



UNIVERSIDAD SIMÓN BOLÍVAR

DPTO. DE COMPUTACIÓN Y TECNOLOGÍA DE LA INFORMACIÓN

CI3715 – INGENIERÍA DE SOFTWARE I

Profs. Alejandro Teruel, Alfonso Reinoza, Jean Carlos Guzmán

Repositorio: <https://github.com/Sealos/SAGE>

SAGE

Integrantes:

Paradigm Shift

Stefano De Colli 09-10203

M. Gabriela Giménez 09-10329

Carlo Polisano 09-10672

Victor Suniaga 09-10832

Alejandro Guevara 09-10971

Stefany Botero 10-10088

Sartenejas, 25 de Noviembre 2014

Historia de Revisiones

Fecha	Versión	Descripción	Autor
<25/11/2014>	<1.0>	<i>Informe SAGE : v1.0.0</i>	<i>Paradigm Shift</i>

Índice general

1. Introducción	4
2. Requisitos	5
2.1 Descripción del dominio	5
2.2 Propuesta y enunciado del problema	5
2.3 Afectados	5
2.4 Historias de usuario	6
2.5 Requisitos no funcionales	8
2.6 Aspectos legales y éticos de la propuesta	9
3. Diseño	11
3.1 Arquitectura del producto	11
3.2 Diseño	11
3.2.1 Diagrama de clases	12
3.2.2 Diagrama de interacción: Secuencia	13
3.3 Diseño de las interfaces con usuarios	14
3.4 Diseño de la base de datos	16
4. Programación	17
4.1 Tabla de revisión histórica	17
5. Verificación	18
5.1 Informe de pruebas unitarias	18
5.2 Informe de pruebas de integración	19
6. Métodos, técnicas y herramientas utilizadas	20
6.1. Métodos	20
6.1.1 Programación extrema (XP)	20
6.1.2 Scrum	22
6.1.3 XP vs. Scrum	23
6.2. Técnicas	24
6.2.1 Programación dirigida por casos de pruebas (TDD)	24
6.2.2 Refactorización	25
6.2.3 Pruebas unitarias	25

6.2.4 Gestión de configuraciones y versiones	26
6.2.5 Pruebas de integración	26
6.2.6 Pruebas de regresión	27
6.3. Herramientas	27
6.3.1 Eclipse	27
6.3.2 Python	28
6.3.3 PyUnit	28
6.3.4 Git	29
6.3.5 Django	29
7. Conclusiones	30
8. Recomendaciones	31
9. Bibliografía	32

Capítulo 1

Introducción

En el presente informe se quiere dar una explicación acerca de la primera versión o versión demo del software desarrollado por el equipo de Paradigm Shift que es el Sistema Automatizado de Gestión de Estacionamientos (SAGE). Se enfocó en varios aspectos como el diseño que se utilizó, la descripción de sus funcionalidades con todas sus características, el tipo de pruebas realizadas para corroborar su correctitud, las metodologías utilizadas para lograr éste proyecto en períodos cortos de tiempo, las técnicas empleadas para cumplir con las metodologías y finalmente, las herramientas empleadas para lograr concluir el proyecto de manera más fácil y rápida.

Capítulo 2

Requisitos

2.1 Descripción del dominio

El Sistema Automatizado de Gestión de Estacionamientos (SAGE), cuya principal innovación es la opción de reservar puestos en línea y el ingreso vía móvil por los parqueros del estado de los puestos.

La Asociación de Propietarios y Administradores de Garajes y Estacionamientos de Caracas (APAGEC) nos ha contratado para desarrollar prototipos de SAGE que permitan explorar la factibilidad de nuevos modelos de negocio.

2.2 Propuesta y enunciado del problema

Se propone la creación de un Sistema Automatizado de Gestión de Estacionamientos para los clientes pertenecientes a APAGEC, con reservas y pagos vía web.

Las funcionalidades propuestas y realizadas para esta primer demo de SAGE son: Ingresar datos del estacionamiento, parametrizar estacionamientos, cambiar estacionamiento, disponibilidad de un horario para reservar. Cada una de estas funcionalidades se describirá más detalladamente en las historias de usuario.

2.3 Afectados

Los Stakeholders o afectados son todas aquellas personas que serán impactados de manera positiva o negativa al término del proyecto, es decir, grupo de personas o instituciones

que se encuentran directa o indirectamente involucrados con las actividades, productos o servicios de una organización y tienen potencial para influir en estos.

El análisis de los stakeholders es una herramienta de clasificación que ayuda a identificar y determinar el impacto o apoyo que cada uno de estos puede generar. La razón principal para la identificación de los stakeholders es que una buena o mala decisión que impacte a alguno de éstos puede generar un impulso y buenos resultados sobre una organización, o por el contrario, hacerla fracasar e incluso desaparecer. Dentro de los stakeholders se pueden encontrar dos tipos:

- **Stakeholders Internos:** La mayoría de los stakeholders claves son las personas que laboran dentro de la organización sobre la cual se va a desarrollar el proyecto, en este grupo encontramos a los clientes internos (normalmente los que quieren el proyecto y pagaron por ello), patrocinadores, desarrolladores, supervisores (jefe del proyecto que tiene gran interés por el éxito del proyecto), y diferentes grupos de apoyo.

En este proyecto tenemos como stakeholders internos al equipo desarrollador (team), al Scrum Master (que es quien lidera al equipo desarrollador) y el cliente que pidió la creación del software desarrollado (product *Owner*).

- **Stakeholders Externos:** Los de este grupo tienen interés intrínseco en el proyecto, aunque no formen parte de la organización, dentro de estos encontramos a los clientes externos, proveedores, contratistas, consultores, usuarios que vayan a usar el proyecto (por ejemplo en este caso clientes que usen el software en desarrollo) entre otros.

Como stakeholders externos tenemos, por ahora, a los clientes que en un futuro utilicen el producto desarrollado para realizar sus reservaciones.

2.4 Historias de usuario

A continuación se proporciona la lista de las historias INVEST que se han acordado en las reuniones de Sprint con el cliente y que ya han sido implementadas.

1. **Nuevo estacionamiento:** Como vendedor de SAGE, puede introducir los datos de un nuevo estacionamiento:
 - Nombre del dueño (no se permiten dígitos).
 - Nombre del estacionamiento (cualquier secuencia de caracteres es permitido).
 - Dirección del estacionamiento.
 - Teléfonos (hasta 3, celulares y teléfonos de Caracas, sólo dígitos).
 - Correos electrónicos (hasta 2, validación mínima).
 - RIF (restringido a un primer carácter que puede ser una "J", "V" o "D", seguido de un "-", luego le siguen 8 números, luego otro "-"[opcional] y finalmente un número).
 - Restricciones acordadas:
 - Puede manejar hasta 5 estacionamientos.
 - Restringido a un sólo día de operación.
 - No se necesita autenticar tipos de usuarios (el demostrador "juega" todos los roles).
2. **Parametrizar estacionamiento:** Como dueño de un estacionamiento, se puede indicar:
 - La capacidad de un estacionamiento.
 - El horario de apertura (p. ej. 8:00 a 15:00) y, si hay un horario restringido para las reservaciones, cuál sería el horario (hora militar).
 - Tarifa (Puede entero o con decimales p. ej. 3.50)
3. **Cambiar entre estacionamientos:** Como propietario de varios estacionamientos puede cambiar de vista para ver las características de sus estacionamientos.
4. Editar datos de **Parametrizar estacionamiento.**

5. **Disponibilidad:** Como usuario se puede hacer una solicitud de reserva:

- Se solicita una reserva desde una hora (p. ej. 9:34) hasta una hora (p. ej. 14:15). Se debe realizar en hora militar.
- El sistema indica si la solicitud es factible o no. La solicitud es factible si existe puesto en el estacionamiento para el tiempo exacto especificado.
- Se reserva un puesto (los carros no se mueven de puesto).
- Restricciones adicionales:
 - Todos los puestos son reservables.
 - La reserva mínima es por una hora.
 - No se sobrevenden reservaciones.

6. **Agregar persistencia al demo:**

- Los datos y parámetros de los estacionamientos se resguardan en una base de datos. Al reiniciarse el demo, los datos y parámetros de los estacionamientos se obtienen de la base de datos.
- Las reservaciones se resguardan en una base de datos y se obtiene de ella al reiniciarse el demo.

2.5 Requisitos no funcionales

Los requisitos no funcionales, también conocidos como características de calidad, son los tipos de requisitos que pueden utilizarse para juzgar la operación de un software, en lugar de sus comportamientos específicos.

- **Mantenibilidad:** Es la capacidad del software para realizar modificaciones específicas. Estas modificaciones pueden incluir correcciones, mejoras o adaptaciones. Este requisito se logra a través del uso de la refactorización, debido a que esta técnica nos permite llevar siempre el código lo más limpio posible y entendible, por lo que así nos permite asegurar su mantenibilidad.

- **Usabilidad:** Es la capacidad del producto o software para ser entendido, aprendido, usado y que es atractivo para el usuario. En referencia al proyecto, se puede decir que éste será usable debido a que se va a ir continuando su desarrollo a través de los trimestres, por lo que el código implementado no será desechado.
- **Eficiencia:** Es la capacidad del software para proporcionar un rendimiento adecuado, respecto a la cantidad de recursos utilizados y bajo ciertas restricciones. El atributo de la eficiencia se ve más enfocado hacia la parte del tiempo. Por lo que al momento de diseñar el software se tomó en cuenta la estructura de acceso más rápido, para así tener los mejores resultados.
- **Portabilidad:** Es la capacidad del producto para ser transferido de un entorno a otro. Por lo que este producto que se ha realizado, puede ser instalado con facilidad en cualquier computadora que tenga un sistema operativo Linux, y que éste a su vez contenga las herramientas necesarias para su correcto funcionamiento.
- **Confiabilidad:** Es la capacidad del software para mantener su nivel de rendimiento bajo ciertas condiciones, en un período de tiempo. Esto se puede lograr a través de la realización de varios tipos de pruebas, que va a permitir que el software no muestre errores de código a la hora de introducir alguna información errada. Dentro de la confiabilidad se pueden encontrar el atributo de tolerancia a fallas, que refleja un poco lo mencionado anteriormente.

2.6 Aspectos legales y ética de la propuesta

Con respecto a los *aspectos legales*, es importante recalcar el **Artículo 10** ya que todos nosotros como futuros ingenieros debemos estar conscientes de que todos los documentos que se hacen en un proyecto nos pertenecen y sin nuestro consentimiento no se puede hacer uso de ellos.

También es importante tener presente el **Artículo 14** “... Los profesionales deberán abstenerse de prestar su concurso profesional cuando esta disposición no sea satisfactoriamente cumplida y dejan de acatarse las medidas que ellos indiquen con ese fin” ya que sino se garantiza la eficacia y seguridad del proyecto en desarrollo se tiene que asumir la responsabilidad del mismo.

Por último, y no menos importante, hay que tener y entender muy bien el **Artículo 37** que hace una clara referencia a la privacidad y confidencialidad de los datos que se ingresen en el software desarrollado, es decir, se tiene que garantizar a las personas que utilicen el software que todos sus datos (personales y de pago electrónico) no podrá ser accesible para terceros que no estén autorizados y puedan hacer un mal uso de los mismos, de no cumplirse esto será penado por la Ley como allí se establece.

En cuanto a la *ética profesional*, se puede destacar el **Artículo 5 (dispensa)**, que menciona que no se debe excusar, por amistad o conveniencia el cumplimiento de actividades obligatorias, cuando se le tiene asignado realizarlas. Debido a que es importante que todos sean responsables y participen y/o realicen las actividades que les corresponde.

También es interesante mencionar el **Artículo 7 (remuneración)**, en donde se exige que no se debe realizar informes con negligencia, ligereza o con opiniones que sean indebidamente optimistas. Esto es importante destacar, ya que los informes deberían de ser objetivos, en dónde se debe encontrar todos los aspectos del software debidamente documentados.

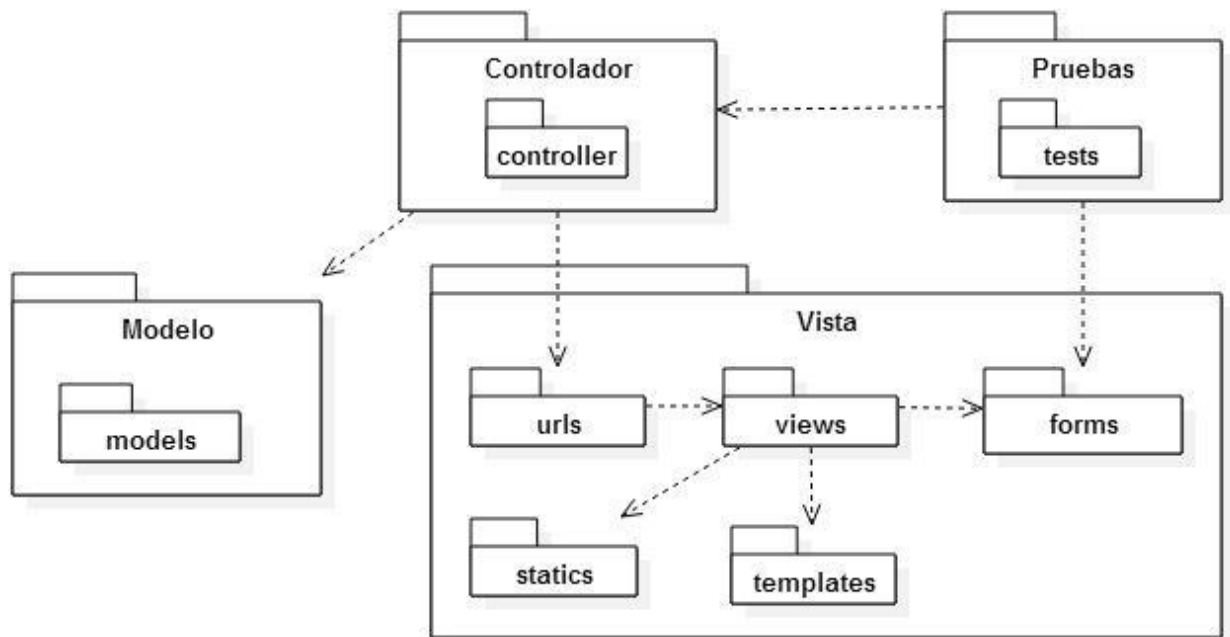
Por último, se puede mencionar el **Artículo 18 (autoría)**, que dice textualmente “Utilizar estudios, proyectos, planos, informes u otros documentos, que no sean el dominio público, sin la autorización de sus autores y/o propietarios”. Esto es un punto primordial, ya que toda información que sea utilizada en el proyecto, específicamente en la documentación, que haya sido tomado de otras fuentes, debe ser referenciado en el mismo, porque sino será tomado como plagio.

Capítulo 3

Diseño

3.1 Arquitectura del producto

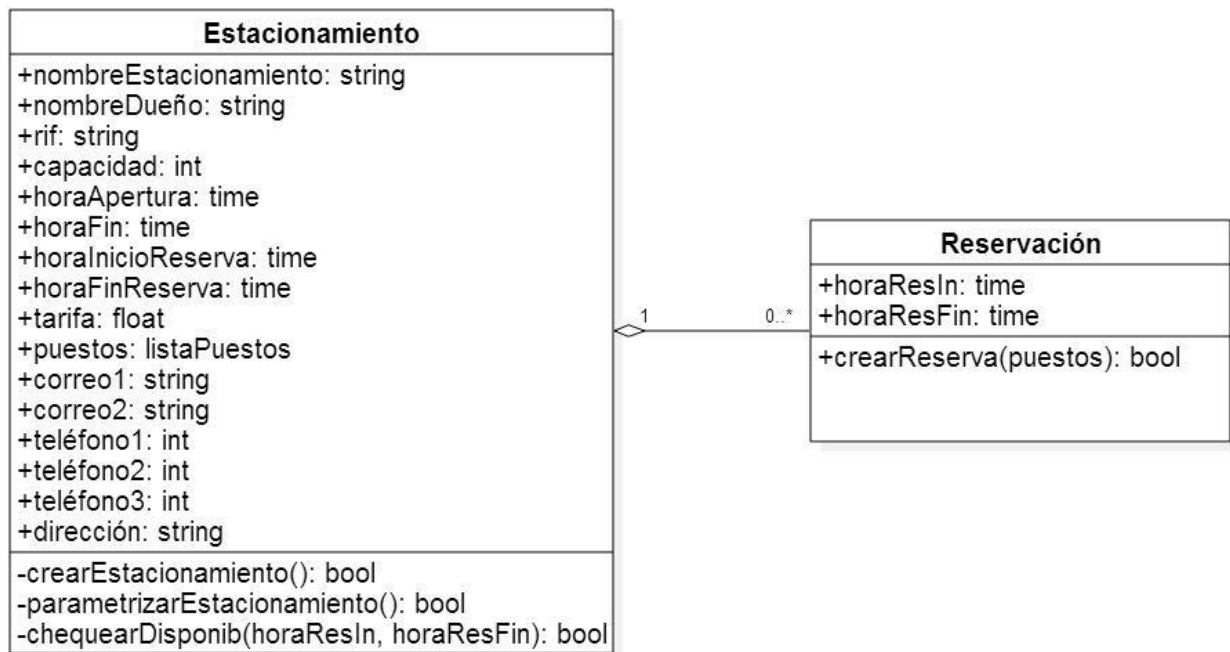
A continuación se presenta el Diagrama UML de paquetes realizado para el diseño de la arquitectura del software, SAGE que se realizó basado en el Modelo Vista Controlador, debido a que Django está inspirado en esta filosofía.



3.2 Diseño

En este punto se presentarán los Diagrama UML de Clases del Dominio, y Interacción, específicamente el Diagrama de Secuencia y posteriormente unas breves anotaciones sobre de los mismos.

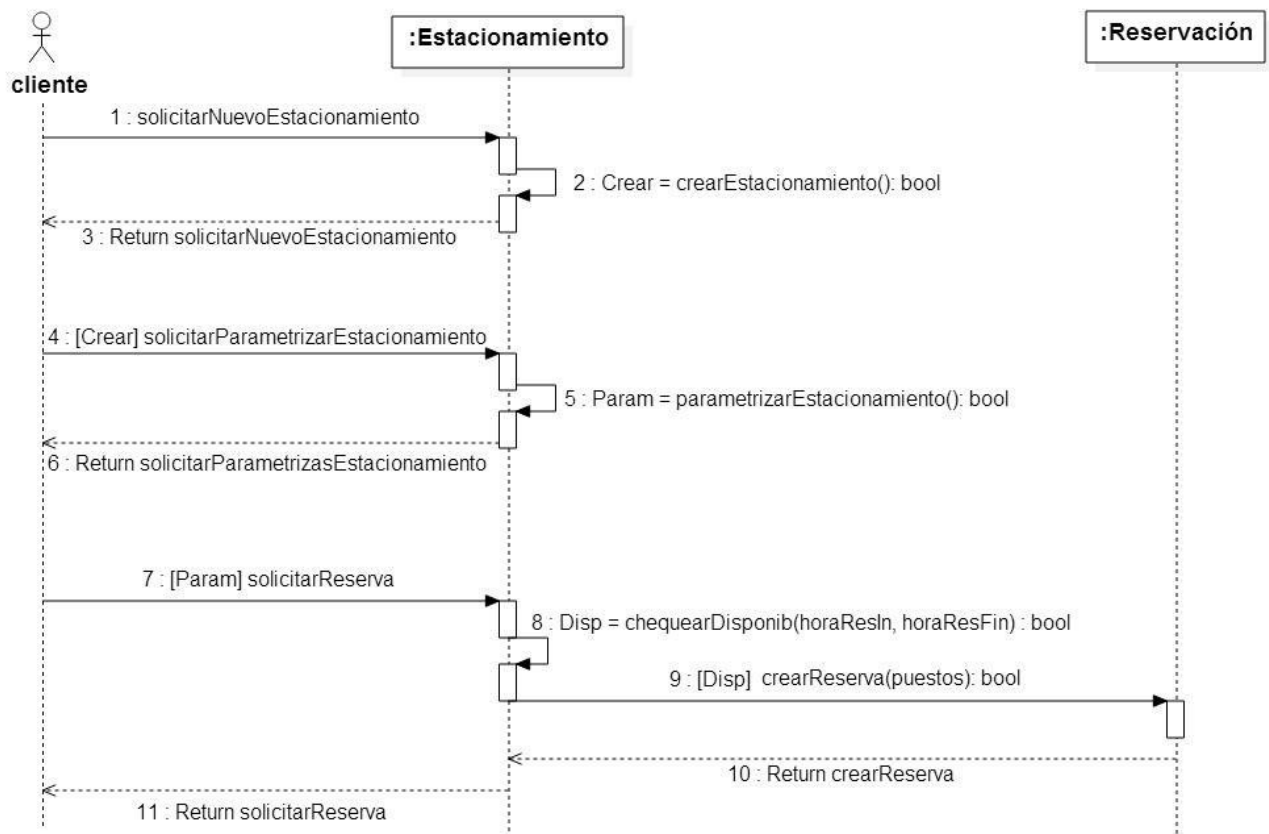
3.2.1 Diagrama de clases



Notaciones:

- **listaPuestos:** Lista que representa cada puesto del estacionamiento y a su vez, cada puesto contiene una lista que inicialmente tiene una tupla con el inicio y fin de las horas de reservación.
- **horaApertura:** Hora que abre el estacionamiento.
- **horaFin:** Hora que cierra el estacionamiento.
- **horaInicioReserva:** Hora en que pueden empezar las reservas en el estacionamiento.
- **horaFinReserva:** Hora en que finaliza la reserva de estacionamientos.
- **horaResIn:** Hora en que inicia una reserva en particular.
- **horaResFin:** Hora en que finaliza una reserva en particular.

3.2.2 Diagrama de interacción: Secuencia



Notaciones:

- **solicitarNuevoEstacionamiento:** El cliente solicita al sistema que va a crear un nuevo estacionamiento y llena el formulario.
- **crearEstacionamiento():** El sistema verifica si hay menos de 4 estacionamientos creados y guarda los datos ingresados en el sistema y le notifica al cliente si fue un éxito o no.
- **solicitarParametrizarEstacionamiento:** Una vez creado el estacionamiento el cliente puede llenar el formulario de parametrizar.
- **parametrizarEstacionamiento():** El sistema agrega la nueva información del estacionamiento previamente creado y se la muestra al cliente.
- **solicitarReserva:** El cliente puede realizar una reserva una vez creado y parametrizado el estacionamiento, por lo que llena el formulario con el bloque de hora a reservar.
- **chequearDisponib(horaResIn, horaResFin):** El sistema verifica si hay disponibilidad de un puesto a esa hora que se reservó y notifica al usuario si hay o no esa disponibilidad.

3.3 Diseño de las interfaces con usuarios

A continuación se presenta las vistas de implementación del diseño de las interfaces del sistema SAGE, para cada funcionalidad creada.

➤ Nuevo estacionamiento

SAGE

Menú

Estacionamientos

Estacionamientos

	Dueño	Nombre	RIF
1	Paradigm Shift	USB-Park	J-19992345-4

Crear estacionamiento

Propietario

Nombre

Direccion

Telefono 1

Telefono 2


Telefono 3

Email 1

Email 2

RIF

Crear estacionamiento



SAGE

Menú

Estacionamiento USB-Park

Estacionamientos

Nombre del estacionamiento:

USB-Park

Nombre del propietario:

Paradigm Shift

Dirección:

Sartenejas, Caracas

Teléfono:

02124567345

Teléfono:

0416-7896543

Correo Electrónico:

paradigmshift@gmail.com

RIF:

J-19992345-4

Parametrizar

Número de Puestos

Horario Apertura


Horario Cierre

Horario Inicio Reserva

Horario Fin Reserva

Tarifa

Parametrizar Estacionamiento



➤ Parametrizar estacionamiento

SAGE

Menú

[Estacionamientos](#)
[Reservar](#)

Estacionamiento USB-Park

Nombre del estacionamiento:

USB-Park

Nombre del propietario:

Paradigm Shift

Dirección:

Sartenejas, Caracas

Teléfono:

02124567345

Teléfono:

0416-7896543

Correo Electrónico:

paradigmshift@gmail.com

RIF:

J-19992345-4

Puestos:

50

Horario inicio de reserva:

07:30:00

Horario fin de reserva:

19:30:00

Horario apertura:

07:00:00

Horario cierre:

20:00:00

Tarifa:

2.30 Bsf.

Parametrizar

Número de Puestos

Horario Apertura


Horario Cierre

Horario Inicio Reserva

Horario Fin Reserva

Tarifa

Parametrizar Estacionamiento



➤ Cambiar entre estacionamientos

SAGE

Menú

[Estacionamientos](#)

	Dueño	Nombre	RIF	
1	Paradigm Shift	USB-Park	J-19992345-4	Q
2	Paradigm Shift	Giro	J-20553672-8	Q

Crear estacionamiento

Propietario

Nombre

Dirección

Telefono 1

Telefono 2


Telefono 3

Email 1

Email 2

RIF

Crear estacionamiento



➤ Disponibilidad

SAGE

Menú

Estacionamientos

Reservar

Estacionamiento USB-Park

Nombre del estacionamiento:

USB-Park

Nombre del propietario:

Paradigm Shift

Dirección:

Sartenejas, Caracas

Teléfono:

02124567345

Teléfono:

0416-7896543

Correo Electrónico:

paradigmshift@gmail.com

RIF:

J-19992345-4

Puestos:

50

Horario inicio de reserva:

07:30:00

Horario fin de reserva:

19:30:00

Horario apertura:

07:00:00

Horario cierre:

20:00:00

Tarifa:


2.30 Bsf.

Reserva

8:45

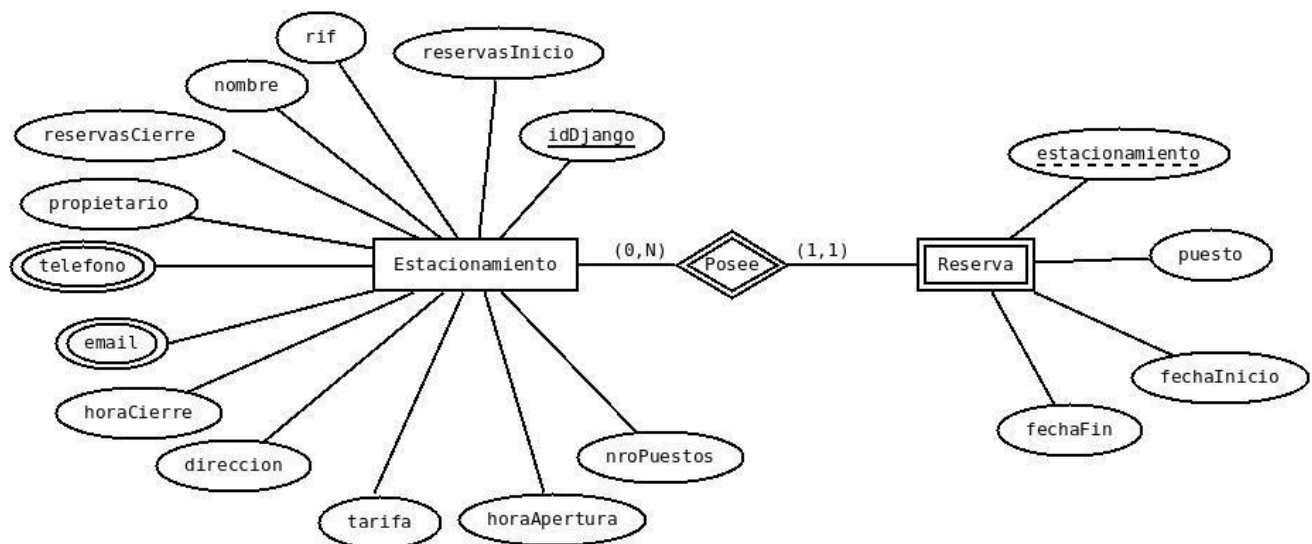
14:30

Chequear Disponibilidad



3.4 Diseño de la base de datos

En este punto se presenta el diagrama de Entidad-Relación que hace referencia a la base de datos generada para el sistema de SAGE.



Capítulo 4

Programación

4.1 Tabla de revisión histórica

La siguiente tabla que se presenta es la tabla de revisión histórica, la cual muestra cada versión del software que quedó en el repositorio. Es esquema de identificación de versiones utilizado para este proyecto es el clásico, que es la utilización de tres números. Por ejemplo: v1.0.1, v1.0.1, v1.0.2, v1.1.0, v2.0.0, etc. Donde el primer número representa un cambio mayor, y el tercero un cambio menor o correcciones menores. Como se realizaron entregas previas antes de llegar al primer release, las notaciones que utilizamos son previas a la v1.0.0, siendo este último el número del release final para este trimestre.

Versión	Descripción
v0.1.0	Se desarrolló la funcionalidad de crear estacionamiento. Se diseñó e implementó la interfaz de nuevo estacionamiento. Permite ver todos tus estacionamientos.
v0.2.0	Se desarrolló la funcionalidad de parametrizar estacionamiento. Se diseñó e implementó la interfaz de nuevo estacionamiento.
v0.3.0	Se desarrolló la funcionalidad de solicitar reserva en donde se verifica si hay puesto disponible según las condiciones del estacionamiento. Se implementó la interfaz correspondiente. Se añadieron algunas pruebas unitarias.
v0.3.1	Se implementó la base de datos. Más pruebas unitarias. Refactorización de código.
v1.0.0	Más pruebas unitarias y creación de pruebas de integración. Refactorización de código. Versión para el release.

Capítulo 5

Verificación

5.1 Informe de pruebas unitarias

Se tomó la decisión de desarrollo, de que al momento de probar la funcionalidad de nuestro sistema, nos enfocamos en probar solamente los códigos desarrollados por nosotros, por lo que las funcionalidades propias de Django no fueron probadas, ya que se piensa que no es necesario, siempre y cuando nuestras funcionalidades funcionen, valga la redundancia, de manera correcta y para eso tenemos todas las pruebas unitarias que se han realizado para este proyecto.

Un ejemplo más conciso sería con la funcionalidad del método POST, este método en Django toma los valores que un usuario registra en una vista (creada por nosotros) en variables temporales que después el sistema decidirá si usar o no, la funcionalidad de este método no fue probada, sin embargo, a las funciones que verifican que estos valores son correctos o si son válidos (funciones creadas por nosotros), sí se les hicieron las pruebas correspondientes para verificar su funcionalidad y correctitud. De igual manera se hicieron las pruebas unitarias de todas las funciones creadas por nosotros.

5.2 Informe de pruebas de integración

En cuanto a las pruebas de integración se tomó la decisión de realizarlas una vez se hubieran realizado las pruebas unitarias correspondientes, de esta manera se garantizaba que los errores que ocurriesen no se debían a errores en la funcionalidad sino a las relaciones entre ellas.

Muchas de las pruebas de integración están conformadas por funciones que hacen llamadas a otras funciones y que por lo tanto utilizan el resultado de éstas para ellas poder dar su resultado.

A continuación se presenta una pequeña tabla informativa acerca del total de número de pruebas, tanto unitarias como de integración, que se realizaron para este proyecto. Cabe mencionar que en el archivo de pruebas llamado *test.py* se encuentran todas las pruebas realizadas y se encuentran debidamente marcadas cada una con un comentario que podría ser caso borde, normal o malicia.

Bordes	Normal	Malicia	Total
41	12	58	105

Capítulo 6

Métodos, técnicas y herramientas utilizadas

6.1 Métodos

Los modelos ágiles de software se refieren a los métodos de ingeniería de software que se fundamentan en el desarrollo iterativo, donde los requisitos y soluciones se incrementan mediante la participación de los miembros del equipo. Existe una gran variedad de modelos ágiles, la mayoría se basa en el desarrollo de software en un periodo corto de tiempo. Estos periodos de tiempo se les llaman iteración, donde se debe desarrollar parte del producto que sea funcional, que incluya sus pruebas y documentos, para así obtener un demo, sin errores, al final de cada iteración.

6.1.1 Programación Extrema (XP)

La programación extrema también conocida como XP (eXtreme Programming) es el más destacado modelo de desarrollo ágil de software. Este modelo integra ciertas prácticas entre las cuales se puede mencionar algunas de las utilizadas en la realización de este proyecto:

- Programación dirigida por casos de prueba.
- Refactorización del código.
- Programación por pares.
- Incorporación del cliente de manera permanente.
- Entregas frecuentes y completas de partes del producto final.
- Integración continua del software.

El modelo ágil XP, se encuentra basado principalmente en los cuatro siguientes valores de la programación: Comunicación, simplicidad, retroalimentación y valentía.

- **Comunicación:** La comunicación durante el proceso de desarrollo debe ser abierta y continua, tanto entre los desarrolladores como entre los desarrolladores y el cliente, o con el representante del cliente. Este punto se puede ver reflejado perfectamente en todas las reuniones realizadas del equipo, al igual que en las reuniones semanales con el cliente.
- **Simplicidad:** La simplicidad se refiere a que el código del software se debe realizar lo más simple posible. Esto también se puede observar a través del proceso de refactorización que se llevó a cabo a la hora de escribir el código del software.
- **Retroalimentación:** La retroalimentación es la respuesta que proporciona el programa frente a la actividad realizada por el usuario dentro del software. Es por eso que se realizan los casos de pruebas antes de programar (TDD), ya que se obtiene una retroalimentación sobre las historias de uso al ejecutar esas pruebas sobre el software que se desarrolló.
- **Valentía:** Por último se requiere de valentía para trabajar en un proyecto de programación extrema. Debido a que se trabaja a la vista de otros, se realizan pruebas de calidad del código de forma continua y se recibe retroalimentación por parte de los miembros del equipo y del cliente.

Dentro de las ventajas que se pueden encontrar en el uso de la programación extrema y la experiencia que obtuvimos al utilizar esta metodología es que estimula a la formación de equipos de alto desempeño que pueden llegar a ser muy productivos a la hora de crear software en poco tiempo, de buena calidad y que es de utilidad para el cliente. Lo que fue de gran conveniencia para la realización de este proyecto, ya que para cada semana se debía entregar partes del software que funcionara, que tuvieran sus documentos necesarios, al igual que sus pruebas unitarias que corroboran la correcta funcionalidad del mismo.

6.1.2 Scrum

El Scrum es otro modelo ágil de desarrollo de ingeniería de software , el cual se basa en un conjunto de prácticas y roles, que pueden tomar como punto de inicio en el proceso del desarrollo del proyecto. Los roles principales dentro el método de Scrum son:

- **ScrumMaster:** Persona encargada de mantener los procesos y que trabaja de forma parecida al director del proyecto. Para éste proyecto el ScrumMaster de nuestro equipo, Paradigm Shift, fue el profesor Alfonso Reinoza.
- **ProductOwner:** Persona que representa a los afectados (Stakeholders). Para este proyecto el que tomó el rol de los stakeholders fue el profesor Alejandro Teruel.
- **Team:** Conjunto de personas que se encarga del desarrollo del software, que para este proyecto vendríamos siendo nosotros, el equipo de Paradigm Shift.

El Sprint es el periodo en el cual se lleva a cabo la realizacion del software, también conocidas como iteraciones. El periodo de iteración para este proyecto fue de una semana.

En cuanto a las reuniones de Scrum se pueden encontrar las siguientes:

- **Daily Scrum:** Cada día de un sprint, se realiza la reunión sobre el estado del proyecto. Estas reuniones fueron realizadas los días viernes en el laboratorio.
- **Scrum de Scrum:** Reuniones realizadas cada día, normalmente se realizan después del Daily Scrum en donde se discute el proyecto, enfocándose especialmente en las áreas de integración. Fueron realizadas casi todos los días, algunos días nos pudimos reunir y discutir en persona, pero la mayoría de las veces las realizamos a través de *Hangouts*. Por otro lado, utilizamos *Trello* para llevar un seguimiento de las tareas que había que realizar durante la semana, a quién le correspondía y cuándo la estaban realizando, y finalmente, al terminirlas se movían a la lista de tareas completadas.

- **Sprint Planning Meeting:** Reuniones realizadas al inicio de cada ciclo de Sprint o Iteración, que para nuestro caso fue cada semana en el laboratorio, en donde el equipo (Team) junto con el ScrumMaster y el cliente definen el trabajo a realizar para la siguiente semana y también se va desglosando poco a poco las siguientes funcionalidades a realizar para las próximas entregas.
- **Sprint Review Meeting:** Son las reuniones donde se revisa el trabajo completado o no completado de la iteración anterior, esta reunión también fue realizada en las horas del laboratorio.
- **Sprint Retrospective:** Son las reuniones realizadas después de cada sprint, en donde todos los miembros del equipo dejan sus impresiones sobre el sprint recién superado. Existen diferentes métodos de retrospectiva pero nuestro equipo utilizó la retrospectiva clásica que se basa en escribir en un papel los aspectos positivos que logramos y los que debemos mejorar para la próxima iteración.

Dentro de los beneficios que ofrece el Scrum, unido a la experiencia conseguida con la utilización de esta metodología es que, al igual que en XP, se crean equipos auto-organizados y funcionales, donde existe una buena comunicación entre todos sus miembros y se logra entregar un producto de buena calidad en un periodo corto de tiempo. También reduce los riesgos debido a que se realizan las funcionalidades de mayor prioridad de primero.

6.1.3 XP vs. Scrum

Para la realización de este proyecto se utilizaron tanto el método de XP como el SCRUM, por lo que se decidió mostrar una especie de tabla comparativa con algunos de los aspectos diferentes entre estas dos metodologías y finalmente indicar cuál de estos aspectos fueron utilizados.

XP	Scrum	Utilizada
Las iteraciones son de 1 a 3 semanas (Más rápidas).	Las iteraciones son de 2 a 4 semanas y se conocen como sprint.	Se utilizaron los Sprints del Scrum pero con el tiempo de duración del XP (1 semana).
Las tareas culminadas son susceptibles a modificaciones en el transcurso de proyecto.	Al culminar con el sprint, las tareas realizadas no se modifican.	Se utilizó el método del XP debido a que se tuvieron que agregar más pruebas a las entregas anteriores.
El equipo de desarrollo sigue estrictamente las prioridades de las tareas que definió el cliente.	El Team trata de seguir el orden que marca el ProductOwner, pero se puede modificar este orden.	Se utilizó el método del Scrum.
Los miembros del equipo trabajan en parejas.	Cada miembro del Team trabaja de forma individual.	Se utilizó el método del XP. Y se utilizaron las mismas parejas a lo largo de todo el desarrollo.

6.2 Técnicas

Las técnicas o procedimientos utilizados en la realización de este proyecto, en base a los modelos ágiles anteriormente descritos son: la programación dirigida por casos de prueba, refactorización, elaboración de pruebas unitarias, gestión de configuraciones y versiones y por último, la realización pruebas de integración y de regresión.

6.2.1 Programación dirigida por casos de pruebas (TDD)

La programación dirigida por casos de prueba es el proceso de elaboración de código, que se convierte en una actividad que va incrementando progresivamente, ya que cada trozo de código que se escribe, se efectúa para pasar un caso de prueba especificado anteriormente. Específicamente, en primer lugar se escribe una prueba y se verifica que la prueba falla. Por consecuencia, se implementa el código que hace que la prueba pase satisfactoriamente y posteriormente se refactoriza el código anteriormente escrito. La finalidad de utilizar esta técnica es lograr desarrollar un código limpio que funcione, cumpliendo con los requisitos establecidos.

Para la realización de este proyecto utilizamos esta técnica, la cuál nos fue de gran ayuda, debido a que nos permite elaborar código de manera más eficiente y rápida, y al mismo tiempo nos permite ir validando la correcta funcionalidad del mismo.

6.2.2 Refactorización

La refactorización es una técnica utilizada junto con el TDD, que permite reestructurar el código anteriormente desarrollado sin cambiar su funcionalidad. Lo que va a tener como propósito un mejor mantenimiento del código. Es por eso que se utiliza junto con la programación dirigida por casos de prueba, ya que va a facilitar el manejo del código anterior para expandir la nueva funcionalidad que se realiza.

Se utilizó esta técnica en el desarrollo del proyecto y también nos resultó de gran utilidad, debido a que si se necesitaba realizar un cambio posterior a la funcionalidad, se podría comprender mucho mejor y más rápido, por lo que nos ahorraría muchísimo tiempo en tratar de descifrar algo que no estuviese refactorizado, para poder modificarlo.

6.2.3 Pruebas unitarias

Las pruebas unitarias son un tipo de pruebas que se enfocan en hacer fallar a la funcionalidad previamente implementada, para así poder identificar sus defectos y poder solucionarlo. Esto tiene como finalidad asegurar que cada parte del sistema funcione correctamente. La idea de estos casos de prueba, es velar por la correcta funcionalidad de cada componente del software por separado, es decir no dependen del resto. Las pruebas unitarias, al igual que la refactorización, son utilizadas también en la técnica del TDD, ya que éstas vendrían siendo las pruebas que se crean antes de comenzar a desarrollar el código.

Este tipo de pruebas se utilizaron en la elaboración del proyecto, ya que nos permitían ir validando la correcta funcionalidad de la componente del software que estuviésemos implementando y así poder asegurar su correctitud.

6.2.4 Gestión de configuraciones y versiones

La gestión de configuraciones y versiones se encarga de las políticas, procesos y herramientas para administrar el sistema cambiante del producto o software. Más específicamente, la versión es la revisión registrada de un componente de un software, mientras que la configuración es el conjunto de versiones de componentes.

En relación al plan de configuraciones y versiones que utilizamos podemos observar algunos aspectos:

- **Modelo de ramas:** Para este proyecto se manejó directamente el tronco master, que tendrá la versión más actualizada del producto. Se utilizó de esta manera ya que no se vio necesario crear otras ramas distintas, debido a que cada pareja tenía una tarea distinta asignada y no iba a producirse ningún problema de merge a la hora de hacer push a esta rama master.
- **Repositorio utilizado:** Al igual que como se indica en la portada de este informe, la dirección del repositorio es <https://github.com/Sealos/SAGE>, en donde se puede encontrar la versión más reciente del proyecto SAGE ubicado específicamente en la rama master.

6.2.5 Pruebas de integración

Las pruebas de integración tienen como propósito unir todas las componentes del sistema, y en verificar y validar que estas componentes encajan apropiadamente. Las pruebas de integración al igual que otros tipos de pruebas, se encargan de buscar fallas específicamente en la unión de las componentes.

Se utilizó esta técnica debido a que al igual que las pruebas unitarias, nos permite asegurar la correctitud de software, pero esta vez en relación a la unión de las componentes del sistema. Lo que también es de gran ayuda porque nos permite demostrarle al cliente que todo el sistema está funcionando como debe.

6.2.6 Pruebas de regresión

Las pruebas de regresión tienen como finalidad descubrir errores o carencia de funcionalidad en referencia al comportamiento esperado del software, o en base a los requerimientos del cliente, que son causados por la realización de un cambio en el programa.

La realización de este tipo de pruebas, en este proyecto, se llevó a cabo en cada paso de iteración o sprint. Y nos permite validar o demostrar que los nuevos cambios introducidos al software no modifican las funcionalidades anteriormente probadas. Lo cual nos resulta beneficioso, para así probarle al cliente que se le está ofreciendo un producto de buena calidad.

6.3 Herramientas

Las herramientas son diversas aplicaciones a nivel de informática que nos permite aumentar la productividad en el desarrollo del software, reduciendo notablemente el tiempo de elaboración. A continuación se presentan las herramientas utilizadas para la elaboración de este proyecto.

6.3.1 Eclipse

Eclipse es una plataforma de desarrollo, compuestas por un conjunto de herramientas de programación de código abierto, diseñada para ser extendida a través de plugins. No usa un lenguaje en específico, sino que es un entorno de desarrollo integrado (IDE) genérico. Entre las principales características de eclipse encontramos:

- **Gestión de proyectos:** El desarrollo en esta herramienta se basa en la creación de proyectos que son una mezcla entre código fuente, documentación, ficheros, configuración, árbol de directorios, entre otros.
- **Depurador de código:** Depurador fácil e intuitivo que permite mejorar el código a partir de la visualización de los fallos que hay luego de depurarlo.
- **Plugins:** Extensa cantidad de plugins que permiten una mejor manipulación de la herramienta. Hay una gran cantidad de plugins tanto gratuitos como pagos.

Al principio, como no se había utilizado esta herramienta antes en la carrera, tuvimos varios problemas instalando las cosas necesarias para el correcto funcionamiento (por ej. PyDev), luego también tuvimos ciertos problemas con las pruebas unitarias cuando se empezó a utilizar el Framework Django para la realización de este proyecto. Poco a poco nos hemos ido familiarizando con esta herramienta aprendiendo a utilizar las distintas funcionalidades que ésta posee.

6.3.2 Python

Python es un lenguaje de programación interpretado que soporta orientación a objetos, programación imperativa y programación funcional. Dentro de los beneficios que tiene la utilización de Python podemos encontrar que nos simplifica la programación y por otro lado, que es bastante rápido y sencillo de aprender. También es flexible, ordenado y limpio, además que cuenta con muchas librerías de las cuales las más importantes y útiles están dentro del código.

Con todo esto podemos decir que Python es un buen lenguaje para la realización de este proyecto, ya que al ser fácil de aprender, no se gasta tanto tiempo en su estudio, lo que nos ayuda a realizar mejores proyectos en una menor cantidad de tiempo, y esto es un punto muy importante ya que con solo 12 semanas se tiene que aprovechar al máximo aprendizaje de cada lenguaje.

6.3.3 PyUnit

PyUnit, forma parte de una familia de herramientas conocidas como xUnit, son un conjunto de frameworks creados por Kent Beck. El uso de esta herramienta es bastante sencillo y permite crear casos de pruebas de acuerdo a la clase que herede del código.

Al ejecutar cada una de estas pruebas se puede obtener como resultado OK que significa que se ha pasado con éxito, FAIL las pruebas no se han pasado con éxito y se muestra una excepción AssertionError, y ERROR que es una excepción distinta de AssertionError.

6.3.4 Git

Git es una herramienta de control de versiones diseñado para generar eficiencia y confiabilidad en el área de mantenimiento de versiones de software, específicamente cuando éstas tienen un gran número de archivos de código.

Para la elaboración de este proyecto se utilizó GitHub para poder descargar y actualizar los archivos que contienen las funcionalidades de SAGE, de manera tal que todos pudiésemos acceder a las versiones más recientes del mismo.

6.3.5 Django

Django es un framework de desarrollo web y que está basado en el modelo vista controlador. El propósito principal de Django es el de facilitar la creación de sitios de web que son complejos.

La utilización de Django fue un poco compleja al inicio debido a que no teníamos ningún conocimiento referente a él. Pero una vez comprendido como éste funcionaba, se nos facilitó mucho a la hora de poder implementar las funcionalidades, específicamente en la creación de la base de datos y de las respectivas pruebas unitarias, ya que Django nos ofrece también una herramienta para la realización de las pruebas a las bases de datos. Por otro lado, la implementación de la interfaz fue mucho más sencilla de trabajar y de poder integrar todas las vistas de las funcionalidades en la misma página.

Capítulo 7

Conclusiones

Para finalizar este informe se puede decir en cuanto al uso de herramientas como Eclipse, Git, Django y el aprendizaje del lenguaje Python, son fundamentales para todo futuro Ingeniero en Computación ya que no sólo es de aprendizaje sencillo sino que también, cada vez más están siendo utilizadas en el mercado tecnológico.

Por otro lado, en cuanto a la utilización de la metodología de Scrum nos permitió aprender de manera rápida y práctica, métodos de trabajo que son usados actualmente en grandes empresas. En estas 6 semanas nos enfrentamos a mucha programación continua con flexibilidad en el proceso y en la definición del producto final, también al proceso de realimentación constante con el cliente sobre el producto, ya que se mostraban cada semana avances del mismo (en cuanto a funcionalidad) y se obtenían críticas constructivas para realizar la semana siguiente. Todo esto en constante interacción y comunicación grupal, lo que es clave en metodologías como estas, debido a que cada miembro del grupo es importante en cada actividad que se realiza, si se quiere lograr la entrega de un proyecto excelente y completo.

Finalmente, la realización de este proyecto en equipo fue una buena experiencia, ya que fue la primera vez que todos trabajamos en esta modalidad, debido a que en la mayoría de los laboratorios se nos exigía trabajar en pareja. Aprendimos un poco a cómo repartirnos las actividades de manera equitativa, al igual que a cómo organizarnos para poder intentar realizar el mayor número de actividades posibles en las horas establecidas.

Capítulo 8

Recomendaciones

Después de haber pasado por esta gran experiencia, podemos mencionar algunas recomendaciones que pueden ser de mucha ayuda:

- Si se va a comenzar un proyecto nuevo que puede necesitar una base de datos, se sugiere implementar enseguida, debido a que facilita mucho la creación de las pruebas.
- Otra recomendación sería que a medida que se vayan creando las pruebas, se vayan documentando para no repetirlas accidentalmente. Al igual que ir comentando el código su justificación (incluido por ser frontera, esquina o malicia).
- Algo muy importante es no dejar la documentación para el final, debido a que se pueden perder muchos aspectos importantes, y es el documento el que posee más valor, debido a que él se encuentra toda la información referente a la codificación y estructura del sistema.
- A medida que se vayan creando funcionalidades es recomendable documentarlas y reflejarlas en los diagramas que se vayan a hacer.
- Manejar la herramienta Git muy bien, ya que no sólo se usa en esta materia y los beneficios que aporta son de gran ventaja para trabajar de manera grupal, también es importante mencionar la facilidad de recuperar versiones anteriores en caso de fallos.
- Recomendamos fuertemente el uso de la herramienta *Trello* (<https://trello.com>) que les permitirá una mejor organización a la hora de repartir las tareas y saber a quién le corresponde realizarlas y evitar que dos personas realicen la misma actividad o queden actividades sin realizar.
- Realizar la refactorización es algo necesario, ya que sin ella sería mucho más complicado, para otra persona que no escribió el código, comprender en poco tiempo la funcionalidad implementada. Y no sólo eso, sino que también ayuda a la hora de realizar una modificación al programa, se puede visualizar mucho más rápido.
- Por último, como equipo les podemos decir, que lo más importante es que exista una buena comunicación entre los miembros del equipo y que traten de no dejar las tareas para última hora, para eso es necesario una buena organización.

Bibliografía

- Decanato de Estudios Profesionales. 2013. Normas Generales para la Redacción y Presentación de Grado y Pasantías Largas e Intermedias. http://www.profesionales.usb.ve/sites/default/files/Archivos_Normas/Normas_DEP_libro_actual.doc, consultado el 15 de Noviembre de 2014.
- Abad, Soraya. 2010. Lineamientos sobre cómo escribir informe técnicos. Disponible en Internet: <http://cpc ldc.usb.ve/documentos/comoEscribirInf07.pdf>, consultado el 15 de Noviembre de 2014.
- WordPress. Fecha desconocida. Software educativo para la enseñanza - Retroalimentación. Disponible en Internet: <http://modeducativos.wordpress.com/retroalimentacion/>, consultado el 16 de Noviembre de 2014.
- Editor. 2011. Tipos de de equipo. Disponible en Internet: <http://www.si-educa.net/basico/ficha245.html>, consultado el 16 de Noviembre de 2014.
- Alejandro Teruel. 2013. Modelos ágiles de desarrollo. Disponible en Internet: <https://sites.google.com/site/ingsoftware1teruel/modelos-agiles>, [Modelos agile.pdf], consultado el 16 de Noviembre de 2014.
- Wikipedia. 2014. Scrum. Dispñible en Internet: <http://es.wikipedia.org/wiki/Scrum>, consultado el 16 de Noviembre de 2014.
- Galvin, Macias, Torres y Videras. 2012. Diferencias entre Scrum y XP. Disponible en Internet: <http://es.slideshare.net/deborahgal/diferencias-entre-scrum-y-xp-12219336>, consultado el 16 de Noviembre de 2014.
- Alejandro Teruel. 2014. Taller 6. Disponible en Internet: <https://sites.google.com/site/ingsoftware1teruel/talleres>, [Taller Intro a Scrum y Equipos de Alto Desempeño.pdf], consultado el 16 de Noviembre de 2014.
- Alejandro Teruel. 2014. Taller 7. Disponible en Internet: <https://sites.google.com/site/ingsoftware1teruel/talleres>, [Reuniones Scrum(pre-sprint, retrospectiva, planificación de sprint).pdf], consultado el 16 de Noviembre de 2014.
- ISO/IEC. 2000. Information technology — Software product quality. Part 1: Quality model. Disponible en Internet:

https://moodle.asignaturas.usb.ve/pluginfile.php/28547/mod_resource/content/1/9126-1_Standard.pdf, consultado el 17 de Noviembre de 2014.

- Wikipedia. 2014. Git. Disponible en Internet: <http://es.wikipedia.org/wiki/Git>, consultado el 21 de Noviembre de 2014.
- Alejandro Teruel. 2014. Programación dirigida por casos de prueba. Disponible en Internet: <https://sites.google.com/site/ingsoftware1teruel/home/programacion>, [Programación dirigida por casos de prueba(TDD).pdf], consultado el 23 de Noviembre de 2014.
- Alejandro Teruel. 2014. Gestión de Configuraciones y Versiones de Software. Disponible en Internet: <https://sites.google.com/site/ingsoftware1teruel/home/gestion-de-configuraciones>, [Gestión de Configuraciones y Versiones del Software 2014.pdf], consultado el 23 de Noviembre de 2014.
- Alejandro Teruel. 2014. Pruebas de Integración. Disponible en Internet: <https://sites.google.com/site/ingsoftware1teruel/home/verificacion>, Pruebas de Integración.pdf], consultado el 23 de Noviembre de 2014.
- Wikipedia. 2014. Pruebas de regresión. Disponible en Internet: http://es.wikipedia.org/wiki/Pruebas_de_regresi%C3%B3n, consultado el 23 de Noviembre de 2014.
- Wikipedia. 2014. Eclipse. Disponible en Internet: [http://es.wikipedia.org/wiki/Eclipse_\(software\)](http://es.wikipedia.org/wiki/Eclipse_(software)) consultado el 23 de Noviembre de 2014.
- Wikipedia. 2014. Python. Disponible en Internet: <http://es.wikipedia.org/wiki/Python> consultado el 23 de Noviembre de 2014.
- Alejandro Teruel. 2013. La Ingeniería de Software y la ley en Venezuela. Disponible en Internet: <https://sites.google.com/site/ingsoftware1teruel/etica>, [La Ley y la Ingeniería del Software.pdf], consultado el 24 de Noviembre de 2014.
- Alejandro Teruel. 2013. Ética en la Ingeniería de Software. Disponible en Internet: <https://sites.google.com/site/ingsoftware1teruel/etica>, [Ética en Ingeniería de Software2.pdf], consultado el 24 de Noviembre de 2014.