**Introducción**

El presente documento corresponde al documento principal de nuestro trabajo final de carrera, el cual nos permitirá obtener el título de Ingenieros en Sistemas de la Universidad Nacional de La Rioja. En éste se detallarán los aspectos principales de nuestro trabajo final de carrera el cual está apuntado al Instituto de Formación Docente “Dr. José S. Salinas” de la Localidad de Olta, Provincia de La Rioja.

El sistema está desarrollado con el propósito de brindarle a la institución escolar seleccionada una herramienta que les permita administrar la gestión de los alumnos en forma segura, con la finalidad de obtener información consistente para los niveles operativos y directivos. Los usuarios potenciales de este sistema son aproximadamente 400 alumnos ya inscriptos en este instituto formador.

El objetivo del proyecto es crear un sistema de información autosustentable buscando mejorar los procesos académicos, hacerlos más eficientes, colaborar en la toma de decisiones, con el análisis institucional y con la transparencia de la información en general.

El proceso de elaboración de nuestro Trabajo Final comenzó con un análisis preliminar de las necesidades concretas de la institución, del entorno en el que debiera funcionar el sistema y la razón por la cual sería útil el desarrollo del mismo. También se llevó a cabo un análisis de factibilidad que nos permitiera saber si era conveniente avanzar con el proyecto.

Describir la estructura del proyecto.

**Antecedentes**

En una primera instancia se realizó una investigación sobre las causas que originan la necesidad de llevar a cabo un eficaz [sistema](http://www.monografias.com/trabajos11/teosis/teosis.shtml) de Inscripción en la institución y las [herramientas](http://www.monografias.com/trabajos11/contrest/contrest.shtml) necesarias para erradicar los [problemas](http://www.monografias.com/trabajos15/calidad-serv/calidad-serv.shtml#PLANT) más comunes en cuanto a los [procesos](http://www.monografias.com/trabajos14/administ-procesos/administ-procesos.shtml#PROCE) de inscripción [manual](http://www.monografias.com/trabajos13/mapro/mapro.shtml).

En este sentido se entiende como Sistema de Inscripción el [procedimiento](http://www.monografias.com/trabajos13/mapro/mapro.shtml) a seguir al inicio de cada año escolar, en el Instituto Superior de Formación Docente “Dr. José S. Salinas” el sistema de inscripción es manual. A partir del relevamiento y análisis de la información se pudo detectar que la mayoría de los [procedimientos](http://www.monografias.com/trabajos13/mapro/mapro.shtml) tienen inconvenientes, entre ellos se pueden destacar: pérdida o duplicación de información debido al proceso manual y a la forma de [almacenamiento](http://www.monografias.com/trabajos12/dispalm/dispalm.shtml), retardos en el proceso de inscripción y comienzo de las clases, y actividades irregulares por la tardanza del proceso.

Los procesos rutinarios en forma manual procesan información lenta, llevarla a un sistema automatizado garantiza un mejor trato de información.

Cabe agregar que entre las dificultades de los estudiantes de la institución en el momento de la inscripción ya mencionadas anteriormente, el 40% de ellos viven zonas aledañas a la ciudad, teniendo que viajar no solamente para asistir a clases y a exámenes, sino que también para realizar cualquier trámite administrativo.

Se observó que la falta de esta tecnología en el Instituto demuestra lo ineficaz del proceso de inscripción, funcionando en forma manual y originando alto grado de desorganización, lentitud en la realización de reportes, duplicación de inscripciones y demoras en el proceso que a su vez ocasiona agotamiento del personal.

El proceso de inscripción se lleva de forma manual, donde se verifica la totalidad de los [documentos](http://www.monografias.com/trabajos14/comer/comer.shtml), se recolectan los datos a través de una ficha denominada Ficha De Inscripción, otorgada por el bedel,  con el fin de crear Legajos de los alumnos y se la anexan los documentos exigidos (Fotocopia de partida de nacimiento, certificado de buena [salud](http://www.monografias.com/Salud/index.shtml), certificado de notas, fotocopia de DNI, certificado de domicilio, 2 fotografías fotocopia y por último fotocopia del título de secundario o en su defecto título secundario en trámite).

**El instituto superior de formación docente**

**BUSCAR HISTORIA MISION VISIÒN ALGO QUESEYO MANDAR ALGUNA VERDURA**

**Situación Problemática**

Mediante la aplicación de las diferentes técnicas de recolección de datos, realizamos un análisis de la situación actual del ISFD, encontramos que el proyecto tiene pertinencia debido a:

* La falta de un espacio de comunicación entre la institución, los estudiantes y los profesores.
* Los fuertes casos de deserción por parte los estudiantes de zonas alejadas al ISFD.
* La carga excesiva de trabajo en épocas de inscripciones para los bedeles.
* Depósitos de expedientes apilados en formato papel, ocupando gran cantidad de espacio y propenso a acumulación de roedores y mucha humedad.
* La manipulación de los libros de actas se tornan muy frágiles.

El impacto de este proyecto traerá no solamente ventajas al proceso de inscripción, sino que también proporcionará beneficios a otros ámbitos no incluidos en las actividades principales del sistema.

**Propuesta de Solución**

Se propone como alternativa de solución desarrollar un sistema automatizado para la gestión de los procesos académicos del ISFD. Este desarrollo, cuyo nombre es EGA (Entorno de Gestión Académica), está constituido por los siguientes módulos principales:

* Módulo de preinscripción de alumnos a las distintas carreras, agilizando los trámites correspondientes a la inscripción
* Módulo que permita generar permisos de exámenes finales.
* Módulo para inscripción a los diferentes espacios curriculares.
* Módulo que permita la descarga de acta volante para los profesores.
* Módulo que autorice los permisos necesarios para llevar a cabo la administración del sistema.
* Módulo de administración que contemple las tareas correspondientes al bedel.

**Denominación del Proyecto**

*Entorno de Gestión Académico.*

(EGA).



**Áreas de conocimiento**

1. **Programación.**
2. **Sistemas y Organización.**
3. **Bases de Datos.**
4. **Redes de Datos.**
5. **Actualizaciones Tecnológicas.**
6. **Administración.**
7. **Derecho Informático.**

**Objetivos**

**Objetivo General**

El objetivo de nuestro Trabajo Final es el de desarrollar e implementar un sistema de información para organizar, sistematizar y controlar la gestión administrativa de toda la información académica del Instituto Superior de Formación Docente. “Dr. José S. Salinas” de Olta, Dpto. General Belgrano Provincia de La Rioja en el año 2014.

**Objetivos Específicos**

Entre los objetivos específicos se encuentran:

* Desarrollar e Implementar módulo de preinscripción de alumnos a las distintas carreras, agilizando los trámites correspondientes a la inscripción.
* Desarrollar e implementar módulo que permita generar permisos de exámenes finales.
* Desarrollar e Implementar módulo para inscripción a los diferentes espacios curriculares.
* Desarrollar e Implementar módulo que permita la descarga de acta volante para los profesores.
* Desarrollar e Implementar módulo que autorice los permisos necesarios para llevar a cabo la administración del sistema.
* Desarrollar e Implementar módulo de administración que contemple las tareas correspondientes al bedel.

**Alcances**

El alcance del sistema dentro de la cobertura será:

* Agilizar trámites de inscripción a las diferentes carreras.
* Registrar datos de inscripciones de los estudiantes a las distintas carreras.
* Generar reporte sobre la inscripción de los estudiantes a los distintos espacios curriculares.
* Emitir informes a los docentes sobre la cantidad de alumnos a rendir en cada mesa de examen final, como así también de inscriptos para el cursado de las materias.
* Presentar listado de materias rendidas y regularizadas de cada estudiante.
* Proporcionar información rápida cuando los diferentes actores institucionales lo requieran.

Como productos entregables:

* El Sistema para gestión académica de la institución propiamente dicho y su código fuente.
* E.G.A contará con un sistema de ayuda que sirva como guía a los usuarios.
* La capacitación para el correcto uso de E.G.A contará con un manual de usuarios redactado de manera comprensible, incluyendo gráficos e imágenes de la interface del sistema.
* Se entregará a su vez el manual de sistema es un extracto de todos los documentos producidos durante el proceso de desarrollo del proyecto.

**Metodología**

Mediante la aplicación de las diferentes técnicas de recolección de datos, realizadas conjuntamente con los stakeholder, capturamos los requerimientos de nuestro cliente. Debido a la problemática observada, como así también la tecnología existente en el medio y la normativa vigente plasmada en el reglamento académico institucional, y a partir de nuestra investigación realizada y los criterios de factibilidad considerados, observamos pertinente la aplicación de la metodología para el proceso de desarrollo del software denominada Proceso Unificado Racional (también conocido como RUP) simultáneamente con el Lenguaje Unificado de Modelado (UML), ya que conjuntamente constituyen la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

**Análisis de Factibilidad**

**Factibilidades Operativas**

Teniendo en cuenta que los encargados de administrar el sistema (bedeles) no cuentan con conocimientos informáticos, EGA se desarrollará teniendo en cuenta los criterios de usabilidad, brindándoles capacitaciones sobre las funcionalidades del mismo, conjuntamente con el manual de usuario correspondiente.

A partir de esta acción, se garantiza la adecuada utilización y funcionamiento del sistema a implementar.

**Factibilidades Técnicas**

Nosotros estudiantes de la carrera Ingeniería en Sistemas, y encargados de diseño, implementación, mantenimiento y control del sistema de información, consideramos estar suficientemente capacitados para llevar a cabo el desarrollo del sistema de gestión propuesto. En particular, debido a que poseemos los conocimientos suficientes, los cuales nos fueron brindados en el transcurso de nuestra carrera universitaria, y considerando además la tecnología existente en el campo informático, como así también la disponible en el ISFD. Podemos por lo tanto afirmar que contamos con los recursos técnicos necesarios para llevar a cabo este proyecto.

**Factibilidades Económicas**

El I.S.F.D Dr. José S. Salinas cuenta con los medios económicos necesarios para llevar a cabo el desarrollo e implementación del sistema de información.

* Agilidad y facilidad en los trámites realizados por el alumno.
* Agilidad y facilidad en las operaciones a realizar por parte del profesor.
* Mayor organización y control por parte del ISFD “Dr. José S. Salinas”.

**Capitulo 2: Marco Teórico**

**Introducción**

Carlos Sabino afirma que "el planteamiento de una investigación no puede realizarse si no se hace explícito aquello que nos proponemos conocer: es siempre necesario distinguir entre lo que se sabe y lo que no se sabe con respecto a un tema para definir claramente el problema que se va a investigar". El correcto planteamiento de un problema de investigación nos permite definir sus objetivos generales y específicos, como así también la delimitación del objeto de estudio.

El marco teórico, marco referencial o marco conceptual tiene el propósito de dar a la investigación un sistema coordinado y coherente de conceptos y proposiciones que permitan abordar el problema. Se trata de integrar al problema dentro de un ámbito donde éste cobre sentido, incorporando los conocimientos previos relativos al mismo y ordenándolos de modo tal que resulten útil a nuestra tarea.

El fin que tiene el marco teórico es el de situar a nuestro problema dentro de un conjunto de conocimientos, que permita orientar nuestra búsqueda y nos ofrezca una conceptualización adecuada de los términos que utilizaremos.

Ezequiel Ander-Egg nos dice que en el marco teórico o referencial "se expresan las proposiciones teóricas generales, las teorías específicas, los postulados, los supuestos, categorías y conceptos que han de servir de referencia para ordenar la masa de los hechos concernientes al problema o problemas que son motivo de estudio e investigación". En este sentido, "todo marco teórico se elabora a partir de un cuerpo teórico más amplio, o directamente a partir de una teoría. Para esta tarea se supone que se ha realizado la revisión de la literatura existente sobre el tema de investigación. Pero con la sola consulta de las referencias existentes no se elabora un marco teórico: éste podría llegar a ser una mezcla ecléctica de diferentes perspectivas teóricas, en algunos casos, hasta contrapuestas. El marco teórico que utilizamos se deriva de lo que podemos denominar nuestras opciones apriorísticas, es decir, de la teoría desde la cual interpretamos la realidad".

Roberto Hernández Sampieri y otros destacan las siguientes funciones que cumple el marco teórico dentro de una investigación:

1. Ayuda a prevenir errores que se han cometido en otros estudios.

2. Orienta sobre cómo habrá de realizarse el estudio (al acudir a los antecedentes, nos podemos dar cuenta de cómo ha sido tratado un problema específico de investigación, qué tipos de estudios se han efectuado, con qué tipo de sujetos, cómo se han recolectado los datos, en qué lugares se han llevado a cabo, qué diseños se han utilizado).

3. Amplía el horizonte del estudio y guía al investigador para que se centre en su problema, evitando desviaciones del planteamiento original.

5. Inspira nuevas líneas y áreas de investigación.

6. Provee de un marco de referencia para interpretar los resultados del estudio.

Para la Bibliografía

SABINO, Carlos, El proceso de investigación, Lumen-Humanitas, Bs.As., 1996..

ANDER-EGG, Ezequiel, Técnicas de Investigación Social, Humanitas, Bs.As., 1990.

HERNANDEZ SAMPIERI, R., FERNANDEZ COLLADO, C. y BAPTISTA LUCIO, P., Metodología de la Investigación, McGraw Hill, México, 2000.

TAMAYO, L. y TAMAYO, M., El proceso de la investigación científica, Limusa S.A., México, 1998

**Introducción (ver título)**

Internet y la Web han influido enormemente tanto en el mundo de la informática como en la sociedad en general. Si nos centramos en la Web, en poco menos de 10 años ha transformado los sistemas informáticos: ha roto las barreras físicas (debido a la distancia), económicas y lógicas, (debido al empleo de distintos sistemas operativos, protocolos, etc.) y ha abierto todo un abanico de nuevas posibilidades. Una de las áreas que más expansión en la Web son las llamadas “webapps”. Estas permiten la generación automática de contenido, la creación de páginas personalizadas según el perfil del usuario o del comercio; además permite la interacción con otros sistemas informáticos dentro de la gestión de una organización.

Esta categoría de software centrado en redes agrupa una amplia gama de aplicaciones. En su forma más sencilla, las webapps son poco más que un conjunto de archivos de hipertexto vinculados que presentan información con uso de texto y gráficas. Sin embargo, desde que surgió Web 2.0, las webapps están evolucionando hacia ambientes de cómputo sofisticados que no sólo proveen características aisladas, funciones de cómputo y contenido para el usuario final, sino que también están integradas con bases de datos corporativas y aplicaciones complejas de negocio.

En este caso, el término aplicación web (o webapps) agrupa todo, desde una simple página web que ayude al consumidor a calcular un pago, hasta un sitio web integral que, por ejemplo, proporcione servicio completo para viajes para gente de negocios o vacacionistas. En esta categoría se incluyen sitios web complejos, con funcionalidad especializada dentro de sitios web y aplicaciones de procesamiento de información que residen en internet o en una intranet o extranet.

**La Naturaleza única de las webapps**

En los primeros días de la Red Mundial (entre 1990 y 1995) los sitios web consistían en un poco más que un conjunto de archivos de hipertexto vinculados que presentan información con uso de texto y gráficas. Al pasar el tiempo, el aumento de HTML por medio de herramientas de desarrollo, permitió a los ingenieros web bridar capacidad de cómputo junto con contenido de información. Habían nacido los sistemas y aplicaciones basados en la web (se denominó a estas en forma colectiva como *webapps*). En la actualidad, las webapps se han convertido en herramientas sofisticadas de cómputo que no sólo proporcionan funciones aisladas al usuario final, sino que también se han integrado con bases de datos corporativas y aplicaciones de negocios.

Las webapps son una de la varias categorías distintas de software. No obstante, podría argumentarse que las webapps son diferentes. Powell sugiere que los sistemas y aplicaciones basados en la web “involucran una mezcla entre las publicaciones impresas y el desarrollo de software, entre la mercadotecnia y la computación, entre las comunicaciones internas y las relaciones exteriores, y entre el arte y la tecnología”. La gran mayoría de webapps presenta los siguientes atributos:

* **Uso intensivo de las redes.**  Una webapps reside en una red y debe atender las necesidades de una comunidad de clientes. La red permite acceso y comunicación mundiales (por ejemplo, internet) o tiene acceso y comunicación limitados ( por ejemplo una intranet corporativa)
* **Concurrencia.** A las webapps puede acceder un gran número de usuarios a la vez. En muchos casos, los patrones de uso entre los usuarios finales varían mucho.
* **Carga impredecible.** El número de usuarios en la webapps cambia en varios órdenes de magnitud de un día a otro. El lunes tal vez la utilicesn cien personas, el jueves quizá mil usen el sistema.
* **Rendimiento.** Si un usuario de la webapps debe esperar demasiado (para entrar, para el procesamiento por parte del servidor, para el formado y despliegue del lado del cliente), él o ella quiźa decidan irse a otra parte.
* **Disponibilidad.** Aunque no es razonable esperar una disponibilidad de 100%, es frecuente que los usuarios de webapps populares demanden acceso las 24 horas de los 365 días del año.
* **Orientadas a los datos.** La función principal de muchas webapps es el uso de hipermedios para presentar al usuario final contenido en forma de texto, gráficas, audio y video. Además, las webapps se utilizan en forma común para acceder a información que existe en bases de datos que no son parte integral del ambiente basado en web (por ejemplo, comercio electrónico o aplicaciones financieras).
* **Contenido sensible.** La calidad y naturaleza estética del contenido constituye un rasgo importante de la calidad de una webapps.
* **Seguridad.** Debido a que las webapps se encuentran disponibles con el acceso a una red, es difícil o imporsible limitar a la población de usuarios finales que pueden acceder a la aplicación. Con el fin de proteger el contenido sensible y brindar modos seguros de trasmición de datos, deben implementarse medidas estrictas de seguridad a través de infraestructuras de apoyo.
* **Estética:** Parte innegable del atractivo de una webapps es su apariencia y percepción.

**Webapps**

En la ingeniería de software se denomina **aplicación web** a aquellas [herramientas](http://es.wikipedia.org/wiki/Aplicaci%C3%B3n_inform%C3%A1tica) que los usuarios pueden utilizar accediendo a un [servidor web](http://es.wikipedia.org/wiki/Servidor_web) a través de [Internet](http://es.wikipedia.org/wiki/Internet) o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web en la que se confía la ejecución al navegador.

Las aplicaciones web son populares debido a lo práctico del [navegador web](http://es.wikipedia.org/wiki/Navegador_web) como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales.

Es importante mencionar que una página Web puede contener elementos que permiten una comunicación activa entre el usuario y la información. Esto permite que el usuario acceda a los datos de modo interactivo, gracias a que la página responderá a cada una de sus acciones, como por ejemplo rellenar y enviar formularios, participar en juegos diversos y acceder a gestores de base de datos de todo tipo.

**Antecedentes**

En los primeros tiempos de la computación cliente-servidor, cada aplicación tenía su propio programa cliente que servía como interfaz de usuario que tenía que ser instalado por separado en cada ordenador personal de cada usuario. El cliente realizaba peticiones a otro programa -el servidor- que le daba respuesta. Una mejora en el servidor, como parte de la aplicación, requería normalmente una mejora de los clientes instalados en cada ordenador personal, añadiendo un coste de soporte técnico y disminuyendo la productividad.

A diferencia de lo anterior, las aplicaciones web generan dinámicamente una serie de páginas en un formato estándar, como HTML o XHTML, soportados por los navegadores web comunes. Se utilizan [lenguajes interpretados](http://es.wikipedia.org/wiki/Lenguaje_interpretado) en el lado del cliente, directamente o a través de plugins tales como JavaScript, Java, Flash, etc., para añadir elementos dinámicos a la interfaz de usuario. Generalmente cada página web en particular se envía al cliente como un documento estático, pero la secuencia de páginas ofrece al usuario una experiencia interactiva. Durante la sesión, el navegador web interpreta y muestra en pantalla las páginas, actuando como cliente para cualquier aplicación web.

## Estructura de las aplicaciones web

Aunque existen muchas variaciones posibles, una aplicación web está normalmente estructurada como una aplicación de tres-capas. En su forma más común, el navegador web ofrece la primera capa, y un motor capaz de usar alguna tecnología web dinámica, por ejemplo: PHP, Java Servlets o ASP,ASP.NET, CGI, ColdFusion, embPerl, Python o Ruby on Rails que constituye la capa intermedia. Por último, una base de datos constituye la tercera y última capa.

El navegador web manda peticiones a la capa intermedia que ofrece servicios valiéndose de consultas y actualizaciones a la base de datos y a su vez proporciona una interfaz de usuario.

## Ventajas

- **Ahorra tiempo**: Se pueden realizar tareas sencillas sin necesidad de descargar ni instalar ningún programa.

- **No hay problemas de compatibilidad**: Basta tener un navegador actualizado para poder utilizarlas.

- **No ocupan espacio** en nuestro disco duro.

- **Actualizaciones inmediatas**: Como el software lo gestiona el propio desarrollador, cuando nos conectamos estamos usando siempre la última versión que haya lanzado.

- **Consumo de recursos bajo**: Dado que toda (o gran parte) de la aplicación no se encuentra en nuestro ordenador, muchas de las tareas que realiza el software no consumen recursos nuestros porque se realizan desde otro ordenador.

- **Multiplataforma**: Se pueden usar desde cualquier sistema operativo porque sólo es necesario tener un navegador.

- **Portables**: Es independiente del ordenador donde se utilice (un PC de sobremesa, un portátil...) porque se accede a través de una página web (sólo es necesario disponer de acceso a Internet). La reciente tendencia al acceso a las aplicaciones web a través de teléfonos móviles requiere sin embargo un diseño específico de los ficheros CSS para no dificultar el acceso de estos usuarios.

- **La disponibilidad suele ser alta** porque el servicio se ofrece desde múltiples localizaciones para asegurar la continuidad del mismo.

- **Los virus no dañan** los datos porque éstos están guardados en el servidor de la aplicación.

- **Colaboración**: Gracias a que el acceso al servicio se realiza desde una única ubicación es sencillo el acceso y compartición de datos por parte de varios usuarios. Tiene mucho sentido, por ejemplo, en aplicaciones online de calendarios u oficina.

- Los navegadores ofrecen **cada vez más y mejores funcionalidades** para crear aplicaciones web ricas..

## Inconvenientes

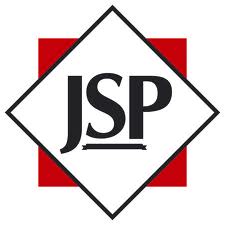
- Habitualmente ofrecen **menos funcionalidades que las aplicaciones de escritorio**. Se debe a que las funcionalidades que se pueden realizar desde un navegador son más limitadas que las que se pueden realizar desde el sistema operativo.

- **La disponibilidad depende de un tercero**, el proveedor de la conexión a internet o el que provee el enlace entre el servidor de la aplicación y el cliente. Así que la disponibilidad del servicio está supeditada al proveedor.

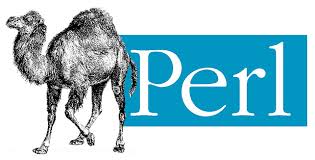
## Lenguajes de programación

Existen numerosos lenguajes de programación empleados para el desarrollo de aplicaciones web en el servidor, entre los que destacan:

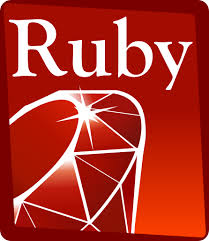
* PHP
* Java, con sus tecnologías Java Servlets y JavaServer Pages (JSP)



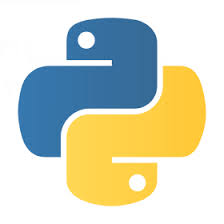
* Javascript



* Perl



* Ruby



* Python



* C#



* Visual Basic con sus tecnologías [ASP](http://es.wikipedia.org/wiki/Active_Server_Pages)/ASP.NET

También son muy utilizados otros lenguajes o arquitecturas que no son propiamente lenguajes de programación, como HTML o XML.

Se utilizan para servir los datos adecuados a las necesidades del usuario, en función de cómo hayan sido definidos por el dueño de la aplicación. Los datos se almacenan en alguna [base de datos](http://es.wikipedia.org/wiki/Base_de_datos) estándar.

**Calidad de una aplicación web**

Usabilidad

Funcionalidad

Calidad de una aplicación Web

Confiabilidad

Eficiencia

Facilidad para recibir mantenimiento

**METODOLOGÍA DE DESARROLLO**

Para llevar a cabo este desarrollo informático se selecciona como metodología a seguir como guía, RUP (del ingles Rational Unificate Proces) en español Proceso Unificado de Rational.

**Introducción**

La primera sección trata sobre las bases teóricas sobre las que se fundamenta el RUP. De entre ellas cabe destacar que se trata de un proceso iterativo e incremental. El desarrollo de los proyectos se hace en iteraciones, cada una de ellas conteniendo trabajo en varios flujos de trabajo. A su vez, las iteraciones se organizan en cuatro fases.

Después, tratamos sobre las fases y lo que se hace en cada una de ellas, así como de los productos que se deben obtener al finalizarlas.

Finalmente se desarrollan los flujos de trabajo, sus objetivos y, nuevamente, los productos que deben obtenerse de ellos.

**Bases teóricas**

El Proceso Unificado de Rational es un proceso de ingeniería del software, su propósito es asegurar la producción de software de alta calidad que se ajuste a las necesidades de sus usuarios finales con unos costos y calendario predecibles. En definitiva la RUP es una metodología de desarrollo de software que intenta integrar todos los aspectos a tener en cuenta durante todo el ciclo de vida del software, con el objetivo de hacer abarcables tanto pequeños como grandes proyectos software. Además Rational proporciona herramientas para todos los pasos del desarrollo así como documentación en línea para sus clientes.

Las características principales de RUP son:

* Guiado/Manejado por casos de uso: La razón de ser de un sistema software es servir a usuarios ya sean humanos u otros sistemas; un caso de uso es una facilidad que el software debe proveer a sus usuarios. Los casos de uso reemplazan la antigua especificación funcional tradicional y constituyen la guía fundamental establecida para las actividades a realizar durante todo el proceso de desarrollo incluyendo el diseño, la implementación y las pruebas del sistema.
* Centrado en arquitectura: La arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por plataformas software, sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales. Es como una radiografía del sistema que estamos desarrollando, lo suficientemente completa como para que todos los implicados en el desarrollo tengan una idea clara de qué es lo que están construyendo, pero lo suficientemente simple como para que si quitamos algo una parte importante del sistema quede sin especificar. Se representa mediante varias vistas que se centran en aspectos concretos del sistema, abstrayéndose de lo demás.
* Iterativo e Incremental: Para hacer más manejable un proyecto se recomienda dividirlo en ciclos. Para cada ciclo se establecen fases de referencia, cada una de las cuales debe ser considerada como un miniproyecto cuyo núcleo fundamental está constituido por una o más iteraciones de las actividades principales básicas de cualquier proceso de desarrollo. En concreto RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en los distintas actividades.

En la Figura 1 tenemos un ejemplo de la distribución del trabajo.

Además de estas características principales cabe destacar las siguientes:

* Desarrollo basado en componentes: La creación de sistemas intensivos en software requiere dividir el sistema en componentes con interfaces bien definidas, que posteriormente serán ensamblados para generar el sistema. Esta característica en un proceso de desarrollo permite que el sistema se vaya creando a medida que se obtienen o que se desarrollan y maduran sus componentes.
* Utilización de un único lenguaje de modelado: UML es adoptado como único lenguaje de modelado para el desarrollo de todos los modelos.
* Proceso Integrado: Se establece una estructura que abarque los ciclos, fases, flujos de trabajo, mitigación de riesgos, control de calidad, gestión del proyecto y control de configuración; el proceso unificado establece una estructura que integra todas estas facetas. Además esta estructura cubre a los vendedores y desarrolladores de herramientas para soportar la automatización del proceso, soportar flujos individuales de trabajo, para construir los diferentes modelos e integrar el trabajo a través del ciclo de vida y a través de todos los modelos. La estructura estática del proceso unificado se define en base a cuatro elementos, que son: los roles(antes workers), que responde a la pregunta ¿quién?, las actividades (activities), que responden a la pregunta ¿cómo?, los productos (artifacts), que responden a la pregunta ¿qué?, y los flujos de trabajo (workflows), que responden a la pregunta ¿cuándo?. La definición de estos términos es:
* Roles: Un rol define el comportamiento y responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. Una persona puede desempeñar diversos roles, así como un mismo rol puede ser representado por varias personas.
* Actividades: Una actividad de un trabajador en concreto es una unidad de trabajo que una persona que desempeñe ese rol puede ser solicitado a que realice. Las actividades tienen un objetivo concreto, normalmente expresado en terminos de crear o actualizar algún producto.
* Productos: Un producto o artefacto es un trozo de información que es producido, modificado o usado por un proceso. Los productos son los resultados tangibles del proyecto, las cosas que va creando y usando hasta obtener el producto final.
* Flujos de trabajo: La mera enumeración de roles, actividades y artefactos no define un proceso, necesitamos definir la secuencia de actividades realizadas por los diferentes roles, así como la relación entre los mismos, que nos producen unos resultados observables. Las distintas iteraciones a realizar consistirá en la ejecución de estos flujos de trabajo con una mayor o menor intensidad dependiendo de la fase e iteración en la que nos encontremos.

3. Las fases

Como ya se ha visto en el apartado anterior, la RUP se divide en cuatro fases.

En la tabla 1 encontramos un resumen de los principales productos de RUP y en que momento deben iniciarse y terminarse. Para estos productos y otros existen plantillas pregeneradas en [7]. La tabla resumen de [6] también proporciona una buena visión de conjunto de lo que hay que hacer en RUP. Por último en la página 41 de [1] está la figura 3-3 que muestra las relaciones de los productos de la tabla 1.

3.1 Inicio

Antes de iniciar un proyecto es conveniente plantearse algunas cuestiones: ¿Cuál es el objetivo? ¿Es factible? ¿Lo construimos o lo compramos? ¿Cuánto va a costar? La fase de inicio trata de responder a estas preguntas y a otras más. Sin embargo no pretendemos una estimación precisa o la captura de todos los requisitos. Más bien se trata de explorar el problema lo justo para decidir si vamos a continuar o a dejarlo. Generalmente no debe durar mucho más de una semana. Los objetivos son:

* Establecer el ámbito del proyecto y sus límites.
* Encontrar los casos de uso críticos del sistema, los escenarios básicos que definen la funcionalidad.
* Mostrar al menos una arquitectura candidata para los escenarios principales.
* Estimar el coste en recursos y tiempo de todo el proyecto.
* Estimar los riesgos, las fuentes de incertidumbre.

Los productos de la fase de inicio deben ser:

* Visión del negocio: Describe los objetivos y restricciones a alto nivel.
* Modelo de casos de uso.

Especificación adicional: requisitos no funcionales.

* Glosario: Terminología clave del dominio.
* Lista de riesgos y planes de contingencia.
* El caso de negocio (business case). Para más detalles ver el flujo de modelado del negocio.
* Prototipos exploratorios para probar conceptos o la arquitectura candidata.
* Plan de iteración para la primera iteración de la fase de elaboración.
* Plan de fases.

No todos los productos son obligatorios, ni deben completarse al 100%, hay que tener en cuenta el objetivo de la fase de inicio. Los síntomas de no haber entendido la fase de inicio:

* Dura más de unas pocas semanas.
* Se intentan definir todos los requisitos.
* Se espera que las estimaciones o los planes sean muy precisos.
* Definir la arquitectura completamente, en lugar de refinarla en la fase de elaboración.
* No se definen el caso de negocio o la visión.
* Los nombres de la mayoría de los casos de uso o actores no se han definido.
* Todos los casos de uso se escriben con detalle.

Al terminar la fase de inicio se deben comprobar los criterios de evaluación para continuar:

* Todos los interesados en el proyecto coinciden en la definición del ámbito del sistema y las estimaciones de agenda.
* Entendimiento los requisitos, evidenciado por la fidelidad de los casos de uso principales.
* Las estimaciones de tiempo, coste y riesgo son creíbles.
* Comprensión total de cualquier prototipo de la arquitectura desarrollado.
* Los gastos hasta el momento se asemejan a los planeados.

Si el proyecto no pasa estos criterios hay que plantearse abandonarlo o repensarlo profundamente.

**3.2 Elaboración**

El propósito de la fase de elaboración es analizar el dominio del problema, establecer los cimientos de la arquitectura, desarrollar el plan del proyecto y eliminar los mayores riesgos. Cuando termina esta fase se llega al punto de no retorno del proyecto: a partir de ese momento pasamos de las relativamente ligeras y de poco riesgo dos primeras fases, a afrontar la fase de construcción, costosa y arriesgada. Es por esto que la fase de elaboración es de gran importancia. En esta fase se construye un prototipo de la arquitectura, que debe evolucionar en iteraciones sucesivas hasta convertirse en el sistema final. Este prototipo debe contener los casos de uso críticos identificados en la fase de inicio. También debe demostrarse que se han evitado los riesgos más graves, bien con este prototipo, bien con otros de usar y tirar.

Los objetivos de esta fase son:

* Definir, validar y cimentar la arquitectura.
* Completar la visión.
* Crear un plan fiable para la fase de construcción. Este plan puede evolucionar en sucesivas iteraciones. Debe incluir los costes si procede.
* Demostrar que la arquitectura propuesta soportará la visión con un coste razonable y en un tiempo razonable.

Al terminar deben obtenerse los siguientes productos:

* Un modelo de casos de uso completa al menos hasta el 80%: todos los casos y actores identificados, la mayoría de los casos desarrollados.
* Requisitos adicionales.
* Descripción de la arquitectura software.
* Un prototipo ejecutable de la arquitectura.
* Lista de riesgos y caso de negocio revisados.
* Plan de desarrollo para el proyecto.
* Un caso de desarrollo actualizado que especifica el proceso a seguir.
* Posiblemente un manual de usuario preliminar.

La forma de aproximarse a esta fase debe ser tratar de abarcar todo el proyecto con la profundidad mínima. Sólo se profundiza en los puntos críticos de la arquitectura o riesgos importantes. En la fase de elaboración se actualizan todas los productos de la fase de inicio el glosario, el caso de negocio, el ROI (Return Of Invest), etcétera.

Los criterios de evaluación de esta fase son los siguientes:

* La visión del producto es estable.
* La arquitectura es estable.
* Se ha demostrado mediante la ejecución del prototipo que los principales elementos de riesgo han sido abordados y resueltos.
* El plan para la fase de construcción es detallado y preciso. Las estimaciones son creíbles.
* Todos los interesados coinciden en que la visión actual será alcanzada si se siguen los planes actuales en el contexto de la arquitectura actual.
* Los gastos hasta ahora son aceptables, comparados con los previstos. Si no se superan los criterios de evaluación quizá sea necesario abandonar el proyecto o replanteárselo considerablemente.

3.3 Construcción

La finalidad principal de esta fase es alcanzar la capacidad operacional del producto de forma incremental a través de las sucesivas iteraciones. Durante esta fase todas los componentes, características y requisitos, que no lo hayan sido hecho hasta ahora, han de ser implementados, integrados y testeados, obteniéndose una versión del producto que se pueda poner en manos de los usuarios (una versión beta). El énfasis en esta fase se pone controlar las operaciones realizadas, administrando los recursos eficientemente, de tal forma que se optimicen los costes, los calendarios y la calidad.

Los objetivos concretos son:

* Minimizar los costes de desarrollo mediante la optimización de recursos y evitando el tener que rehacer un trabajo o incluso desecharlo.
* Conseguir una calidad adecuada tan rápido como sea práctico.
* Conseguir versiones funcionales (alfa, beta, y otras versiones de prueba) tan rápido como sea práctico.

Los productos de la fase de construcción son:

* Modelos Completos (Casos de Uso, Análisis, Diseño, Despliegue e Implementación)
* Arquitectura íntegra (mantenida y mínimamente actualizada)
* Riesgos Presentados Mitigados
* Plan del Proyecto para la fase de Transición
* Manual Inicial de Usuario (con suficiente detalle)
* Prototipo Operacional – beta
* Caso del Negocio Actualizado

3.4 Transición

La finalidad de la fase de transición es poner el producto en manos de los usuarios finales, para lo que típicamente se requerirá desarrollar nuevas versiones actualizadas del producto, completar la documentación, entrenar al usuario en el manejo del producto, y en general tareas relacionadas con el ajuste, configuración, instalación y usabilidad del producto.

En concreto algunas de las cosas que puede incluir esta fase:

* Testeo de la versión Beta para validar el nuevo sistema frente a las expectativas de los usuarios.
* Funcionamiento paralelo con los sistemas legados que están siendo sustituidos por nuestro proyecto.
* Conversión de las bases de datos operacionales.
* Entrenamiento de los usuarios y técnicos de mantenimiento.
* Traspaso del producto a los equipos de marketing, distribución y venta.

Los principales objetivos de esta fase son:

* Conseguir que el usuario se valga por sí mismo.
* Un producto final que cumpla los requisitos esperados, que funcione y satisfaga
* suficientemente al usuario.

Los productos de la fase de transición según son:

* Prototipo Operacional
* Documentos Legales
* Caso del Negocio Completo
* Línea de Base del Producto completa y corregida que incluye todos los modelos del sistema
* Descripción de la Arquitectura completa y corregida

Las iteraciones de esta fase irán dirigidas normalmente a conseguir una nueva versión.

Las actividades a realizar durante las iteraciones dependerán de su finalidad, si es corregir algún error detectado, normalmente será suficiente con llevar a cabo los flujos de trabajo de implementación y test, sin embargo, si se deben añadir nuevas características, la iteración será similar a la de una iteración de la fase de construcción. La complejidad de esta fase depende totalmente de la naturaleza del proyecto, de su alcance y de la organización en la que deba implantarse.

4. Los flujos de trabajo

En RUP se definen nueve flujos de trabajo distintos, separados en dos grupos. Los flujos de trabajo de ‘ingeniería’ son los siguientes:

* Modelado del negocio.
* Requisitos.
* Análisis y diseño.
* Implementación.
* Test.
* Despliegue.

Los flujos de trabajo de apoyo son:

* Administración del proyecto.
* Configuración y control de cambios.
* Entorno.

Aunque los nombres de los flujos de trabajo de ingeniería recuerden a las etapas de una metodología en cascada, en RUP como ya hemos visto las fases son distintas, y estos flujos de trabajo serán visitados una y otra vez a lo largo de todo el proceso.

4.1 Administración del Proyecto

El objetivo de la administración de un proyecto es conseguir equilibrar el completar los objetivos, administrar el riesgo y superar las restricciones para desarrollar un producto que sea acorde a los requisitos de los usuarios.

Para conseguir esto el flujo de trabajo se centra en tres aspectos:

* Planificar un proyecto iterativo y cada iteración particular.
* Administrar el riesgo.
* Monitorizar el progreso del proyecto a través de métricas.

La planificación de un proyecto debe acometerse en dos niveles de abstracción: un plan de “grano grueso” para las fases y un plan de “grano fino” para cada iteración. El plan de desarrollo (o plan de fases) debe contener las fechas esperadas para los hitos principales, cuándo se tendrá la arquitectura, cuándo estará la primera versión beta, etcétera. Estas fechas coincidirán, generalmente, con el final de las fases. También debería tener una previsión de las necesidades de personal y medios, así como fechas de hitos menores, sólo si se conocen. Este plan debe obtenerse temprano en la fase de inicio y no debe ir más allá de una o dos páginas. Debe actualizarse siempre que sea necesario. Debe realizarse un plan de iteración por cada iteración, como cabría suponer. Este plan se elabora hacia la segunda mitad de la iteración, lo que significa que en un momento dado habrá dos planes activos: el de la iteración en curso y el de la próxima, que es construido en ésta. En este plan se detallarán fechas importantes para la iteración: compilaciones importantes, revisiones o llegada de componentes.

La administración del riesgo consiste en ocuparse de las incógnitas de un proyecto, las cuestiones que pueden llevarlo a pique. En concreto hay que identificar los riesgos, típicamente en la fase de inicio, y hacerles frente. Para ello trataremos de evitarlos, transferirlos o asumirlos. En este último caso habrá que tratar de mitigar el riesgo y definir un plan de contingencia por si el riesgo se convierte en un problema real. En definitiva la administración del riesgo consistirá en gestionar una lista de riesgos. Monitorizar un proyecto es importante para mantenerlo bajo control. Tenemos que “medir” para ver como de bien nos ajustamos a los planes, la calidad requerida y los requisitos. También es necesario para planificar de forma precisa y ver cuál es el comportamiento del proyecto frente a cambios. Tomar métricas no es gratis, así que hay que justificar por qué se mide. En este flujo de trabajo también se obtiene el caso de negocio (business case). Consiste en el contexto del negocio, criterios de éxito del proyecto (como por ejemplo, ser pagados) y una previsión de financiera (gastos, salarios, etcétera). Si esperamos vender el sistema, también tendrá que haber una aproximación a los beneficios que obtendremos: el ROI (Return Of Investment).

**Producto: Plan de desarrollo.**

Ya hemos visto cuales son los objetivos de este producto. Ahora veremos con detalle uno, así como las partes de que se compone la plantilla. La sección 1 se comenta en el anexo 1. La sección 2 pretende proporcionar una breve visión de conjunto del proyecto, sus objetivos, restricciones, los entregables que se van a producir (los productos) y que evolución se espera. La sección 3 trata de la organización de la gente que desarrollará el proyecto y sus interacciones con el exterior. En el ejemplo se consideran cuatro roles: gestor del proyecto, analistas, desarrolladores y testeadores.

La sección 4 es la más importante, trata de la gestión del proyecto propiamente. En el ejemplo vemos como se ha obtenido un plan de fases en el que se indica el número de iteraciones previstas. Después se detalla que se va a hacer en cada fase y los hitos que se espera obtener. Como se puede ver el ejemplo carece de las secciones 5, 6 y 7. Esto se debe a que contendrían planes para proyectos de gran envergadura, como aseguramiento de la calidad, infraestructuras, etcétera.

**Producto: Plan de iteración.**

Este producto trata la planificación de grano fino. La sección 2 contiene el plan. En el ejemplo 1 se refieren a unos diagramas de Gantt creados con Microsoft Project. Estos diagramas dan una previsión detallada del tiempo asignado a cada tarea. Una forma menos formal de planificar sería poner sólo la fecha de fin, como en el ejemplo 2. En la sección 3 se reseñan todos los recursos humanos, financieros o de otra índole necesarios para completar la iteración. La sección 4 es una lista de los casos de uso implicados en esta iteración. Si se desea puede justificarse su elección.

Por último está la sección 5 con los criterios de evaluación de la iteración. En los ejemplos no está la sección, mal hecho por su parte.

**Producto: Lista de Riesgos.**

Este producto sólo tiene una sección además de la primera: la lista de riesgos propiamente dicha. Para cada uno hay que indicar: su magnitud, una descripción, su impacto (donde afecta), indicadores para monitorizarlo, una estrategia para mitigarlo y el plan de contingencia por si el riesgo se hace real.

En los ejemplos se ven dos formas expresar cada riesgo: en forma de tablas o con texto puro. Que cada cual elija la que la más le guste, pero si se va a hacer algo raro, conviene explicarlo antes, como en el ejemplo 1.

4.2 Modelado del negocio

Con este flujo de trabajo pretendemos llegar a un mejor entendimiento de la organización donde vamos a implantar nuestro producto. Los principales motivos para esto son los siguientes: asegurarnos de que el producto será algo útil, no un obstáculo; conseguir que encaje de la mejor forma posible en la organización; y tener un marco común para los desarrolladores, los clientes y los usuarios finales. Este flujo de trabajo no será siempre necesario. Si sólo añadimos funcionalidad que no verán los usuarios directamente, no hará falta. Para modelar el negocio usaremos las mismas técnicas que para modelar software, facilitando que ambas partes entiendan los modelos. En concreto tendremos casos de uso de negocio, actores de negocio, etcétera. Los diagramas también tendrán su equivalente de negocio. Dependiendo del tipo de software que estemos construyendo, el modelado del negocio cambiará. No se trata de modelar la organización de arriba abajo, sólo la parte que nos toca. En concreto en [1] se pueden encontrar seis escenarios para escoger el más apropiado o hacernos una idea de qué hay que modelar. En la fase de inicio habrá que hacer una valoración del negocio. Tras hacerla decidiremos cómo vamos ha hacer el modelado del negocio: por ejemplo, ver en que escenario de los seis estamos.

Como ya dije usaremos una extensión de UML para modelar el negocio. En concreto habrá modelos de casos de uso de negocio y modelos de objetos de negocio. Con los primeros capturaremos QUÉ se hace y QUIÉN lo hace. Con los segundos veremos CÓMO se hace. Una gran ventaja de ésta aproximación al modelado del negocio es que es un forma clara y concisa de mostrar las dependencias entre el negocio y el sistema que estamos construyendo.

4.3 Requisitos

Este es uno de los flujos de trabajo más importantes, porque en él se establece QUÉ es lo que tiene que hacer exactamente el sistema que construyamos. En esta línea los requisitos son el contrato que debemos cumplir, de modo que los usuarios finales tienen que comprender y aceptar los requisitos que especifiquemos. Tal como indica [2], los requisitos se dividen en dos grupos. Los requisitos funcionales son las cosas que el sistema puede hacer, su funcionalidad. Se modelan mediante diagramas de casos de uso. Los requisitos no funcionales representan aquellos atributos que debe exhibir el sistema, pero que no son una funcionalidad específica. Por ejemplo requisitos de usabilidad, fiabilidad, eficiencia, portabilidad, etcétera. Para capturar los requisitos es preciso entrevistar a todos los interesados en el proyecto, no sólo a los usuarios finales, y anotar todas sus peticiones. A partir de ellas hay que descubrir lo que necesitan y expresarlo en forma de requisitos. En este flujo de trabajo, y como parte de los requisitos de usabilidad, se diseña la interfaz gráfica de usuario. Para ello habitualmente se construyen prototipos de la GUI que se contrastan con el usuario final. Además ésta es una buena forma de explicitar requisitos ( Usuario: “¿Dónde está el botón de calcular la sobrecarga de TLA?” Analista: “¿La qué?”). En definitiva, en este flujo de trabajo hay que analizar el problema, comprender las necesidades de los interesados y expresarlas en forma de requisitos; construir diagramas de casos de uso para los requisitos funcionales, los no funcionales describirlos textualmente en especificaciones suplementarias. Además hay que gestionar los cambios en los requisitos a lo largo de todo el proceso. Dentro de éste flujo, en la fase de inicio hay que crear la Visión. Se trata de un documento que de una vista general del núcleo de los requisitos del proyecto, características clave y restricciones principales. Algunos dominios de negocio pueden ser algo enrevesados al principio, por ejemplo si hay que crear una aplicación para una base aérea. Por este motivo puede ser de gran ayuda construir un Glosario que recoja la terminología usada a lo largo del proyecto o la organización.

**Producto: Modelo de Casos de Uso.**

Este producto se obtiene con la plantilla de Especificación de Casos de Uso. En teoría habría que hacer un documento por caso de uso, y así puede hacerse. Sin embargo en los ejemplos se optó por fusionar todos los casos de uso en un documento. Por cada caso de uso hay que dar una pequeña descripción. A continuación hay que describir el flujo de eventos del caso. Primero se destaca el flujo principal y después vienen los alternativos. Si una alternativa es simple, puede ponerse con el flujo principal. Si un caso es complejo, puede ponerse figuras explicativas, diagramas UML o lo que se necesite. Lo siguiente es poner que requisitos especiales hay, si los hay. Luego vienen las precondiciones y las postcondiciones. Finalmente los puntos de extensión. Si se opta por juntar todas las especificaciones de casos de uso convendrá hacer una primera sección como en el ejemplo 1, describiendo los actores, las relaciones de los casos de uso y mostrando los diagramas.

**Producto: Especificación Adicional.**

En este producto se especifican todos los requisitos no funcionales de nuestro sistema. Como ya se dijo, se trata de atributos que no dan funcionalidad, por ejemplo lo fácil que es de usar o si cumple con una normativa concreta.

La plantilla para este producto está dividida en secciones que indican distintos tipos de requisitos no funcionales. Nosotros tendremos que ir simplemente colocando nuestros requisitos en la sección adecuada y borrar las que no vayamos a usar. Del mismo modo podemos crear secciones nuevas si las necesitamos.

En el ejemplo 2 se optó por obviar esta clasificación y poner todos los requisitos juntos. En mi opinión las secciones añaden claridad y sirven de recordatorio para no dejarnos nada, pero si hay pocos requisitos, pueden ser un estorbo.

**Producto: Visión.**

El propósito de la visión es reunir, analizar y definir necesidades y características del sistema a alto nivel.

La sección 2 pretende posicionar el problema que da lugar al sistema. Para ello se describe el problema, que oportunidad para hacer negocio hay y que lugar en el mercado ocupará el sistema como producto. Si no pretendemos vender nada, bastará con describir el problema, como en el ejemplo 2. La sección 3 pretende dar a conocer todos los interesados en el sistema y los usuarios finales.

Hay que indicar los problemas que cada uno ve que deben ser resueltos. No hay que poner las peticiones concretas sino el trasfondo de cada interesado. Si el producto se va a vender, hay que poner un estudio de la población objetivo. El ejemplo puede aclarar bastante este punto. Las secciones 4 y 5 dan una visión de conjunto del producto y sus capacidades. Algunos puntos de la plantilla sólo son apropiados si se pretende vender el producto. Las secciones restantes dan un mayor detalle sobre el producto. En principio, para un proyecto pequeño, podrían no ser necesarios tantos apartados si se puede escribir un texto breve y claro que describa lo mismo. Sin embargo es buena idea mirarse todas las secciones para no olvidar nada.

**Producto: Glosario.**

En el glosario se recoge el vocabulario propio del dominio del sistema, y que dependiendo del proyecto pueden ser términos muy especializados. Además puede usarse para definir un diccionario informal de tipos de datos. El núcleo de este producto es la sección 2 donde se va introduciendo a modo de diccionario, y normalmente por orden alfabético, la definición de los diferentes conceptos. En las sección 3 se definen aquellos estereotipos(especificaciones de UML) que no son los predefinidos por RUP o UML y que pueden o deben ser usados en el proyecto. Ni el ejemplo 1 ni el 2 han hecho uso de este último apartado.

4.4 Análisis y diseño

El objetivo de este flujo de trabajo es traducir los requisitos a una especificación que describe cómo implementar el sistema. El análisis consiste en obtener una visión del sistema que se preocupa de ver QUÉ hace, de modo que sólo se interesa por los requisitos funcionales. Por otro lado el diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, en definitiva CÓMO cumple el sistema sus objetivos. Como se dice en [1] el diseño debe ser suficiente para que el sistema pueda ser implementado sin ambigüedades. De hecho, cuando la precisión del diseño es muy grande, la implementación puede ser hecha por un generador automático de código. Al principio de la fase de elaboración hay que definir una arquitectura candidata: crear un esquema inicial de la arquitectura del sistema, identificar clases de análisis y actualizar las realizaciones de los casos de uso con las interacciones de las clases de análisis. Durante la fase de elaboración se va refinando esta arquitectura hasta llegar a su forma definitiva. En cada iteración hay que analizar el comportamiento para diseñar componentes. Además si el sistema usará una base de datos, habrá que diseñarla también, obteniendo un modelo de datos. El resultado final más importante de este flujo de trabajo será el modelo de diseño. Consiste en colaboraciones de clases, que pueden ser agregadas en paquetes y subsistemas. Otro producto importante de este flujo es la documentación de la arquitectura software, que captura varias visiones arquitectónicas del sistema.

**Producto: Modelo de Diseño**

No se dispone de una plantilla para este producto. Es sencillamente la estructuración de los distintos diagramas y modelos que se tengan referentes a la parte de diseño del sistema. En el ejemplo 1 la información se ha estructurado en cuatro apartados, en el primero se muestran los paquetes que forman el sistema y sus relaciones, en el segundo se muestra lo que contiene cada paquete, y en el tercero y cuarto se muestra una vista lógica del sistema, general en el tercer apartado y detallada en el cuarto, mediante diagramas de clases de diseño.

**Producto: Documentación de la Arquitectura Software**

En este documento se da una descripción de la arquitectura del sistema, véase el apartado de características principales del RUP en el apartado de Bases teóricas de este trabajo. En el apartado 2 del documento se anticipan las vistas que van a ser necesarias para la descripción, así como la representación escogida para las mismas. En la sección 3 se describen los requerimientos y objetivos del sistema que sean de influyan en la arquitectura del mismo. A partir de aquí viene un apartado por cada vista (Casos de uso, lógica, proceso, despliegue, e implementación) que se incluya en el documento. Así como una vista optativa adicional del almacenamiento de los datos persistentes. Se termina con una sección para describir los requerimientos en tiempo y espacio, y otra con una explicación de cómo contribuye la arquitectura a garantizar la calidad del software.

**4.5 Implementación**

En este flujo de trabajo se implementan las clases y objetos en ficheros fuente, binarios, ejecutables y demás. Además se deben hacer los tests de unidad: cada implementador es responsable de testear las unidades que produzca. El resultado final de este flujo de trabajo es un sistema ejecutable. En cada iteración habrá que hacer lo siguiente:

* Planear que subsistemas deben ser implementados y en que orden deben ser integrados, formando el Plan de Integración.
* Cada implementador decide en que orden implementa los elementos del subsistema. Si encuentra errores de diseño, los notifica.
* Se testean los subsitemas individualmente.
* Se integra el sistema siguiendo el plan.

La estructura de todos los elementos implementados forma el modelo de implementación. La integración debe ser incremental, es decir, en cada momento sólo se añade un elemento. De este modo es más fácil localizar fallos y los componentes se prueban más a fondo. En fases tempranas del proceso se pueden implementar prototipos para reducir el riesgo. Su utilidad puede ir desde ver si el sistema es viable desde el principio, probar tecnologías o diseñar la interfaz de usuario. Los prototipos pueden ser exploratorios (desechables) o evolucionarios. Estos últimos llegan a transformarse en el sistema final.

**Producto: Modelo de implementación**

La plantilla para este producto no la proporciona UPEDU (ver anexo I), que consiste en una visión general lo que tiene que ser implementado, y un apartado para cada iteración (que coincide con la plantilla de RUP del plan de integración de constructos) con los componentes y subsistemas a implementar durante esa iteración, así como de los resultado software (builds) que se han de obtener y el testeo que se ha de realizar sobre ellos (para lo que se puede hacer referencia al plan de test).

**4.6 Test**

Este flujo de trabajo es el encargado de evaluar la calidad del producto que estamos desarrollando, pero no para aceptar o rechazar el producto al final del proceso de desarrollo, sino que debe ir integrado en todo el ciclo de vida. Como se dice en [1] : “El papel del testeo no es asegurar la calidad, pero sí evaluarla, y proporcionar una realimentación a tiempo, de forma que las cuestiones de calidad puedan resolverse de manera efectiva en tiempo y coste. ”

Los principales aspectos a ser evaluados en un producto software son la Fiabilidad (resistente a fallos), la Funcionalidad (hace lo que debe) y el Rendimiento (lleva a cabo su trabajo de manera efectiva). Los tests pueden hacerse a diferentes niveles dependiendo del objetivo de los mismos, a saber: Test de unidad (se testean las unidades mínimas por separado, y normalmente se hace durante la implementación misma), de integración (varias unidades juntas), de sistema (sobre la aplicación o sistema completo) y de aceptación (realizado sobre el sistema global por los usuarios o terceros). Dentro de cada uno de estos niveles podemos distinguir distintos tipos de test. A la representación de lo que será testeado y cómo debe de hacerse es a lo que se le llama el modelo de test. Incluye la colección de casos de test, procedimientos de test, scripts, resultados esperados...

Las actividades de este flujo comienzan pronto en el proyecto con el plan de test (el cual contiene información sobre los objetivos generales y específicos del testeo en el proyecto, así como las estrategias y recursos con que se dotará a esta tarea), o incluso antes con alguna evaluación durante la fase de inicio, y continuará durante todo el proyecto. El desarrollo del flujo de trabajo consistirá en planificar qué es lo que hay que testear, diseñar cómo se va a hacer, implementar lo necesario para llevarlos a cabo, ejecutarlos en los niveles necesarios y obtener los resultados, de forma que la información obtenida nos sirva para ir refinando el producto a desarrollar.

**Producto: Plan de Test**

El RUP diferencia entre un Plan de Test global, donde se describen los objetivos y mecanismos que se van a utilizar para el proyecto en general, así como un plan de test específico para cada iteración donde se especifica qué elementos se deben testear, cuales son los objetivos que se persiguen con esas pruebas, y la aproximación a utilizar para conseguir esos objetivos. Incluyendo también una estimación de los recursos necesarios para llevarlos a cabo.

**4.7 Configuración y gestión de cambios**

La finalidad de este flujo de trabajo es mantener la integridad de todos los artefactos que se crean en el proceso, así como de mantener información del proceso evolutivo que han seguido. Las causas por las que la evolución de los artefactos puede causar problemas según [8] son:

* Actualización simultánea: Se da cuando dos personas trabajan por separado sobre el mismo artefacto a la vez, el último en hacer las modificaciones sobrescribe lo hecho por el primero.
* Notificación limitada: Cuando un problema ha sido resuelto en un artefacto compartido por varios roles y algunos de ellos no son notificados del cambio.
* Multiples versiones: Cuando se trabaja con diferentes versiones del producto al mismo tiempo en diferentes flujos de trabajo, pueden surgir problemas si los cambios no son convenientemente monitorizados y propagados. La configuración y gestión de cambios cubre tres funciones interdependientes:
* La gestión de la configuración, que maneja la estructura del producto, la identificación de los elementos, configuraciones válidas de los mismos versiones, versiones y espacios de trabajo.
* Gestión de las peticiones de cambio, que coordina el proceso de modificar artefactos de una manera consistente.
* Métricas y status, que se encarga de extraer información para la correcta administración del proyecto de las herramientas que soportan las dos funciones anteriores.

**4.8 Entorno**

La finalidad de este workflow es dar soporte al proyecto con las adecuadas herramientas, procesos y métodos. Es decir tener a punto las herramientas que se vayan a necesitar en cada momento, así como definir la instancia concreta de proceso unificado que se va a seguir. En concreto las responsabilidades de este flujo de trabajo incluyen:

* Selección y adquisición de herramientas.
* Establecer y configurar las herramientas para que se ajusten a la organización.
* Configuración del proceso.
* Mejora del proceso.
* Servicios técnicos.

El principal artefacto que se usa en este flujo de trabajo es el caso de desarrollo que especifica para el proyecto actual en concreto, cómo se aplicará el proceso unificado, que productos se van a utilizar y cómo van a ser utiliados. Además se tendrán que definir las líneas guía (los pasos concretos y políticas a seguir) para los distintos aspectos del proceso, como pueden ser el modelado del negocio y los casos de uso, para la interfaz de usuario, el diseño, la programación, el manual de usuario, ...

Las actividades que se deben llevar a cabo durante este flujo de trabajo según [6] son:

* Preparar el entorno para el trabajo.
* Preparar el entorno para una iteración.
* Preparar las líneas de guía para una iteración.
* Dar soporte al entorno durante la iteración.

**4.9 Despliegue**

El objetivo de este flujo de trabajo es producir con éxito distribuciones del producto y distribuirlo a los usuarios. Las actividades implicadas incluyen:

* Testear el producto en su entorno de ejecución final.
* Empaquetar el software para su distribución.
* Distribuir el software.
* Instalar el software.
* Proveer asistencia y ayuda a los usuarios.
* Formar a los usuarios y al cuerpo de ventas.
* Migrar el software existente o convertir bases de datos.

Este flujo de trabajo se desarrolla con mayor intensidad en la fase de transición, ya que el propósito tanto del flujo como de la fase es asegurar una aceptación y adaptación sin complicaciones del software por parte de los usuarios. Aunque la ejecución de este flujo de trabajo debe empezar en fases anteriores, para preparar el camino, sobre todo con actividades de planificación, pero también con la elaboración del manual de usuario, tutoriales, ... Dado el amplio rango de aplicaciones que se pueden dar y sus diversas características los productos necesitados por este flujo de trabajo puede variar en gran medida. Aunque el artefacto clave es una distribución (release) del producto, que en general puede consistir de:

* Software ejecutable (en todos los casos).
* Productos de instalación: scripts, herramientas, archivos, guías, información sobre licencia, ...
* Notas de la distribución, describiendola al usuario final.
* Material de apoyo, como pueden ser los manuales de usuario, de operaciones y mantenimiento.
* Materiales formativos.

A diferencia de otros flujos de trabajo RUP da un detalle menor al despliegue, debido a la ya citada diversidad y especificidad de cada proyecto.

**5. Conclusiones**

En este documento se ha dado una visión general de lo que es el RUP, así como de la estructura bidimensional que sigue, dividiendo el proceso en fases, y estas en flujos de trabajo. Se han dado apuntes de lo que se espera de cada fase así como la forma de manejar los flujos de trabajo. Para aplicar el RUP en pequeños equipos y proyectos se deberá mapear los diferentes roles (a los que no hemos dado especial relevancia en este documento) entre los distintos miembros del equipo, pero la diferencia clave con un proyecto de mayor envergadura, según [9], es el grado de formalidad a la hora de usar los distintos artefactos, planes del proyecto, requisitos, clases, ... En [10] se pueden encontrar consejos sobre qué artefactos utilizar y cómo hacerlo.

El RUP es una metodología completa y extensa que intenta abarcar todo el mundo del desarrollo software, tanto para pequeños proyectos, como proyectos más ambiciosos de varios años de duración. Por lo que existe una gran cantidad de documentación sobre el mismo, tanto en libros como en la red, eso sí en inglés. Es sin embargo difícil empezar a aplicar esta metodología en una organización. Por eso esperamos que este documento sirva tanto para familiarizar con el Proceso Unificado a aquellos que no lo conocían, así como de servir de guía durante la ejecución del mismo.

[**http://anaylenlopez.files.wordpress.com/2011/03/trabajo-guia20rup.pdf**](http://anaylenlopez.files.wordpress.com/2011/03/trabajo-guia20rup.pdf)

**Lenguaje para el modelado: UML 2.0**

Para el análisis y el diseño optamos por las herramientas que nos ofrece el Lenguaje Unificado de Modelado (UML), este nos permite represar de manera más sencilla toda la documentación del sistema; definirlo, detallar sus artefactos, documentar y construir. En otras palabras, es el lenguaje en el que estará descrito el modelo.



Lenguaje Unificado de Modelado (LUM o **UML**, por

sus siglas en inglés, Unified Modeling Language)

es el lenguaje de modelado de sistemas de software

más conocido y utilizado en la actualidad;

está respaldado por el OMG

(Object Management Group).

Es un lenguaje gráfico para visualizar,

especificar, construir y documentar un sistema.

UML ofrece un estándar para describir un

"plano" o modelo del sistema, incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Es importante resaltar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa Lenguaje Unificado de Modelado, no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, sin embargo, la programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

**Diagramas UML**

El UML está compuesto por diversos elementos gráficos que se combinan para conforma diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos.

La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Recordemos que un modelo es una representación simplificada de la realidad; el modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema.

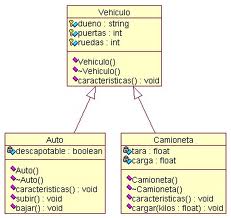
A continuación se describen los más importantes:

**Diagrama de clase**

Para modelar clases, incluidos sus atributos, operaciones, relaciones y asociaciones con otras clases, el UML proporciona un diagrama de clase, que aporta una visión estática o de estructura de un sistema, sin mostrar la naturaleza dinámica de las comunicaciones entre los objetos de las clases.

Los elementos principales de un diagrama de clases son cajas, que son los iconos utilizados para representar clases e interfaces. Cada caja se divide en partes horizontales para contener el nombre de la clase, sus atributos y operaciones. Un atributo es algo que un objeto de dicha clase conoce o puede proporcionar todo el tiempo. Las operaciones o comportamientos de la clase es lo que pueden hacer los objetos de la clase. Por lo general se implementa como un método de la clase.

El propósito de este diagrama es representar los objetos fundamentales del sistema, es decir los que percibe el usuario y con los que espera tratar de completar su tarea en vez objetos de sistema o de un modelo de programación.



**Diagrama de Objetos**

Los Diagramas de Objetos están vinculados con los Diagramas de Clases. Un objeto es una instancia de una clase, por lo que un diagrama de objetos puede ser visto como una instancia de un diagrama de clases. Los diagramas de objetos describen la estructura estática de un sistema en un momento particular y son usados para probar la precisión de los diagramas de clases.

Nombre de los objetos

Cada objeto es representado como un rectángulo, que contiene el nombre del objeto y su clase subrayadas y separadas por dos puntos.

Atributos

Como con las clases, los atributos se listan en un área inferior. Sin embargo , los atributos de los objetos deben tener un valor asignado.

**Diagrama de Caso de Uso**

Los casos de uso y el diagrama de caso de uso UML ayudan a determinar la funcionalidad y características del software desde la perspectiva del usuario.

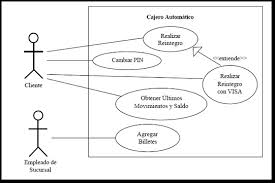
Un caso de uso describe la manera en la que un usuario interactua con el sistema, definiendo los pasos requeridos para lograr una meta específica. Las variaciones en la secuencia de pasos describen escenarios.

Un diagrama UML de caso de uso es un panorama de todos los casos de uso y sus relaciones. El mismo proporciona un gran cuadro de la funcionalidad del sistema.

Los diagramas de casos de uso son a menudo confundidos con los casos de uso. Mientras los dos conceptos están relacionados, los casos de uso son mucho más detallados que los diagramas de casos de uso.

La descripción escrita del comportamiento del sistema al afrontar una tarea de negocio o un requisito de negocio. Esta descripción se enfoca en el valor suministrado por el sistema a entidades externas tales como usuarios humanos u otros sistemas.

La posición o contexto del caso de uso entre otros casos de uso. Dado que es un mecanismo de organización, un conjunto de casos de uso coherentes y consistentes promueven una imagen fácil de comprender del comportamiento del sistema, un entendimiento común entre el cliente/propietario/usuario y el equipo de desarrollo.



**Diagrama de Actividad**

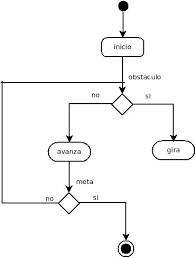
Un diagrama de actividad UML muestra el comportamiento dinámico de un sistema o de parte de un sistema a través del flujo de control entre acciones que realiza el sistema. Es similar a un diagrama de flujo, excepto porque el diagrama de actividad puede mostrar flujos concurrentes.

El componente principal de un diagrama de actividad es un nodo acción, representado mediante un rectángulo redondeado, que corresponde a una tarea realizada por el sistema de software. Las flechas desde un nodo acción hasta otro indican el flujo de control; es decir, una flecha entre dos nodos acción significa que, después de completar la primera acción, comienza la segunda acción. Un punto negro sólido forma el nodo inicial que indica el punto de inicio de la actividad. Un punto negro rodeado por un círculo negro es el nodo final que indica el fin de la actividad.

Un fork representa la separación de actividades en dos o mas actividades concurrentes. Se dibuja como una barra negra horizontal con una flecha apuntando hacia ella y dos o más flechas apuntando en sentido opuesto. Cada flecha continua representa un flujo de control que puede ejecutarse de manera concurrente con los flujos correspondientes a las otras flechas continuas. Dichas actividades concurrentes pueden realizarse en una computadora, usando diferentes hebras o incluso diferentes computadoras.

Un Join es una forma de sincronizar flujos de control concurrentes. Se representa mediante una barra negra horizontal con dos o más flechas entrantes y una flecha saliente. El flujo de control representado por la flecha saliente no puede comenzar la ejecución hasta que todos los flujos representados por las flechas entrantes se hayan completado.

El nodo de decisión corresponde a una rama en el flujo de control con base en una condición. Tal nodo se representa como un triángulo blanco con una flecha entrante y dos o más flechas salientes. Cada flecha saliente se etiqueta con una guardia (condición dentro de corchetes).



**Diagrama de Secuencias**

Los diagramas de clases y los de objetos representan información estática. No obstante, en un sistema funcional, los objetos interactúan entre sí, y tales interacciones suceden con el tiempo. Los diagramas de secuencia muestran la interacción dinámica entre los objetos enfocándose en el pasaje de mensajes. Muestran la interacción y comunicación entre objetos a medida que trascurre el tiempo de ejecución, con lo que podemos ordenar de manera temporal los eventos que ocurren en el sistema. Podemos especificar, por ejemplo, en un sistema web de foros, que los usuarios antes de podes escribir un mensaje deben registrarse en el sitio. Sus componentes principales son los objetos, los mensajes que intercambian y su orden cronológico.

Rol de la Clase

El rol de la clase describe la manera en que un objeto se va a comportar en el contexto. No se listan los atributos del objeto.

Activación

Los cuadros de activación representan el tiempo que un objeto necesita para completar una tarea.

Mensajes

Los mensajes son flechas que representan comunicaciones entre objetos. Las medias flechas representan mensajes asincrónicos.

Los mensajes asincrónicos son enviados desde un objeto que no va a esperar una respuesta del receptor para continuar con sus tareas.

Líneas de Vida

Las líneas de vida son verticales y en línea de puntos, ellas indican la presencia del objeto durante el tiempo.

|  |  |  |
| --- | --- | --- |
| Diagrama | Estructura | Propiedades |
| De clases | Estática | * Enfocados en las clases y su interacción |
| De objetos | Estática | * Instancias concretas de un diagrama de clases general |
| De casos de uso | Dinámica | Captura Requerimientos. Enfocados en la interacción de los usuarios con el sistema |
| De secuencia | Dinámica | Modela el intercambio de mensajes entre objetos y los ordena temporalmente |
| De colaboración | Dinámica | Modela cómo dos o más clases se organizan para cumplir con un determinado objetivo |
| De estado | Dinámica | Basado en estados y transiciones |
| De actividad | Dinámica | Modelan el flujo de control |
| De componentes | Estática | Interacciones a alto nivel. Modelan la estructura de paquetes |
| De producción | Estática | Detalles de configuración y arquitecturales |

***Tabla 1.*** *Resumen de las características de los nueve diagramas principales de UML*

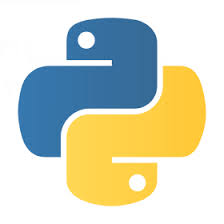
**Diagrama de componentes**

Los diagramas de componentes reflejan la estructura estática de nuestro sistema,pero desde un punto de vista físico. Generalmente, ya tenemos la codificación de nuestro sistema realizada y plasmamos las ubicaciones físicas dentro de los diagramas de componentes. Un componente puede representar tanto el código de una clase como de una instancia, una interfaz o, incluso, también puede interpretarse como paquetes. Sus principales usos son los siguientes:

* Modelar el código fuente.
* Modelar las distintas versiones de nuestro software final.
* Modelar base de datos físicas.

De los tres usos que mencionamos, sin dudas el primero es el mas popular de todos. Es importante que tengamos en cuenta que cuando hablamos de modelar código fuente, incluimos librerías, ejecutables, tablas, archivos y documentos, es decir, el objetiivo es modelar la visión global del código.

Los elementos que componen un diagrama de componentes son los componentes propiamente dichos sumados a la especificación de interfaces, as{i como también toda la variedad posible de relaciones entre todos los elementos. Los componentes, a su vez, en su estructura interna pueden incluir tanto paquetes como clases.



**Python**

**Introducción**

Es un lenguaje de programación interpretado o de script, fue creado por Gido Van Rossum a principios de los años 90. Este nombre proviene de un grupo humorístico llamado Monty Python del cual Guido era muy aficionado. Tiene una sintaxis limpia, favoreciendo código legible, evitando el punto y coma en cada sentencia y en vez de utilizar manejo de llaves hace uso de la indentación o del también denominado bloque de espacio.

Es un lenguaje de programación multiparadigma, haciendo uso de la programación orientada a objetos, imperativa y funcional.

Entre sus características principales podemos mencionar que Python es un lenguaje interpretado, hace uso de intérprete para la ejecución de cada instrucción de programa; tiene tipado dinámico, su tipo de dato se determinará en tiempo de ejecución según el tipo de valor al que se le asigne. Es fuertemente tipado, no se permite tratar a una variable como si fuera de un tipo distinto al que tiene. Es Multiplataforma, está disponible para Windows Mac y distribuciones Unix.

Es administrado por la [Python Software Foundation](http://es.wikipedia.org/wiki/Python_Software_Foundation). Posee una licencia de [código abierto](http://es.wikipedia.org/wiki/C%C3%B3digo_abierto), denominada [Python Software Foundation License](http://es.wikipedia.org/wiki/Python_Software_Foundation_License), que es compatible con la [Licencia pública general de GNU](http://es.wikipedia.org/wiki/GNU_General_Public_License) a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores. Es una apuesta por la simplicidad, versatilidad y rapidez de desarrollo.

Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad.

En los últimos años el lenguaje se ha hecho muy popular, gracias a varias razones como:

* La cantidad de librerías que contiene, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero.
* La sencillez y velocidad con la que se crean los programas. Un programa en Python puede tener de 3 a 5 líneas de código menos que su equivalente en Java o C.
* La cantidad de plataformas en las que podemos desarrollar, como Unix, Windows, OS/2, Mac, Amiga y otros.
* Además, Python es gratuito, incluso para propósitos empresariales.

**Historia**

Python fue creado a finales de los ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.3

El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos Monty Python.

En 1991, van Rossum publicó el código de la versión 0.9.0 en alt.sources. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y los tipos modulares, como: str, list, dict, entre otros. Además en este lanzamiento inicial aparecía un sistema de módulos adoptado de Modula-3; van Rossum describe el módulo como “una de las mayores unidades de programación de Python”.

El modelo de excepciones en Python es parecido al de Modula-3, con la adición de una cláusula else. En el año 1994 se formó comp.lang.python, el foro de discusión principal de Python, marcando un hito en el crecimiento del grupo de usuarios de este lenguaje.

Python alcanzó la versión 1.0 en enero de 1994. Una característica de este lanzamiento fueron las herramientas de la programación funcional: lambda, reduce, filter y map. Van Rossum explicó que “hace 12 años, Python adquirió lambda, reduce(), filter() y map(), cortesía de un hacker de Lisp que las extrañaba y que envió parches”. El donante fue Amrit Prem; no se hace ninguna mención específica de cualquier herencia de Lisp en las notas de lanzamiento.

La última versión liberada proveniente de CWI fue Python 1.2. En 1995, van Rossum continuó su trabajo en Python en la Corporation for National Research Initiatives (CNRI) en Reston, Virginia, donde lanzó varias versiones del software.

Durante su estancia en CNRI, van Rossum lanzó la iniciativa Computer Programming for Everybody (CP4E), con el fin de hacer la programación más accesible a más gente, con un nivel de 'alfabetización' básico en lenguajes de programación, similar a la alfabetización básica en inglés y habilidades matemáticas necesarias por muchos trabajadores. Python tuvo un papel crucial en este proceso: debido a su orientación hacia una sintaxis limpia, ya era idóneo, y las metas de CP4E presentaban similitudes con su predecesor, ABC. El proyecto fue patrocinado por DARPA. En el año 2007, el proyecto CP4E está inactivo, y mientras Python intenta ser fácil de aprender y no muy arcano en su sintaxis y semántica, alcanzando a los no-programadores, no es una preocupación activa.

En el año 2000, el equipo principal de desarrolladores de Python se cambió a BeOpen.com para formar el equipo BeOpen PythonLabs. CNRI pidió que la versión 1.6 fuera pública, continuando su desarrollo hasta que el equipo de desarrollo abandonó CNRI; su programa de lanzamiento y el de la versión 2.0 tenían una significativa cantidad de traslapo. Python 2.0 fue el primer y único lanzamiento de BeOpen.com. Después que Python 2.0 fuera publicado por BeOpen.com, Guido van Rossum y los otros desarrolladores de PythonLabs se unieron en Digital Creations.

Python 2.0 tomó una característica mayor del lenguaje de programación funcional Haskell: listas por comprensión. La sintaxis de Python para esta construcción es muy similar a la de Haskell, salvo por la preferencia de los caracteres de puntuación en Haskell, y la preferencia de Python por palabras claves alfabéticas. Python 2.0 introdujo además un sistema de recolección de basura capaz de recolectar referencias cíclicas.

Posterior a este doble lanzamiento, y después que van Rossum dejó CNRI para trabajar con desarrolladores de software comercial, quedó claro que la opción de usar Python con software disponible bajo GNU GPL era muy deseable. La licencia usada entonces, la Python License, incluía una cláusula estipulando que la licencia estaba gobernada por el estado de Virginia, por lo que, bajo la óptica de los abogados de Free Software Foundation (FSF), se hacía incompatible con GPL. CNRI y FSF se relacionaron para cambiar la licencia de software libre de Python para hacerla compatible con GPL. En el año 2001, van Rossum fue premiado con FSF Award for the Advancement of Free Software.

Python 1.6.1 es esencialmente el mismo que Python 1.6, con unos pocos arreglos de bugs, y con una nueva licencia compatible con GPL.1

Python 2.1 fue un trabajo derivado de Python 1.6.1, así como también de Python 2.0. Su licencia fue renombrada a: Python Software Foundation License. Todo el código, documentación y especificaciones añadidas, desde la fecha del lanzamiento de la versión alfa de Python 2.1, tiene como dueño a Python Software Foundation (PSF), una organización sin ánimo de lucro fundada en el año 2001, tomando como modelo la Apache Software Foundation. Incluido en este lanzamiento fue una implementación del scoping más parecida a las reglas de static scoping (del cual Scheme es el originador).

Una innovación mayor en Python 2.2 fue la unificación de los tipos en Python (tipos escritos en C), y clases (tipos escritos en Python) dentro de una jerarquía. Esa unificación logró un modelo de objetos de Python puro y consistente.

También fueron agregados los generadores que fueron inspirados por el lenguaje Icon.

Las adiciones a la biblioteca estándar de Python y las decisiones sintácticas fueron influenciadas fuertemente por Java en algunos casos: el package logging, introducido en la versión 2.3, está basado en log4j; el parser SAX, introducido en 2.0; el package threading, cuya clase Thread expone un subconjunto de la interfaz de la clase homónima en Java.

**Características del lenguaje**

* Propósito general

Se pueden crear todo tipo de programas. No es un lenguaje creado específicamente para la web, aunque entre sus posibilidades sí se encuentra el desarrollo de páginas.

* Multiplataforma

Hay versiones disponibles de Python en muchos sistemas informáticos distintos. Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él.

* Interpretado

Quiere decir que no se debe compilar el código antes de su ejecución. En realidad sí que se realiza una compilación, pero esta se realiza de manera transparente para el programador. En ciertos casos, cuando se ejecuta por primera vez un código, se producen unos bytecodes que se guardan en el sistema y que sirven para acelerar la compilación implícita que realiza el intérprete cada vez que se ejecuta el mismo código.

* Interactivo

Python dispone de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que puede ayudarnos a entender mejor el lenguaje y probar los resultados de la ejecución de porciones de código rápidamente.

* Orientado a Objetos

La programación orientada a objetos está soportada en Python y ofrece en muchos casos una manera sencilla de crear programas con componentes reutilizables.

* Funciones y librerías

Dispone de muchas funciones incorporadas en el propio lenguaje, para el tratamiento de strings, números, archivos, etc. Además, existen muchas librerías que podemos importar en los programas para tratar temas específicos como la programación de ventanas o sistemas en red o cosas tan interesantes como crear archivos comprimidos en .zip.

* Sintaxis clara

Por último, destacar que Python tiene una sintaxis muy visual, gracias a una notación identada (con márgenes) de obligado cumplimiento. En muchos lenguajes, para separar porciones de código, se utilizan elementos como las llaves o las palabras clave begin y end. Para separar las porciones de código en Python se debe tabular hacia dentro, colocando un margen al código que iría dentro de una función o un bucle. Esto ayuda a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona  tengan un aspecto muy similar.

**Filosofía**

Los usuarios de Python se refieren a menudo a la Filosofía Python que es bastante análoga a la filosofía de Unix. El código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico". Contrariamente, el código opaco u ofuscado es bautizado como "no pythonico" ("unpythonic" en inglés). Estos principios fueron famosamente descritos por el desarrollador de Python Tim Peters en El Zen de Python

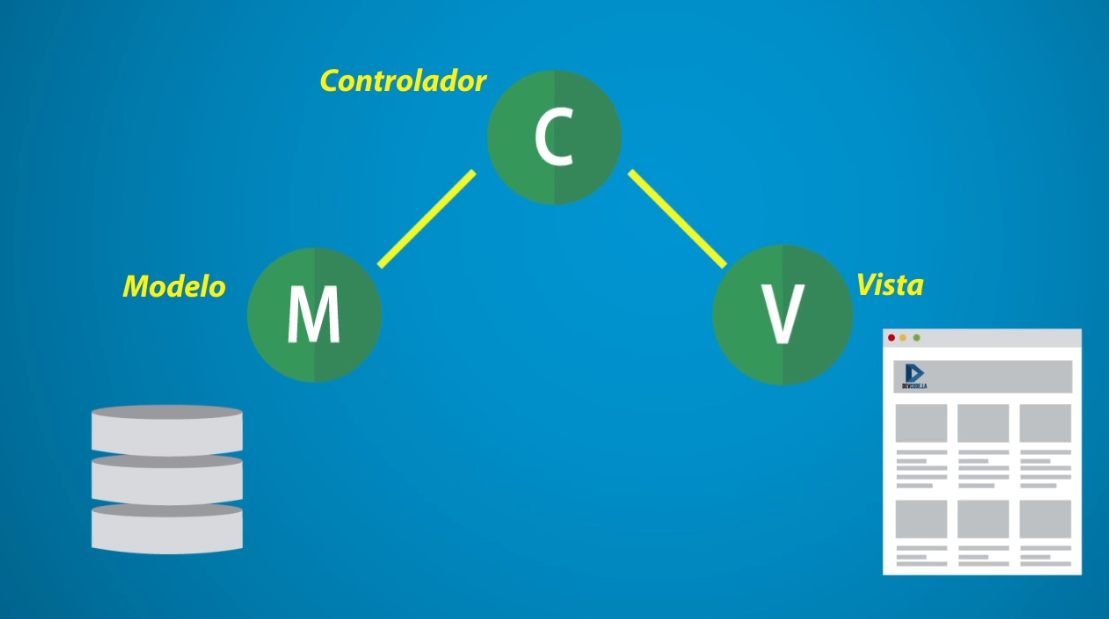
* Bello es mejor que feo.
* Explícito es mejor que implícito.
* Simple es mejor que complejo.
* Complejo es mejor que complicado.
* Plano es mejor que anidado.
* Disperso es mejor que denso.
* La legibilidad cuenta.
* Los casos especiales no son tan especiales como para quebrantar las reglas.
* Aunque lo práctico gana a la pureza.
* Los errores nunca deberían dejarse pasar silenciosamente.
* A menos que hayan sido silenciados explícitamente.
* Frente a la ambigüedad, rechaza la tentación de adivinar.
* Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
* Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
* Ahora es mejor que nunca.
* Aunque nunca es a menudo mejor que ya mismo.
* Si la implementación es difícil de explicar, es una mala idea.
* Si la implementación es fácil de explicar, puede que sea una buena idea.
* Los espacios de nombres (namespaces) son una gran idea.

**Django**

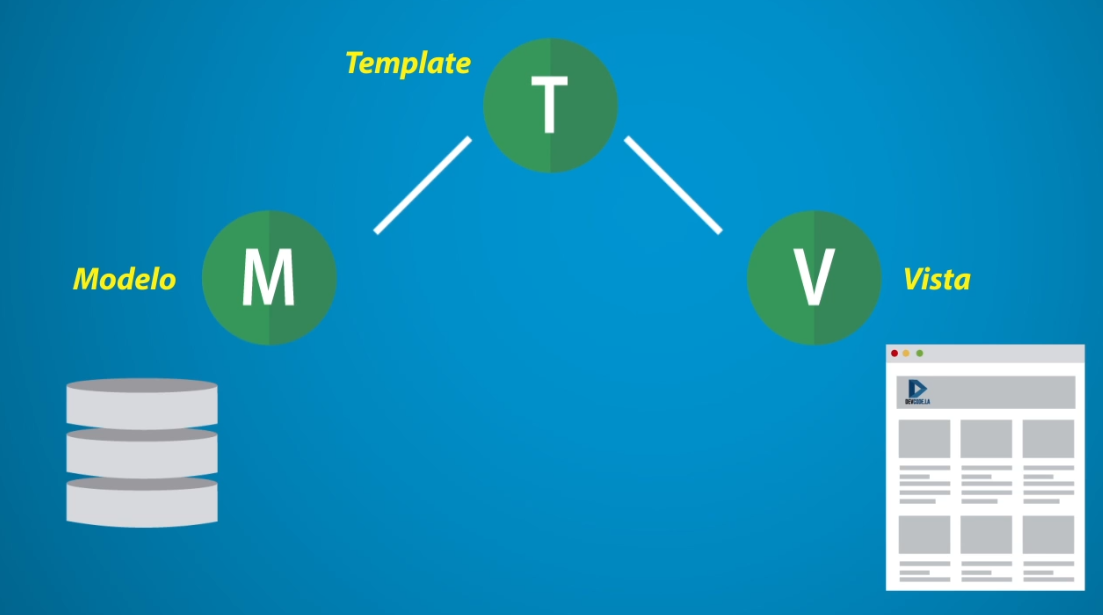
Este framework surge gracias a un grupo de personas que realizaban desarrollo para un diario llamado "Lawrence Journal World", estas personas se dieron cuenta que había código repetido y funciones que necesitaban en otros proyectos, los problemas que se les presentaban ya se habían solucionado anteriormente, en otras palabras todo era un caos. A partir de esto elaboran Django con el objetivo de no repetir código "Don,t repeat yourself", entre sus principales ventajas se encuentran:

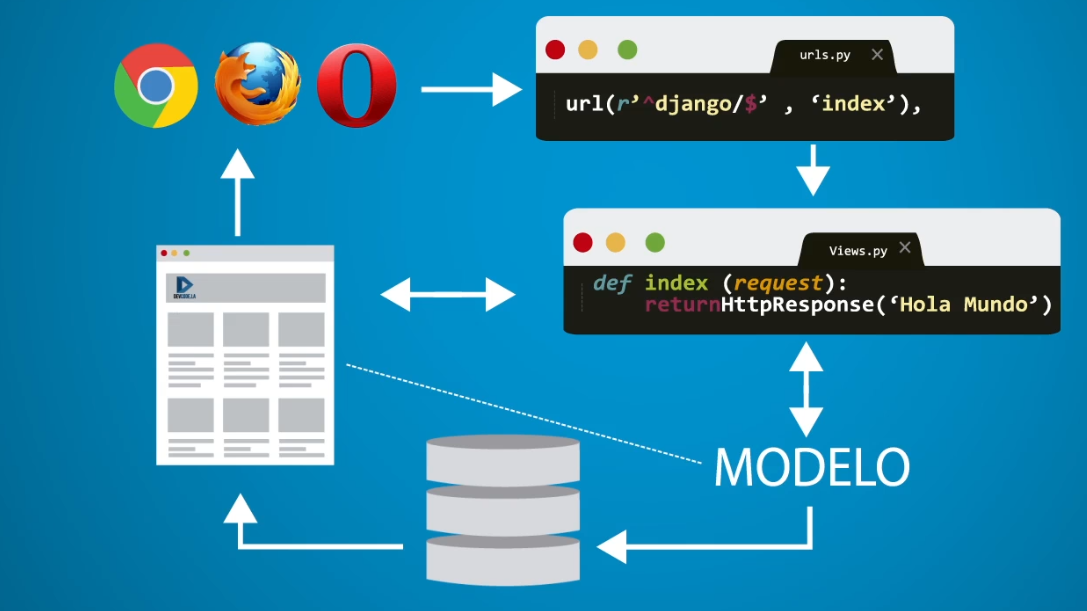
* Permite construir aplicaciones web mas rápido y con menos código,
* Brinda un sistema de administración para la aplicación.
* Urls limpias y amigables.
* Una poderos ORM.

Se apoya en el patrón de desarrollo MTV Modelo-Vista-Template, el cual es una modificación del famoso patrón MVC Modelo - vista – Controlador el cual separa los datos y la lógica de negocio de la interfaz de usuario, el modelo se encarga de manipular la información de la base de datos, la vista decide como mostrar la información en el navegador web y el controlador se encarga de la comunicación entre el modelo y la vista.

***Imagen 1.*** *Patron Model-Vista-Controlador*

En django todo es un poco diferente debido a que hicieron una modificación de MVC, el cual llamaron Model-Vista-Template. En donde el modelo es el que manipula la información de la base de datos, la vista decide que información va a mostrar y elige en que template se visualizara esta información y el template se encarga de manipular la información que viene de la vista, organizarla y ver la manera en que se va a mostrar en el navegador

***Imagen 2.*** *Patron Model-Vista-Template*

 ***Imagen 3.*** *Funcionamiento iluuustrado de django.*

¿Por qué Python y Django?

Python es un lenguaje de programación que estuvo en constante crecimiento en los últimos años, si lo comparamos con otros lenguajes podemos observar que su sintaxis es muy simple, lo cual hace que la programación sea más limpia, manejable y divertida. Estas virtudes más un framework tan poderoso como django hace del desarrollo web algo sencillo.

**Proyectos construidos en Django**



**Instagram**

Red social comprada por Facebook la cual permite subir fotografías, compartirlas y aplicar efectos en ellas. Recibe 40 millones de fotos por día y posee 90millones de usuarios activos.



**Disqus**

Servicio en línea que ofrece un sistema de discusiones y comentarios el cual puede ser agregado en nuestros sistemas. Realiza 45mil peticiones por segundo y posee 90mil millones de usuario por mes.



**Pinterest**

Red social para compartir imágenes que permite a los usuarios crear y administrar en tableros personales temáticos colecciones de imágenes como eventos, intereses, hobbies y mucho más. Posee 70millones de usuarios activos.



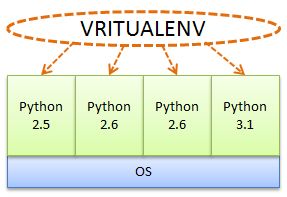
**Openstack**

Es un proyecto de [computación en la nube](http://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube), de [software libre](http://es.wikipedia.org/wiki/Software_libre) y de [código abierto](http://es.wikipedia.org/wiki/C%C3%B3digo_abierto). Es una comunidad nueva la cual posee 10mil miembros y mil colaboradores activos.



**Mozilla Firefox**

Es un navegador web gratuito, creado por una organización sin fines de lucros, y es actualmente utilizado por muchísimas personas.

**Virtualenviroment**

Es un sistema que administra y controla múltiples instancias de entornos virtuales y que sirve para que las dependencias de nuestros proyectos sean independientes unas de otras, evitando así conflictos entre ellas.

Podemos tener varios proyectos, trabajando en cada uno con versiones de django y python totalmente diferente gracias a esta herramienta.



Bibliografía para postgres: http://www.postgresql.org.es/

**PostgreSQL**

Introducción

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarle a otras bases de datos comerciales.

PostgreSQL utiliza un modelo cliente/servidor y usa *multiprocesos* en vez de *multihilos* para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando.

Componentes más importantes en un sistema PostgreSQL.

* **Aplicación cliente**: Esta es la aplicación cliente que utiliza PostgreSQL como administrador de bases de datos. La conexión puede ocurrir via TCP/IP ó sockets locales.
* **Demonio postmaster**: Este es el proceso principal de PostgreSQL. Es el encargado de escuchar por un puerto/socket por conexiones entrantes de clientes. También es el encargado de crear los procesos hijos que se encargaran de autentificar estas peticiones, gestionar las consultas y mandar los resultados a las aplicaciones clientes
* **Ficheros de configuración**: Los 3 ficheros principales de configuración utilizados por PostgreSQL, postgresql.conf, pg\_hba.conf y pg\_ident.conf
* **Procesos hijos postgres**: Procesos hijos que se encargan de autentificar a los clientes, de gestionar las consultas y mandar los resultados a las aplicaciones clientes
* **PostgreSQL share buffer cache**: Memoria compartida usada por POstgreSQL para almacenar datos en caché.
* **Write-Ahead Log (WAL)**: Componente del sistema encargado de asegurar la integridad de los datos (recuperación de tipo REDO)
* **Kernel disk buffer cache**: Caché de disco del sistema operativo
* **Disco**: Disco físico donde se almacenan los datos y toda la información necesaria para que PostgreSQL funcione.

**Características**

La última serie de producción es la 9.3. Sus características técnicas la hacen una de las bases de datos más potentes y robustas del mercado. Su desarrollo comenzó hace más de 16 años, y durante este tiempo, *estabilidad, potencia, robustez, facilidad de administración e implementación de estándares* han sido las características que más se han tenido en cuenta durante su desarrollo. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema.

A continuación alguna de las características más importantes y soportadas por PostgreSQL.

#### Generales

* Es una base de datos 100% [ACID](http://es.wikipedia.org/wiki/ACID)
* Integridad referencial
* Tablespaces
* Nested transactions (savepoints)
* Replicación asincrónica/sincrónica / Streaming replication - Hot Standby
* Two-phase commit
* PITR - point in time recovery
* Copias de seguridad en caliente (Online/hot backups)
* Unicode
* Juegos de caracteres internacionales
* Regionalización por columna
* Multi-Version Concurrency Control (MVCC)
* Multiples métodos de autentificación
* Acceso encriptado via SSL
* Actualización in-situ integrada (pg\_upgrade)
* SE-postgres
* Completa documentación
* Licencia BSD
* Disponible para Linux y UNIX en todas sus variantes (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) y Windows 32/64bit.

#### Programación / Desarrollo

* Funciones/procedimientos almacenados (stored procedures) en numerosos lenguajes de programación, entre otros PL/pgSQL (similar al PL/SQL de oracle), PL/Perl, PL/Python y PL/Tcl
* Bloques anónimos de código de procedimientos (sentencias DO)
* Numerosos tipos de datos y posibilidad de definir nuevos tipos. Además de los tipos estándares en cualquier base de datos, tenemos disponibles, entre otros, tipos geométricos, de direcciones de red, de cadenas binarias, UUID, XML, matrices, etc
* Soporta el almacenamiento de objetos binarios grandes (gráficos, videos, sonido, ...)
* APIs para programar en C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, PHP, Lisp, Scheme, Qt y muchos otros.

#### SQL

* SQL92,SQL99,SQL2003,SQL2008
* Llaves primarias (primary keys) y foráneas (foreign keys)
* Check, Unique y Not null constraints
* Restricciones de unicidad postergables (deferrable constraints)
* Columnas auto-incrementales
* Indices compuestos, únicos, parciales y funcionales en cualquiera de los metodos de almacenamiento disponibles, B-tree, R-tree, hash ó GiST
* Sub-selects
* Consultas recursivas
* Funciones 'Windows'
* Joins
* Vistas (views)
* Disparadores (triggers) comunes, por columna, condicionales.
* Reglas (Rules)
* Herencia de tablas (Inheritance)
* Eventos LISTEN/NOTIFY

Algunos de los límites de PostgreSQL son:

|  |  |
| --- | --- |
| Límite | Valor |
| Máximo tamaño base de dato | Ilimitado (Depende de tu sistema de almacenamiento) |
| Máximo tamaño de tabla | 32 TB |
| Máximo tamaño de fila | 1.6 TB |
| Máximo tamaño de campo | 1 GB |
| Máximo número de filas por tabla | Ilimitado |
| Máximo número de columnas por tabla | 250 - 1600 (dependiendo del tipo) |
| Máximo número de índices por tabla | Ilimitado |

**Historia**

El *proyecto PostgreSQL* tal y como lo conocemos hoy en día empezó en 1996, aunque las bases y el trabajo en la que se asienta tienen sus comienzos en la década de los 70. A continuación se presenta una descripción de la historia de PostgreSQL.

#### Ingres 1977-1985 - *"El comienzo"*

La década de los 70 fue una década de desarrollos y pruebas de nuevos conceptos en el nuevo mundo de los gestores de bases de datos.

IBM había estado trabajando desde 1973 con los primeros conceptos, ideas y teorías sobre bases de datos relacionales. Su proyecto "System R" fue entre otras cosas la primera implementación del lenguaje SQL (Structured Query Language). Este proyecto, sus decisiones de diseño y muchos de los algoritmos usados, influenciaron muchos de los sistemas de bases de datos relacionales que aparecieron posteriormente.

Por aquel entonces un profesor de la Universidad de Berkeley, Michael Stonebraker, leyó unos artículos publicados por IBM sobre "System R" que le hicieron interesarse en el tema. Utilizando el dinero de otro proyecto que ya tenía asignado, Ingres (INteractive Graphics REtrieval System), Stonebraker empezó a desarrollar sus ideas sobre bases de datos relacionales. Durante estos años Ingres mantuvo su código fuente abierto y permaneció en gran medida similar en conceptos a "System R".

A principio de los 80, Ingres estuvo compitiendo con Oracle por el liderazgo en el mundo de bases de datos relacionales y su código e implementación evolucionaron y fueron el origen de otras bases de datos relacionales, entre ellas podemos citar a Informix, NonStop SQL y Sybase (Microsoft SQL Server fue una versión licenciada de Sybase hasta su version 6.0).

Michael Stonebraker dejo la Universidad de Berkeley en 1982 para comercializar Ingres pero volvió a la misma en 1985 con nuevas ideas.

#### Postgres 1986-1994 - *Despues (post) de ingres*

Después de su vuelta a Berkeley en 1985, Michael Stonebraker lideró un nuevo proyecto llamado *Postgres* (después de Ingres) patrocinado por la *Defense Advanced Research Projects Agency (DARPA)*, la *Army Research Office (ARO)*, la *National Science Foundation (NSF)*, y *ESL, Inc*. Con este proyecto y basándose en la experiencia obtenida con Ingres, Stonebraker tenía como meta mejorar lo que habían conseguido y aprendido en el desarrollo de Ingres. Y aunque se baso en muchas ideas de Ingres, no se baso en el código fuente del mismo.

Los objetivos iníciales de este proyecto fueron:

* Proporcionar un mejor soporte para objetos complejos
* Proporcionar a los usuarios la posibilidad de extender los tipos de datos, operadores y métodos de acceso.
* Proporcionar los mecanismos necesarios para crear bases de datos activas (triggers, etc)
* Simplificar el código encargado de la recuperación del sistema después de una caída del mismo
* Hacer cambios mínimos (preferiblemente ninguno) en el modelo relacional.
* Mejorar el lenguaje de consulta QUEL heredado de Ingres (POSTQUEL).

La última versión de Postgres en este proyecto fue la versión 4.2.

#### Postgres95 1994-1995 - *Nueva vida en el mundo opensource*

En 1994, dos estudiantes de Berkeley, Andrew Yu y Jolly Chen, empezaron a trabajar con el código de Postgres (versión 4.2) y llamaron al proyecto Postgres95. Hicieron una limpieza general del código, arreglaron errores en el mismo, e implementaron otras mejoras, entre las que destacan:

* Sustitución de POSTQUEL por un intérprete del lenguaje SQL
* Reimplementación de las funciones agregadas
* psql fue creado para ejecutar consultas SQL
* El interface de objetos grandes (large-object) fue revisado
* Un pequeño tutorial sobre Postgres fue creado
* Postgres se pudo empezar a compilar con GNU make y GCC sin parchear

La versión 1.0 de Postgre95 vio la luz en 1995, el código era 100% ANSI C, un 25% más corto en relación con la versión 4.2 y un 30-50% más rápido. El código fue publicado en la web y liberado bajo una licencia BSD, y más y más personas empezaron a utilizar y a colaborar en el proyecto.

#### PostgreSQL 1996-actualidad - *Proyecto PostgreSQL*

En 1996, Andrew Yu y Jolly Chen ya no tenían tanto tiempo para dirigir y desarrollar Postgres95. Algunos de los usuarios habituales de las listas de correo del proyecto decidieron hacerse cargo del mismo y crearon el llamado *"PostgreSQL Global Development Team"*.

En un principio este equipo de desarrolladores al cargo de la organización del proyecto estuvo formado por Marc Fournier en Ontario, Canadá, Thomas Lockhart en Pasadena, California, Vadim Mikheev en Krasnoyarsk, Rusia y Bruce Momjian in Philadelphia, Pennsylvania. El nombre fue cambiado de Postgres95 a PostgreSQL y lanzaron la versión 6.0 en enero de 1997.

Hoy en día el grupo central (core team) de desarrolladores está formado por 6 personas, existen 38 desarrolladores principales y más 21 desarrolladores habituales. En total alrededor de 65 personas activas, contribuyendo con el desarrollo de PostgreSQL.

Existe también una gran comunidad de usuarios, programadores y administradores que colaboran activamente en numerosos aspectos y actividades relacionadas con el proyecto. Informes y soluciones de problemas, tests, comprobación del funcionamiento, aportaciones de nuevas ideas, discusiones sobre características y problemas, documentación y fomento de PostgreSQL son solo algunas de las actividades que la comunidad de usuarios realiza.

No tenemos que olvidar tampoco que existen muchas empresas que también colaboran con dinero y/ó con tiempo/personas en mejorar PostgreSQL. Muchos desarrolladores y nuevas características están muchas veces patrocinadas por empresas privadas.

En los últimos años los trabajos de desarrollo se han concentrado mucho en la velocidad de proceso y en características demandadas en el mundo empresarial. Durante los años de existencia del Proyecto PostgreSQL, el tamaño del mismo, tanto en número de desarrolladores, como en números de línea de código, funciones y complejidad del mismo ha ido aumentando año a año.

Según COCOMOII, obtendríamos estos números para PostgreSQL 9.0.0:

|  |  |
| --- | --- |
| Descripción | Valor |
| Números de líneas de código (PG-9.0.0) | 969.562 |
| Habilidad de los programadores (alta) | 0,6 |
| Complejidad del proyecto (alta) | 1,24 |
| Precio/hora ($100.000/año - 1.875horas/año) | $53,3 |
|  |  |
| Programadores-año | 618,71 |
| Precio por línea de código | $65,30 |
| Precio Total | $63.316.697 |
| Líneas de código por persona/día | 7 |
| Tiempo de desarrollo del proyecto (años) | 3.6 |
| Número medio de programadores | 171,4 |