void*: función que establece si el alumno es líder o no.
- getLider(); *bool*: devuelve 1 si es líder o 0 si no lo es.

La representación gráfica en UML de la clase Profesor se muestra en la figura 3.4.

Figura 3.4

3.2.2. Relaciones entre clases

3.2.2.1. Relación Profesor-Agenda

Esta relación une las clases Profesor y Agenda. Representa la utilización de una agenda por un profesor. La relación es uno a muchos: la clase Profesor participa con la cardinalidad mínima 1 y máxima * (n^2), mientras que la clase Agenda participa con la cardinalidad mínima y máxima 1.

[2] n equivale a la representación enésima de una cantidad.

La representación gráfica en UML de esta asociación queda reflejada en la figura 3.5.

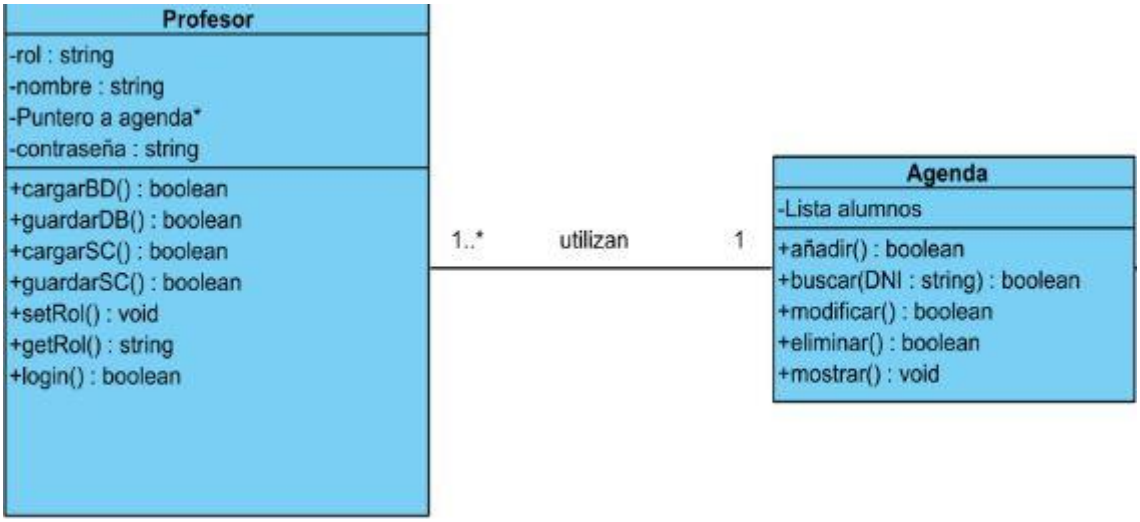

Figura 3.5

3.2.2.2. Relación Alumno-Agenda

Esta relación une las clases Agenda y Alumno. Representa la pertenencia de un alumno a una agenda. La relación en uno a muchos: la clase Alumno participa con la cardinalidad mínima y máxima 1, mientras que la clase Agenda participa con la cardinalidad mínima 1 y máxima * (n).

La representación gráfica en UML de esta composición queda reflejada en la figura 3.6.

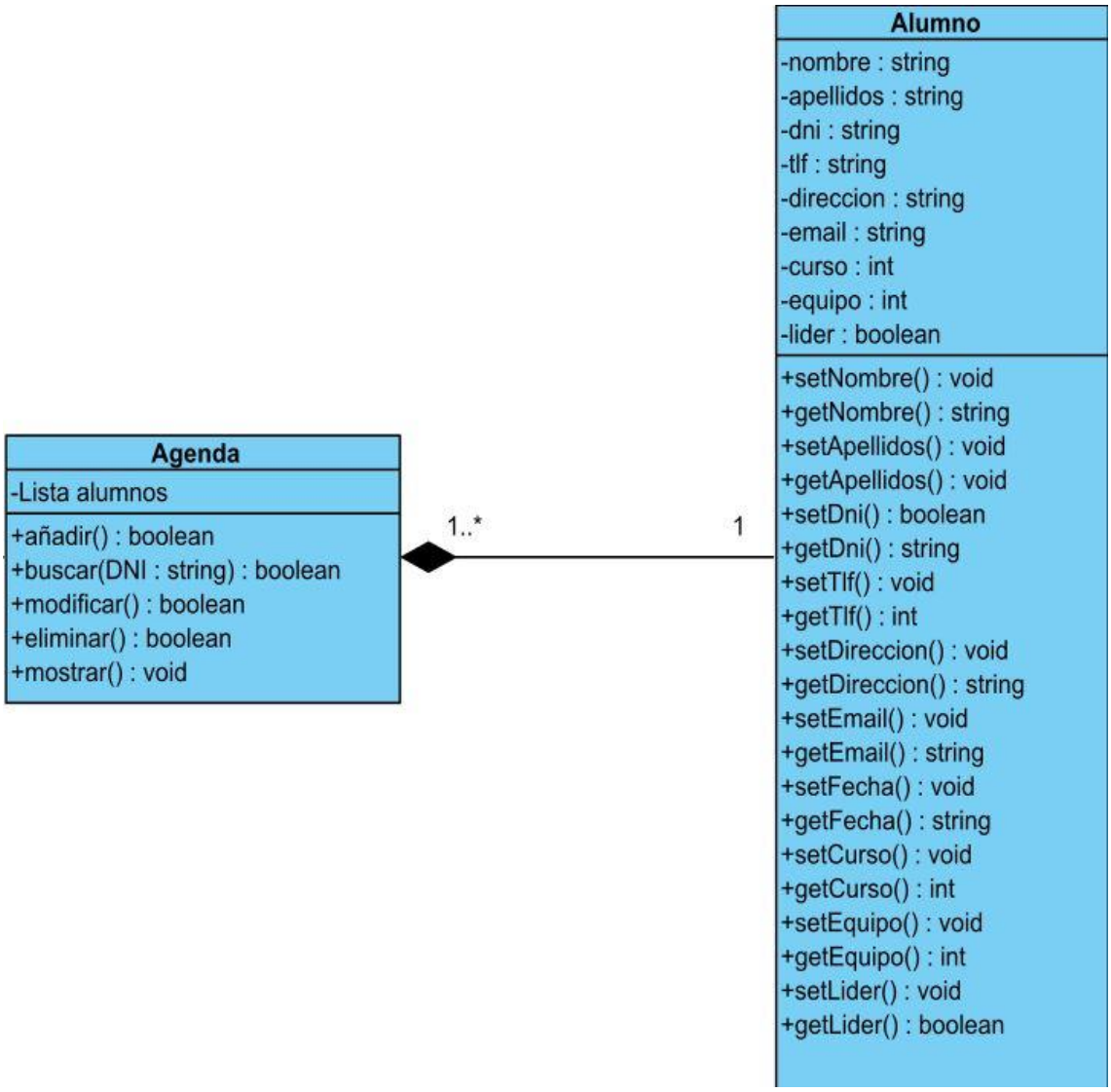

Figura 3.6

3.2.3. Diagrama de clases

En la figura 3.7 se muestra el diagrama de clases del sistema, obtenido mediante la unión de todas las relaciones existentes entre las clases consideradas para el dominio del problema.

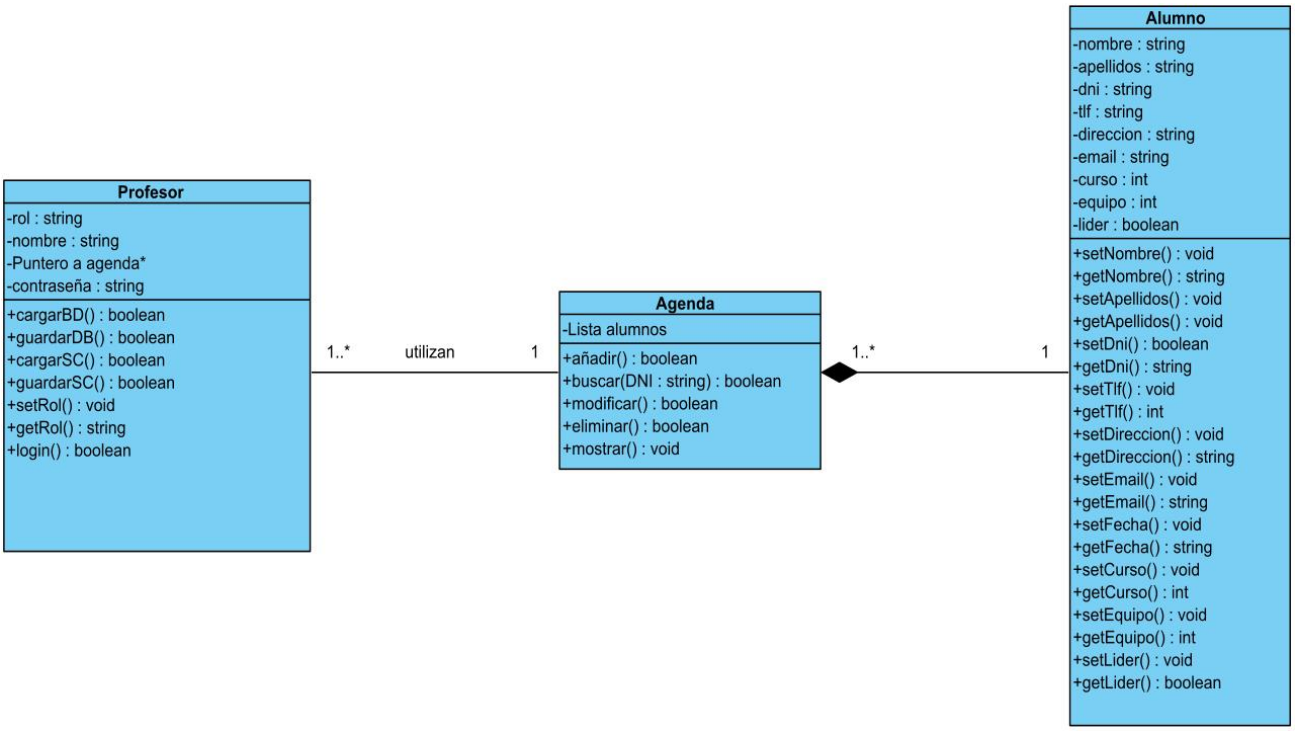

Figura 3.7

3.3. Análisis del comportamiento

Para comenzar a modelar el comportamiento del sistema, se van a construir una serie de Diagramas de secuencia.

Un diagrama de secuencia es un diagrama de interacción que destaca la ordenación temporal de los mensajes. Gráficamente, un diagrama de secuencia es una tabla que representa objetos, dispuestos a lo largo del eje X, y mensajes, ordenados según se suceden en el tiempo, a lo largo del eje Y.

3.3.1. Diagrama de secuencia: Añadir alumno

Este diagrama de secuencia, mostrado en la figura 3.8, describe el flujo de eventos existentes en el caso de añadir a un nuevo alumno al sistema.

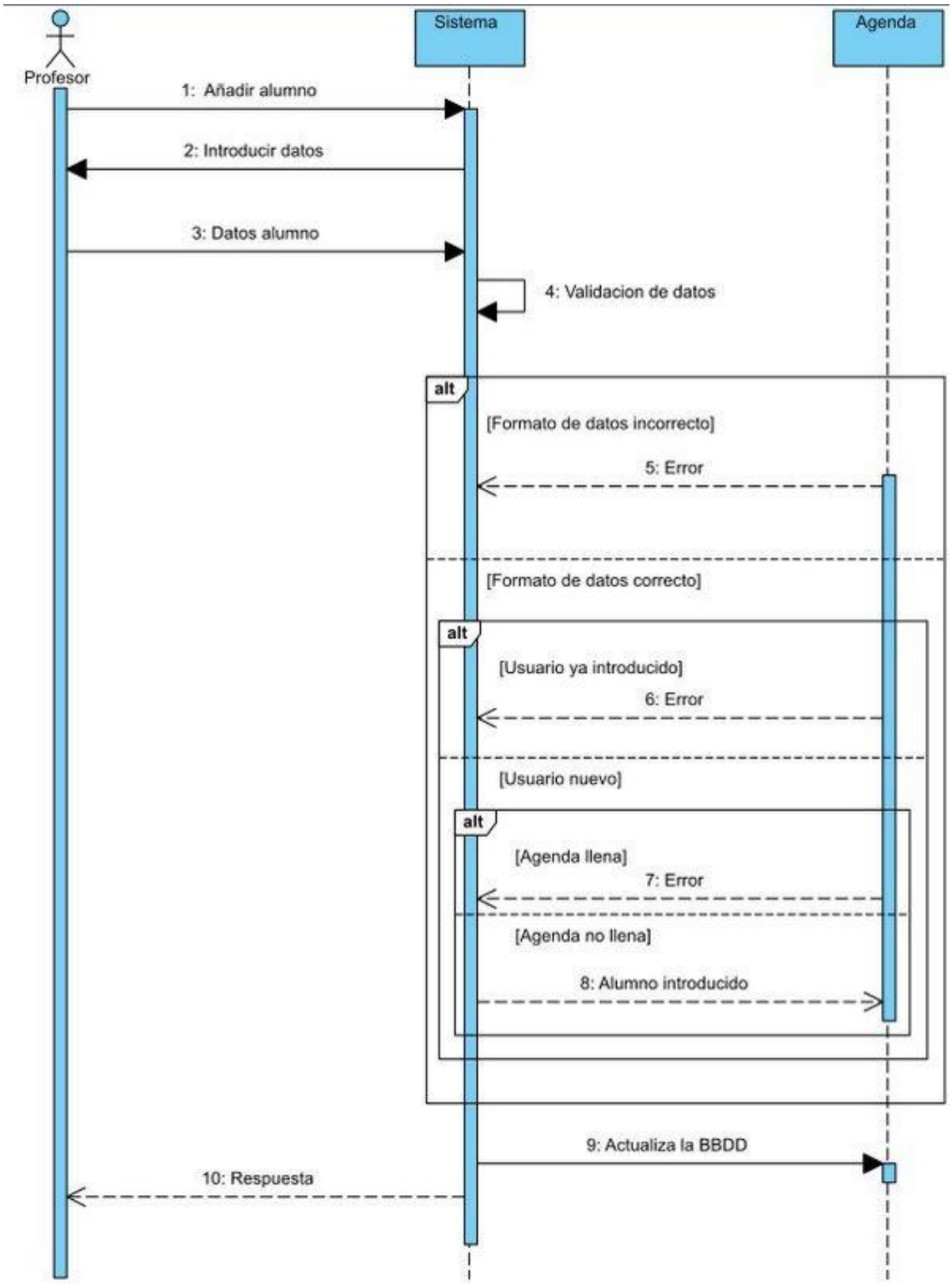
Profesor
Sistema
Agenda
1: Añadir alumno
2: Introducir datos
3: Datos alumno
4: Validacion de datos
alt
[Formato de datos incorrecto]
5: Error
[Formato de datos correcto]
alt
[Usuario ya introducido]
6: Error
[Usuario nuevo]
alt
[Agenda llena]
7: Error
[Agenda no llena]
8: Alumno introducido
9: Actualiza la BBDD
10: Respuesta

Figura 3.8

El usuario, al seleccionar la opción de añadir un nuevo alumno, recibe la respuesta del sistema para introducir los datos. Una vez introducidos, el sistema verifica que los datos introducidos sean válidos para el mismo. Dependiendo de su validez, se recibirá un mensaje de error si estos no son válidos, un mensaje de error si el alumno ya existe en el sistema (se traduce como la existencia del mismo DNI en la agenda), un mensaje de error si la agenda está llena y, por último, si no está llena y cumple los requisitos de validez, la información será apta para entrar en la agenda. Tras actualizarse la agenda, devuelve un mensaje de éxito.

3.3.2. Diagrama de secuencia: Buscar

Este diagrama de secuencia, mostrado en la figura 3.9, describe el flujo de eventos existentes en el caso de buscar a un alumno al sistema.

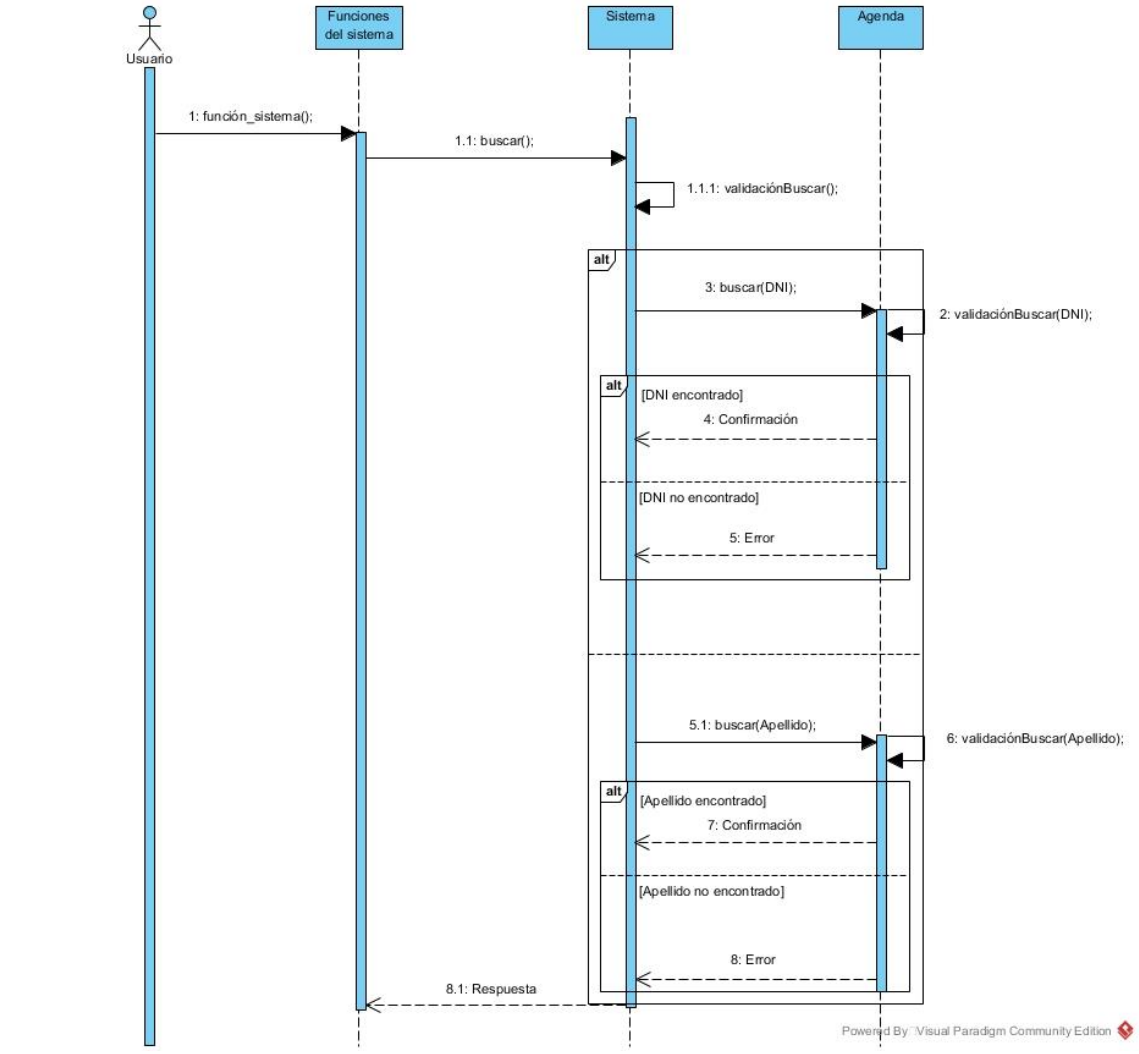

Figura 3.9

El usuario solicita al sistema la búsqueda de un alumno. Tras la solicitud, el sistema debe validarla: si no hay alumnos en el sistema, la solicitud de validación será denegada; por el contrario, si hay alumnos en el sistema, la validación se

realizará con éxito. Una vez la validación se ha obtenido con éxito, el sistema buscará por DNI o por apellido, dependiendo del campo introducido.

- Si el DNI es encontrado en la agenda, el sistema devolverá una confirmación, por el contrario, se producirá un error que nos llevará de vuelta al menú principal.
- Al igual que ocurre con el DNI, si el apellido es encontrado en la agenda, se producirá una respuesta de confirmación, de la otra manera, nos enviará un mensaje de error y nos devolverá al menú principal.

3.3.3. Diagrama de secuencia: Modificar alumno

Este diagrama de secuencia, mostrado en la figura 3.10, describe el flujo de eventos existentes en el caso de modificar a un alumno.

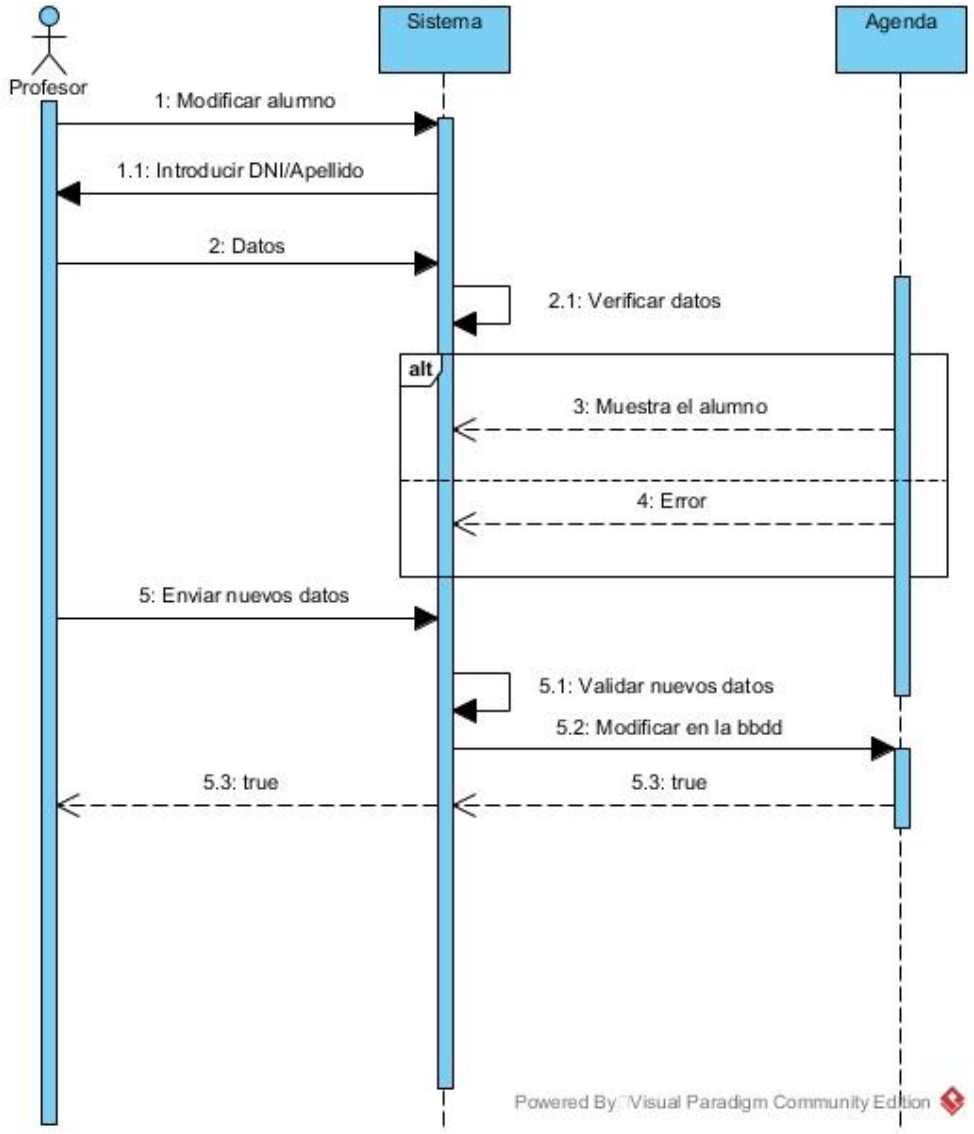

Figura 3.10

El usuario solicita al sistema la modificación de un alumno. Tras dicha solicitud, el sistema pide el envío de los datos necesarios para la modificación. Después de verificarlos, se pueden obtener dos posibles primeras respuestas:

- En primer lugar, si los datos se encuentran en el sistema es que existe el alumno, con lo cual se podrá modificar. El usuario introduce los nuevos datos y el sistema envía una última respuesta de éxito.
- Sin embargo, si los datos introducidos no se encuentran en el sistema, es decir, no existe el alumno, no habrá datos posibles que modificar, con lo que nos devolverá un mensaje de error.

3.3.4. Diagrama de secuencia: Eliminar alumno

Este diagrama de secuencia, mostrado en la figura 3.11, describe el flujo de eventos existentes en el caso de eliminar a uno y varios alumnos.

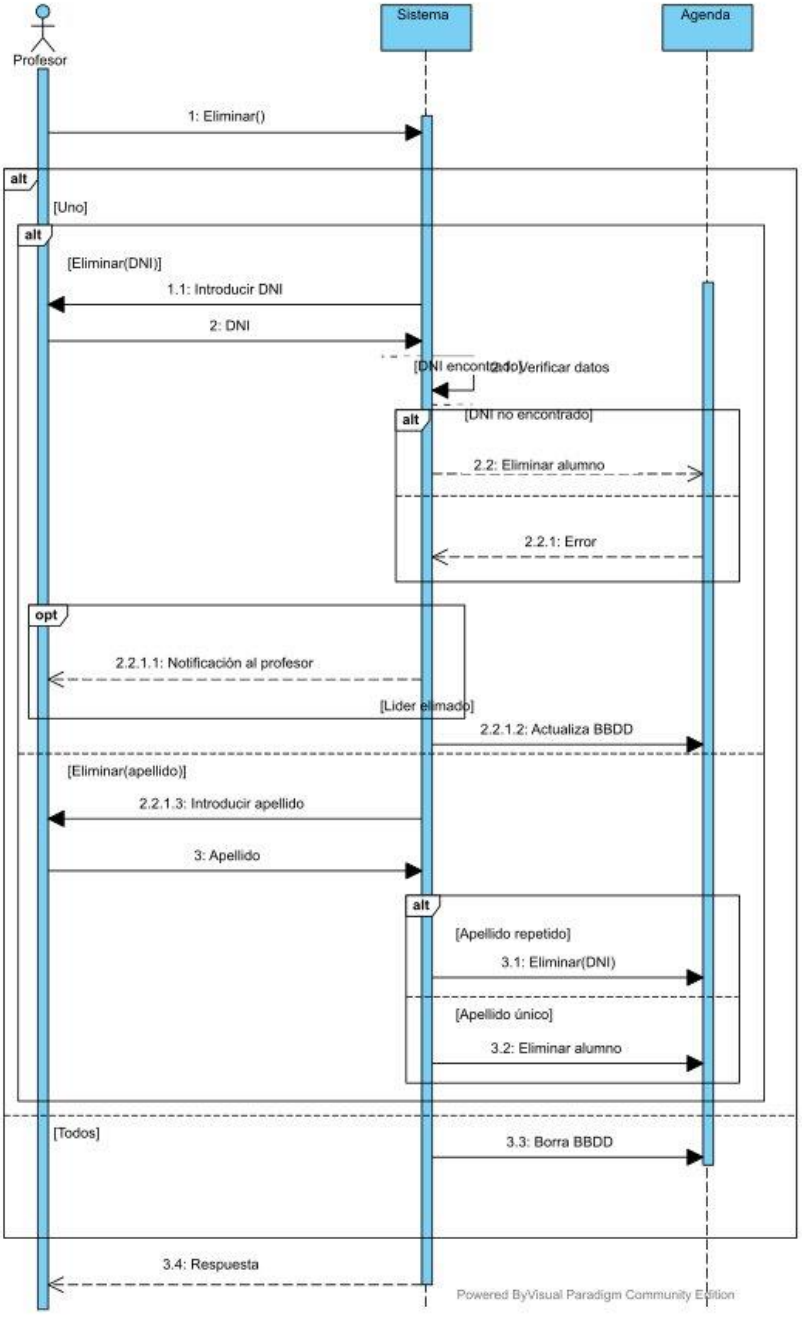

Figura 3.11

El usuario solicita la opción de eliminar. El sistema le da opción a eliminar a un alumno o a todos. En caso de que seleccione eliminar todos, el sistema borrará todos los alumnos de la base de datos automáticamente.

En caso de que solicite eliminar solo a uno, el sistema le dará opción a buscar al alumno por DNI o por apellido.

Si selecciona por DNI, se comprobará si el DNI existe en la base de datos. En caso de que exista, eliminará al alumno. En caso de que el DNI no se encuentre en la base de datos, mostrará un mensaje de error.

Si selecciona eliminar por apellido, comprobará si ese apellido es único en la base de datos. Si lo es, eliminará al alumno. Por el contrario, si el apellido no es único, pasará a pedir el DNI para eliminar al alumno por DNI.

3.3.5. Diagrama de secuencia: Mostrar alumno

Este diagrama de secuencia, mostrado en la figura 3.12, describe el flujo de eventos existentes en el caso de mostrar a un alumno.

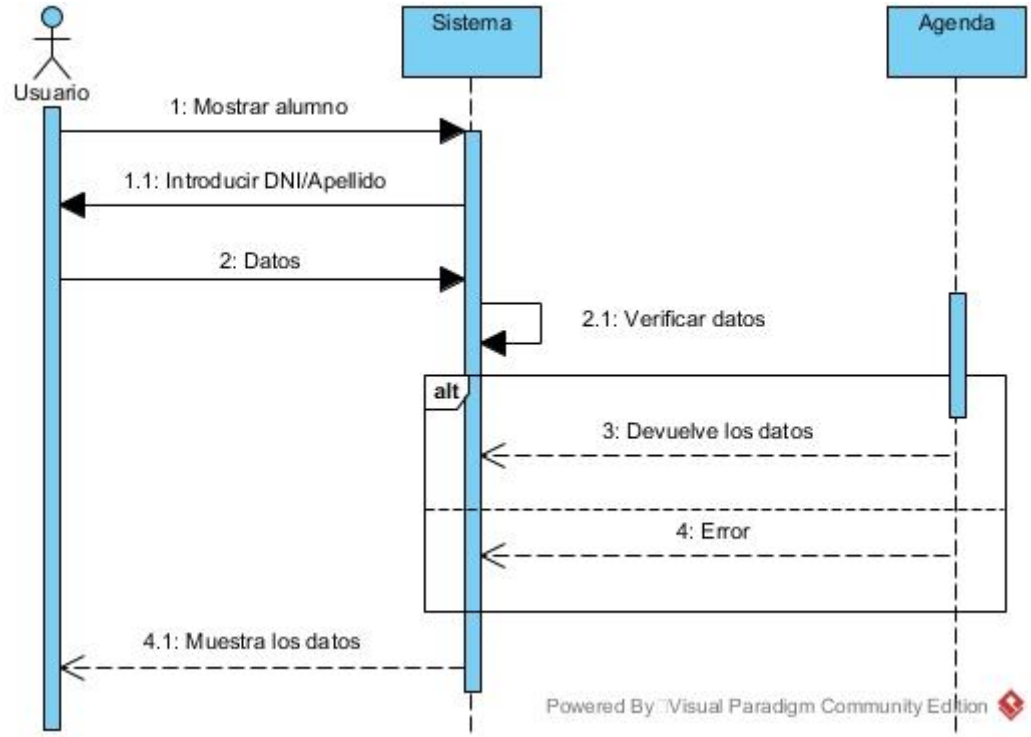

Figura 3.12

El usuario solicita al sistema la muestra de los datos de un alumno. Este solicita el envío de los datos necesarios para la muestra, ya sea DNI o apellido. Tras la introducción de los datos, este devolverá los datos del alumno si se encuentra en el sistema; por el contrario, si no se encuentra, devolverá un mensaje de error.

3.3.6. Diagrama de secuencia: Cargar

Este diagrama de secuencia, mostrado en la figura 3.13, describe el flujo de eventos existentes en el caso de cargar la agenda en el sistema.

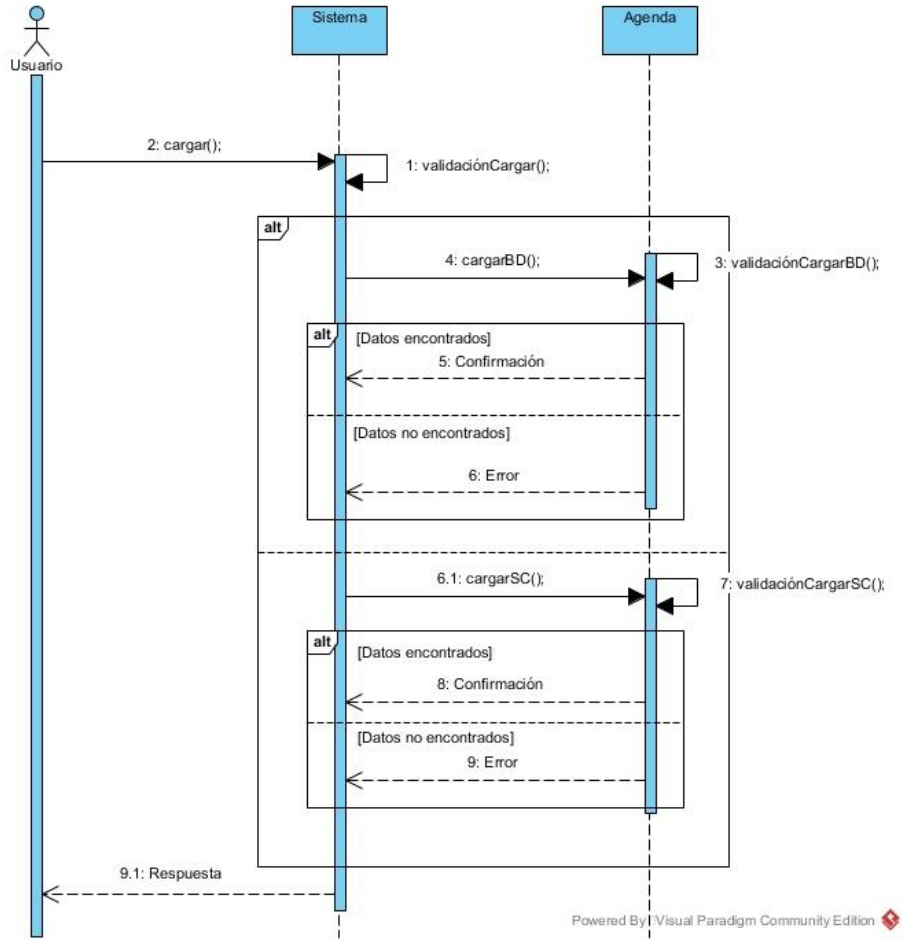

Figura 3.13

El usuario carga los datos de un fichero binario en el sistema que le permite acceder a la agenda. Si hay datos que cargar, se enviará una confirmación de la carga; por el contrario, si no existen datos en el fichero, el sistema no cargará nada. El usuario tiene la alternativa de cargar cualquier fichero, ya sea el original o la copia de seguridad que haya realizado. La ejecución se realizará de la misma manera.

3.3.7. Diagrama de secuencia: Guardar

Este diagrama de secuencia, mostrado en la figura 3.14, describe el flujo de eventos existentes en el caso de guardar los datos introducidos o las modificaciones realizadas.

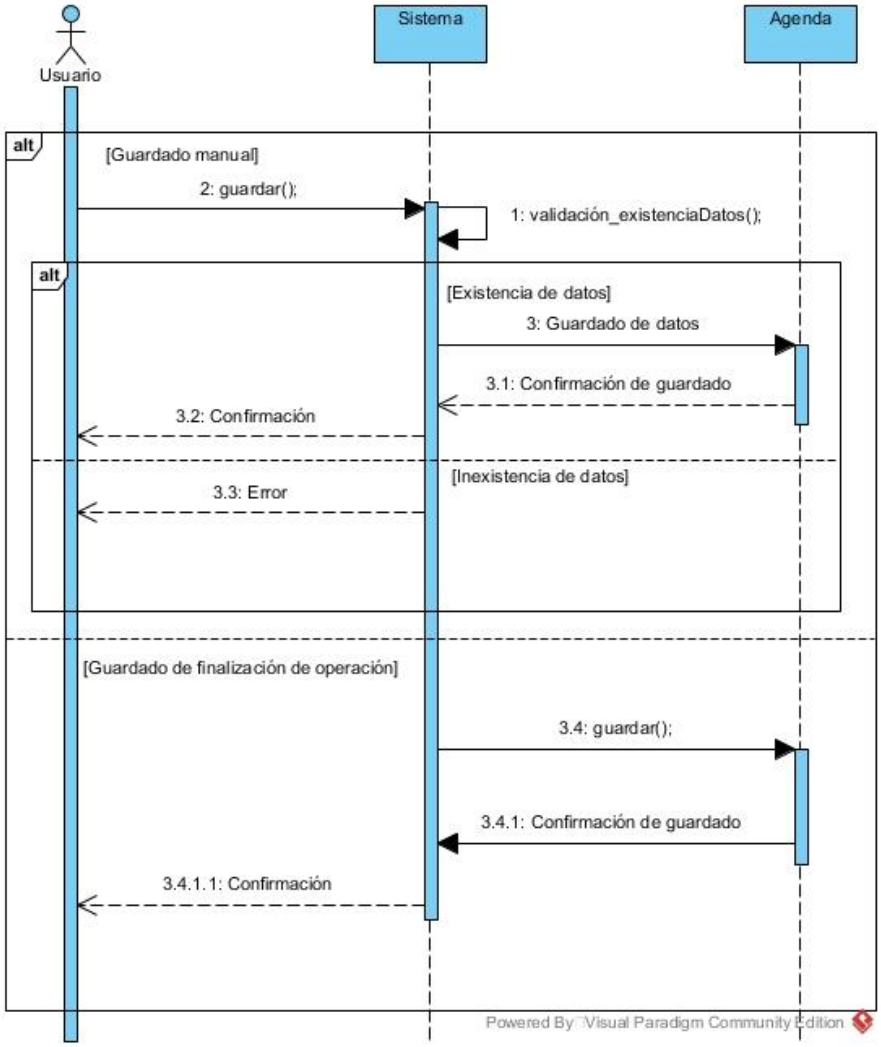

Figura 3.14

El usuario puede solicitar al sistema el guardado de los datos de manera manual o tras realizar una operación que modifique los datos de la agenda. Si existen datos que guardar, el sistema los guardará en la agenda, enviando una

confirmación si se han guardado con éxito; sin embargo, si no hay existencia de datos, la respuesta enviada será de un mensaje de error.

Éste último aspecto no está presente en el autoguardado, ya que de por sí este funciona cuando se han realizado cambios y, por consiguiente, tiene datos que guardar.

3.3.8. Diagrama de secuencia: Login

Este diagrama de secuencia, mostrado en la figura 3.15, describe el flujo de eventos existentes en el caso de un usuario no identificado entre al sistema.

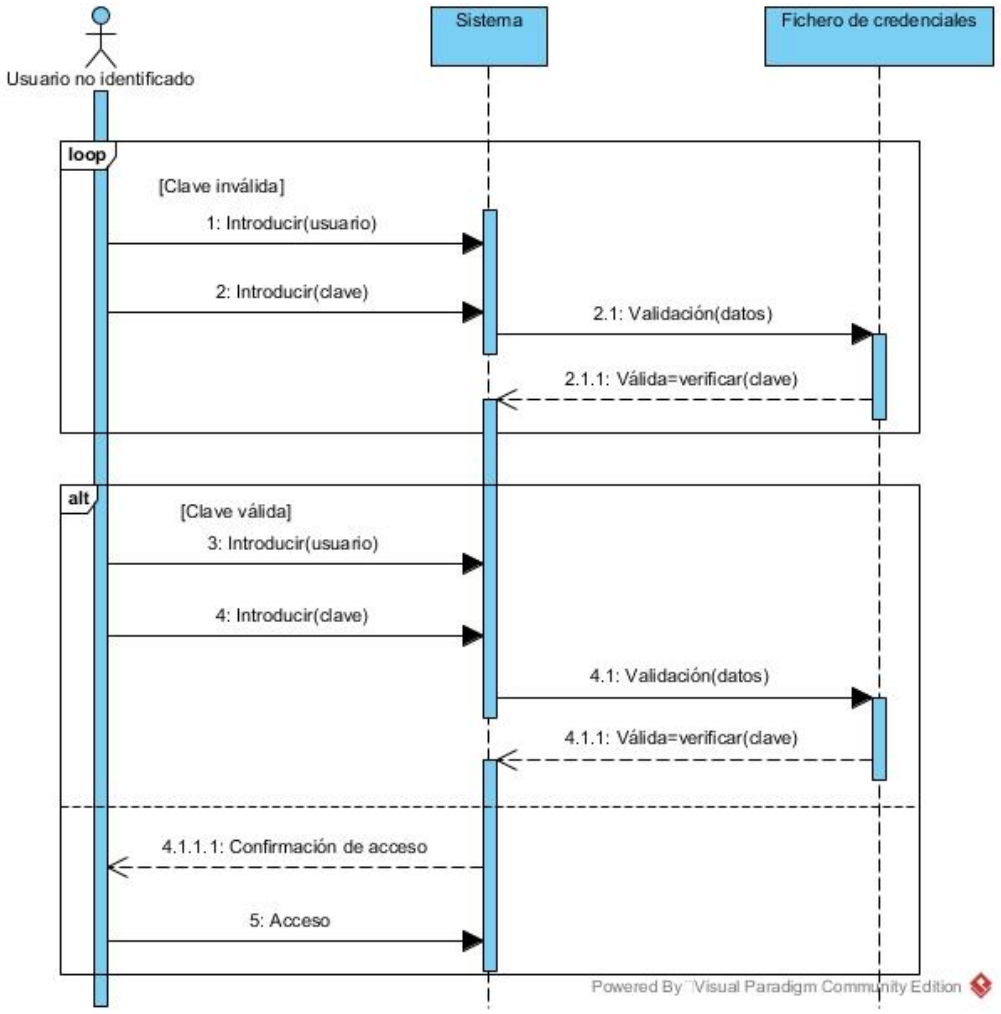

Figura 3.15

El usuario debe introducir las credenciales necesarias para validar su acceso al sistema. Mientras dichas credenciales sean inválidas, el sistema seguirá preguntando de nuevo en forma de bucle dichas credenciales. Si, por el contrario, las credenciales son válidas, se produce una confirmación del acceso al sistema, permitiendo a dicho usuario la entrada al mismo.

4. Técnicas de validación

En este apartado se llevarán a cabo las técnicas de validación propias del modelado que estamos desarrollando.

4.1. Matriz de trazabilidad

La matriz de trazabilidad nos ayuda a realizar un seguimiento de los requisitos. Lo que buscamos es que se estén cumpliendo todos los requisitos. En nuestro proyecto aplicaremos la matriz <requisitos funcionales – casos de uso> y la matriz <casos de uso – clases>.

4.2.1. Matriz RF/CU

Se han tenido en cuenta 8 requisitos funcionales. En cuanto a los casos de uso, hemos definido 8 también.

Como podemos comprobar, cada RF está cubierto por un CU, es decir, el RF1 está cubierto por el CU1, el RF2 por el CU2, y así sucesivamente. Con esto hemos asegurado que todas las funcionalidades se han tenido en cuenta:

	CU1	CU2	CU3	CU4	CU5	CU6	CU7	CU8
RF1	X							
RF2		X						
RF3			X					
RF4				X				
RF5					X			
RF6						X		
RF7							X	
RF8								X

Figura 4.1

En la matriz se puede observar claramente que cada RF está cubierto por un CU.

4.2.2. Matriz CU/C

Con la matriz casos de uso – clases, lo que comprobamos es que

cada caso de uso tiene asignado una clase. En caso de que haya uno o varios casos de uso sin asignar a una clase, tendríamos que mejorar la clase o crear otra.

En nuestro caso hemos definido 3 clases: Profesor, Alumno y Agenda.

	CU1	CU2	CU3	CU4	CU5	CU6	CU7	CU8
Profesor						X	X	X
Agenda	X	X	X	X	X			
Alumno								

Figura 4.2

Como podemos observar en la matriz, todos los casos de uso se asignan a una clase, por lo que nuestras clases están bien definidas.

Sin embargo, observamos que no hay ningún CU asociado a la clase alumno. No obstante, dicha clase es necesaria para el correcto funcionamiento del software, ya que nos proporciona los datos del alumno.

5. Procedimiento *SCRUM*

El equipo de desarrollo está formado por tres personas: Sergio Gómez Fernández, Ventura Lucena Martínez y Eloy Gómez García, siendo el *Scrum Master* Sergio; y Ventura y Eloy los encargados de implementar las funcionalidades. Como beneficiaros finales, tenemos a los profesores que van a utilizar nuestro software.

5.2. *Product Backlog*

En cuanto a la prioridad y el tiempo empleado de las funcionalidades tenemos:

	LOGIN	CARGAR	GUARDAR	AÑADIR	BUSCAR	ELIMINAR	MODIFICAR	MOSTRAR
ID	008	006	007	001	002	004	003	005
PRIORIDAD	1	1	1	2	3	4	4	5
TIEMPO	2.30 h	2 h	2 h	3 h	3 h	2.30 h	3 h	3 h

Para el desarrollo de nuestro proyecto se han estimado 21 horas totales.

5.3. *Sprint Backlog*

Los roles que se han seguido en nuestro proyecto son los siguientes:

→ **Sergio Gómez Fernández**: *Scrum Master* y tester.

→ **Ventura Lucena Maertínez**: Programador y tester.

→ **Eloy Gómez García:** Programador y tester.

Se han tardado tres semanas en desarrollar nuestro software. Los avances que se han ido haciendo cada semana han sigo los siguientes:

Semana 1 (27/11/2018 – 04/12/2018)

1. Creación del repositorio. **(Sergio)**
2. Creación del *Product Backlog*. **(Sergio)**
3. Creación de la clase alumno. **(Ventura)**
4. Creación de la función guardar. **(Eloy)**
5. Creación de la función cargar. **(Eloy)**
6. Creación de la función registro. **(Eloy)**
7. Comprobación de su funcionamiento y prueba de la misma. **(Ventura)**
8. Creación del *Burndown Chart*. **(Sergio)**

Semana 2 (04/12/2018 - 11/12/2018)

1. Comprobación de la función guardar a partir de un alumno de prueba. **(Ventura)**
2. Corrección del *Product Backlog*. **(Sergio)**
3. Modificación de la clase alumno. **(Ventura)**
4. Prueba de las funciones guardar y cargar en un prueba.cc. **(Ventura)**
5. Actualización del *Burndown Chart*. **(Sergio)**

Semana 3 (11/12/2018 - 16/12/2018)

1. Creación de la clase agenda. **(Ventura)**

2. Modificación de la clase registro. **(Ventura)**

3. Mejora y de la clase agenda y agenda.h. **(Ventura)**

4. Modificación de la clase profesor y profesor.h. **(Eloy)**

5. Creación de la clase login. **(Eloy)**

5. Creación de la función main. **(Eloy)**

6. Restricciones de los atributos del alumno (DNI, telefono). **(Ventura)**

7. Modificación final del *Burndown Chart*. **(Sergio)**

8. Cambios en el constructor de la función añadir y modificar. **(Ventura)**

9. Corrección de limpieza del buffer. **(Ventura y Eloy)**

10. Corrección de la función de comprobación del DNI. **(Sergio)**

11. Testeo del programa. **(Ventura, Sergio y Eloy)**

12. Finalización del *Sprint Backlog*. **(Sergio)**

5.4. *BurnDown Chart*

En la figura 5.1 se muestra cómo se han ido reduciendo las horas de nuestro proyecto por cada sprint realizado:

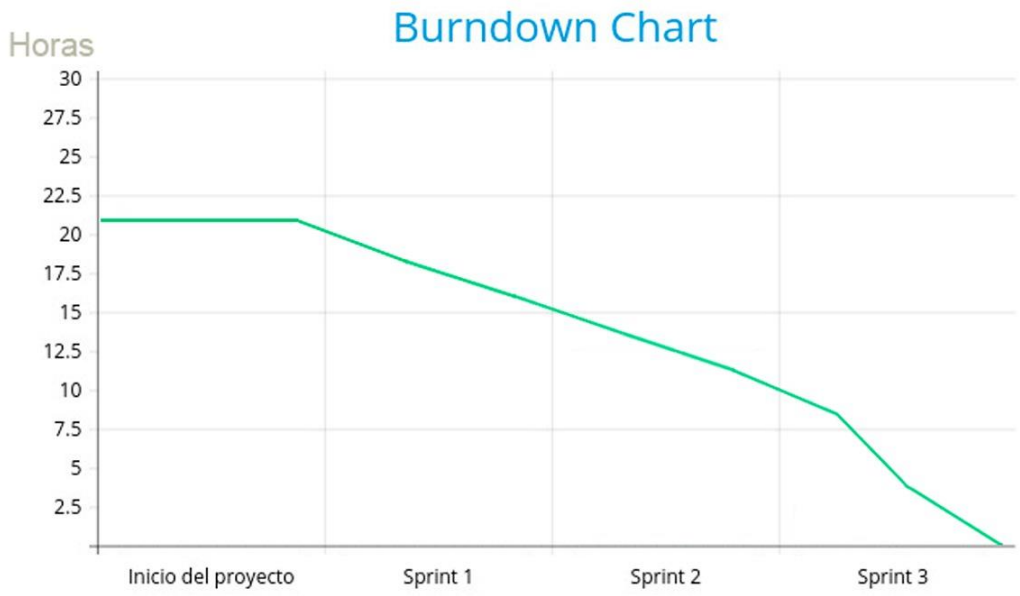

Figura 5.1

Como se puede observar en el gráfico, al inicio del proyecto se estimaron 21 horas. Tras realizar el primer sprint, se redujo el tiempo en 6 horas. En la segunda semana, se trabajó 5 horas, por tanto, quedaban 10 horas restantes. Finalmente, en la última semana, se trabajó durante 10 horas aproximadamente hasta la finalización del proyecto.

Bibliografía

- Ingeniería del Software – Escuela Politécnica Superior de Córdoba.
- Desarrollo del código en lenguaje C++:
  1. http://c.conclase.net/
  2. http://www.cplusplus.com/