Universidad de las Ciencias Informáticas

Facultad 6

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Título: "Herramienta para la generación de datos ficticios en bases de datos PostgreSQL y MySQL.**"**

Autores:

Yordaika Verdecia Agüero.

Meyker Rodríguez Leyva.

Tutores:

Ing. Mikel Díaz Hernández.

Ing. Flavio Enrique Roche Rodríguez

La Habana, Junio de 2014

"El individualismo debe ser, mañana, la realización completa de las capacidades de todo un individuo en beneficio absoluto de una colectividad."

Ernesto Che Guevara

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis que tiene por título: y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Yordaika Verdecia Agüero.

_____

Firma del Autor

Meyker Rodríguez Leyva

_____

Firma del Autor

Ing. Mikel Díaz Hernández.

_____

Firma del Tutor

Ing. Flavio Enrique Roche Rodríguez

_____

Firma del Tutor

DATOS DE CONTACTO

Ing. Flavio Enrique Roche
Especialidad de graduación: Ingeniero en Ciencias Informáticas
e-mail: feroche@uci.cu

Ing. Mikel Díaz Hernández
Especialidad de graduación: Ingeniero en Ciencias Informáticas
e-mail: mdiazhdez@uci.cu

DEDICATORIA

A mi mamá, a mis hermanitas Dariana y Dariannet, y demás familiares y amigos que confiaron en que podía seguir adelante.
Yordaika.

A mis abuelos Tito y Amparo, a mis sobrinos, a mi mamá y a mi novia.
Meyker.

AGRADECIMIENTOS

Agradezco grandemente a mis padres que siempre han estado dispuestos a realizar cualquier sacrificio por mí, les agradezco mucho todo su apoyo y amor incondicional.

A mi abuelito Manuel por quererme como a una más de sus hijas.

A mi tía Paquita por su preocupación y apoyo constante.

A mis tías Yamicela y Adis por su cariño y amor hacia mí.

Agradezco también a todos mis compañeros de aula y a los profes Jorge Emilio y Garnache.

A mis tutores por su apoyo y ayuda.

A mi compañero de tesis por su compañerismo y por los días de fuerte trabajo.

Quiero agradecer especialmente a mi novio, sin su apoyo, comprensión, amor y cariño, estos 5 años de estudio hubieran sido muy difíciles para mí.

Yordaika.

Agradezco a mi mamá y a mi novia sin ellas no hubiese llegado hasta aquí.

A mi familia Michel, Yohansy, Branly, a mi papá, a Viviana, a Isabel gracias por el cariño y la preocupación constante.

A mis amigos Dariel, Alfonso, Jávico, Yadian, Alfredo, Carlos, Bofill gracias por considerarme amigos de ustedes.

A la Choty, a Flopy, Dunia, Katy, Haileen, Roxana, Yuni y Lili gracias por haberme soportado.

A todos los profesores que me ayudaron en mi formación y en especial al profe Omar que lo considero además un amigo, gracias.

Y a Yordi por tanto esfuerzo.

Meyker.

RESUMEN

En la actualidad se produce un gran número de productos informáticos que trabajan con los gestores de bases de datos PostgreSQL y MySQL. Antes de que estos productos sean liberados y entregados al usuario final son sometidos a un proceso de prueba donde se verifica que los datos contenidos en las bases de datos coinciden con los que realmente deben manejar. Para la realización de estas pruebas los desarrolladores recurren a herramientas generadoras de datos ficticios, pues generar la información suficiente para poblar las bases de datos de forma manual sería un trabajo imperioso y requeriría un gasto de tiempo considerable. El presente trabajo está enfocado precisamente en el desarrollo de una herramienta que permite la generación de datos ficticios sobre las bases de datos PostgreSQL y MySQL. Para ello se realizó un estudio del funcionamiento de las aplicaciones ya existentes que realizan el proceso de generación de datos para los distintos gestores de base de datos. Como resultado se obtuvo una herramienta que permite poblar tanto bases de datos de PostgreSQL como de MySQL, dando la posibilidad al usuario de escoger el tipo de generación que considere más factible y configurar los parámetros para la generación. El desarrollo de esta herramienta constituye una alternativa de gran ayuda en la realización de pruebas a los productos de software desarrollados en los centros productivos de la Universidad de Ciencias Informáticas y específicamente en el Centro de Tecnologías de Gestión de Datos (DATEC).

PALABRAS CLAVES: PostgreSQL, MySQL, base de datos, datos ficticios.

ABSTRACT

At present, it is produced a large number of software products that work with PostgreSQL and MySQL database management systems (DBMS). Before these products are released and delivered to the final user they undergo a testing process where it is verified that the data contained in the databases match those actually handle. To carry out these tests, developers turn to fictitious data generating tools, since generating sufficient information to populate the databases manually would be a compelling work and would require a considerable waste of time. This paper focuses specifically on the development of a tool that allows generating fictitious data on PostgreSQL and MySQL databases. As for that, it was made a study of the functioning of existing applications that make the process of data generation for different DBMS. The result was a tool that allows populating both PostgreSQL and MySQL databases, giving the user the possibility to choose the type of generation considered the most feasible and set the parameters for it. The development of this tool is an alternative of great help in testing the software products developed in the production centers at the University of Informatics Sciences and specifically in the Data Management Technology Center (DATEC).

KEY WORDS: PostgreSQL, MySQL, database, fake data.

TABLA DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La información es en la actualidad uno de los pilares fundamentales para la producción, el comercio, la inversión, la transferencia de tecnología y los negocios. Hoy en día es una necesidad apremiante para las empresas guardar dicha información de forma rápida y fácil, pues es fundamental garantizar que la misma se encuentre lo más organizada posible y garantizar así que cualquier persona que la necesite pueda acceder a ella en cualquier momento.

Con el desarrollo acelerado de la informática surgen los llamados sistemas de información, con el paso del tiempo y el perfeccionamiento de las técnicas han llegado a conocerse en la actualidad como Bases de Datos (BD), en las cuales se almacenan grandes volúmenes de información. Se le llama base de datos precisamente a una colección de datos recopilados y estructurados que existen durante un período de tiempo. Estas tienen mayor calidad, flexibilidad y manejo desde la creación de los Sistemas Gestores de Bases de Datos (SGBD), los cuales permiten el almacenamiento modificación y extracción de la información. Entre los programas que permiten la gestión de las bases de datos se encuentran los gestores PostgreSQL y MySQL, ambos son muy utilizados mundialmente por la rica colección de funcionalidades que brindan al usuario.

La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la misma. Las bases de datos constituyen una de las herramientas más difundidas en la sociedad. Estas han sido utilizadas especialmente como fuentes de consulta y se ha encontrado en ellas una herramienta importante para el desarrollo del conocimiento.

En la actualidad el uso de los datos almacenados en BD para la toma de decisiones se hace una necesidad, por tal motivo la cantidad de datos que se utilizan incrementan exponencialmente cada día. Almacenar estos datos es una tarea imperiosa, así como utilizarlos en determinado grado. Un punto crítico en el trabajo con BD es la realización de pruebas antes del despliegue pues el comportamiento de las BD generalmente no es el mismo en desarrollo que en producción, ya que el rendimiento en las etapas iniciales no es similar a cuando hay gran volumen de datos y consultas concurrentes. En el Centro de Tecnologías de Gestión de Datos (DATEC) de la Universidad de las Ciencias Informáticas se producen software que implican el manejo de datos, tanto de PostgreSQL como de MySQL, por lo que antes del proceso de despliegue es necesario determinar si el volumen de datos de la BD creada se corresponde

con el que realmente manejará la aplicación; para esto los desarrolladores precisan generar un elevado volumen de datos que se asemeje a la realidad, crear dicho volumen manualmente llevaría un gasto de tiempo considerable.

Con el objetivo de minimizar tiempo y recursos en el proceso de pruebas a las BD se crean las herramientas generadoras de datos. Estas se encargan de generar grandes volúmenes de información ficticia, la cual es consistente pero no tiene ningún valor real. Las herramientas generadoras de datos evitan el difícil trabajo de insertar manualmente los datos en las tablas; en las que se pueden introducir valores que en la aplicación final pueden estar validados de manera distinta, provocando errores a la hora de consultar los datos. Además favorecen los tiempos de entrega del producto informático pues muchos de estos no responden de la manera esperada cuando tienen que procesar grandes cúmulos de información.

En el centro DATEC, en muchas ocasiones los clientes han expresado inconformidades por errores detectados a la hora de poner en funcionamiento los productos entregados. Estos errores son muy comunes en las aplicaciones que requieren el manejo de datos pues el comportamiento de estas nunca va a ser igual en la etapa de desarrollo que luego del despliegue, donde son puestas en funcionamiento con volúmenes de datos reales. Por esta razón es importante la realización de pruebas a las bases de datos de estos productos. Actualmente las herramientas generadoras de datos, para la realización de pruebas, con las que se cuenta en el centro no son libres ni de código abierto y no cuentan con funcionalidades específicas que necesita el desarrollador. Además para realizar las pruebas a las bases de datos de PostgreSQL y de MySQL se requiere del uso de varias herramientas generadoras de datos lo que provoca atrasos en la entrega de los productos al cliente final.

Teniendo en cuenta los elementos anteriormente planteados se define como **problema de investigación**: ¿Cómo permitir la generación de datos ficticios sobre las Bases de Datos PostgreSQL y MySQL para facilitar la realización de pruebas durante el desarrollo de software?

Objeto de estudio: Proceso de generación de datos ficticios, enmarcado en el **campo de acción**: La generación de datos ficticios sobre las bases de datos PostgreSQL y MySQL.

Para dar solución al problema antes planteado se define como **objetivo general**: Desarrollar una herramienta que permita la generación de datos ficticios sobre las bases de datos PostgreSQL y MySQL.

Para satisfacer el objetivo general se definen los siguientes **objetivos específicos**:

- Caracterizar las herramientas y algoritmos que realizan el proceso de generación de datos ficticios.
- Implementar la herramienta para la generación de datos ficticios sobre las bases de datos PostgreSQL y MySQL.
- Validar la herramienta para la generación de datos ficticios sobre las bases de datos PostgreSQL y MySQL.

Con el objetivo de dar cumplimiento a los objetivos especificados anteriormente se realizarán las siguientes **tareas de investigación**:

- Caracterización de las herramientas existentes que realizan la generación de datos ficticios.
- Identificación de las funcionalidades de la herramienta para la generación de datos ficticios.
- Elaboración de las Historias de Usuario de la herramienta a desarrollar.
- Elaboración del modelo de dominio para el proceso de generación de datos ficticios.
- Elaboración las Tarjetas CRC de la herramienta para la generación de datos ficticios.
- Implementación de la herramienta para la generación de datos ficticios.
- Identificación de las tareas de ingeniería para la implementación de la herramienta generadora de datos.
- Validación de la implementación de la herramienta para la generación de datos ficticios.
- Elaboración de los caso de prueba para validación de la herramienta generadora de datos.

Durante el desarrollo del presente trabajo fueron utilizados los siguientes métodos de investigación:

- **Histórico-Lógico:** Este método permitió realizar un estudio para comprender el proceso de generación de datos y las tendencias actuales de las herramientas que realizan esta operación.
- **Analítico-Deductivo:** El empleo de este método permitió analizar la información y arribar a conclusiones en la investigación.

Estructura del trabajo

El presente trabajo está formado por: introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas, bibliografía, anexos y glosario de términos. A continuación se muestra una breve descripción de cada uno de los capítulos.

Capítulo 1: Fundamento teórico

Se describe el marco teórico del trabajo. Se presentan un grupo de conceptos para lograr un mejor entendimiento del trabajo a desarrollar. Además se realiza un estudio de la metodología, herramientas y lenguajes a utilizar en el desarrollo.

Capítulo 2: Diseño y características del sistema propuesto

Se caracteriza el sistema propuesto y se realiza toda la fase de diseño del negocio, teniendo como resultado las historias de usuario, lista de reserva del producto, tarjetas Clase-Responsabilidad-Colaboración entre otros artefactos que se generan a través del uso de la Metodología XP.

Capítulo 3: Implementación y pruebas

Se describe la implementación del sistema y se desarrollan las tareas de ingeniería. Además se definen los estándares de codificación con los que debe cumplir la aplicación y las pruebas a las que será sometido el sistema.

CAPÍTULO 1: FUNDAMENTO TEÓRICO

Introducción

El presente capítulo comprende el marco teórico del trabajo donde serán abordados los principales conceptos referentes al dominio del problema. Se realiza un estudio de las herramientas con funcionalidades similares a la solución propuesta. Se define la metodología de desarrollo de software que más se ajusta a las necesidades del trabajo y se hace además una selección y descripción de las herramientas y tecnologías necesarias para el desarrollo de la herramienta generadora de datos.

1.1 Conceptos asociados a la investigación.

En el presente trabajo se hará mención de algunos términos o conceptos importantes de la investigación que se realiza. A continuación de definen algunos de estos conceptos con el objetivo de evacuar las dudas que puedan surgir a cerca de estos.

Datos

Los datos son símbolos que describen condiciones, hechos, situaciones o valores. Estos se caracterizan por no contener ninguna información. Un dato puede significar un número, una letra, un signo ortográfico o cualquier símbolo que represente una cantidad, una medida, una palabra o una descripción. La importancia de los datos está en su capacidad de asociarse dentro de un contexto para convertirse en información. Por si mismos los datos no tienen capacidad de comunicar un significado y por tanto no pueden afectar el comportamiento de quien los recibe. Para ser útiles, los datos deben convertirse en información para ofrecer un significado, conocimiento, ideas o conclusiones. (1)

Base de Datos (BD)

Colección compartida de datos relacionados desde el punto de vista lógico, junto con una descripción de esos datos, diseñada para satisfacer las necesidades de información de una organización.(2)

El Dr. Christopher J. Date define a las BD como: un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa. (3)

Las bases de datos son un conjunto de datos almacenados entre los que existen relaciones lógicas y han sido diseñadas para satisfacer los requerimientos de información de una empresa u organización. En una base de datos, además de los datos, también se almacena su descripción. (4)

Teniendo en cuenta la definición de los autores Christopher y Mercedes Márquez puede concluirse que una BD está constituida por un conjunto de información almacenada y relacionada entre sí, el cual es utilizado por los sistemas de aplicación de una empresa para la toma de decisiones.

1.2 Sistemas Gestores de bases de Datos (SGBD).

Un SGBD es un programa de ordenador que facilita una serie de herramientas para manejar bases de datos y obtener resultados (información) de ellas. Además de almacenar la información, se le pueden hacer preguntas sobre esos datos, obtener listados impresos, generar pequeños programas de mantenimiento de la BD, o ser utilizado como servidor de datos para programas más complejos realizados en cualquier lenguaje de programación. (5)

En general, un SGBD es un software de BD que:

- Centraliza los datos en un único “lugar” lógico al que acceden todos los usuarios y aplicaciones.
- Es utilizable por múltiples usuarios y aplicaciones concurrentemente.
- Ofrece visiones parciales del conjunto total de información, según las necesidades de un usuario en particular.

Posee herramientas para asegurar:

- La independencia de datos a varios niveles: permitiendo la modificación de las definiciones de datos sin afectar a las aplicaciones o esquemas que no utilizan esos datos.
- La integridad de los datos: que los datos sean correctos en todo momento, de acuerdo con las especificaciones o reglas impuestas al sistema.
- La seguridad de los datos: que sólo las personas autorizadas puedan acceder a determinados datos y que sólo puedan efectuar las operaciones para las que han sido autorizados.

Teniendo en cuenta las definiciones anteriores se puede definir como SGBD a un conjunto de aplicaciones que permitan a los usuarios definir, crear y mantener las BD. Estas aplicaciones deben ser capaces de

administrar las BD y permitir además la interacción entre estas y los usuarios, haciendo posible acceder de forma organizada a los datos almacenados.

1.2.1 PostgreSQL

PostgreSQL es un SGBD distribuido bajo licencia BSD (*Berkeley Software Distribution o Distribución de Software Berkeley*) y su código fuente se encuentra disponible libremente. Es el sistema de gestión de bases de datos de código abierto más utilizado y en sus últimas versiones tiene características a la altura de otras bases de datos de corte comercial:

- PostgreSQL es un modelo Objeto-Relacional porque aproxima los datos a un modelo objeto-relacional y es capaz de manejar complejas rutinas y reglas. Como por ejemplo, soporta consultas SQL declarativas, control de concurrencia multiversión, soporte multiusuario, transacciones, optimización de consultas, herencia y arreglos.
- La integridad referencial es utilizada para garantizar la coherencia de datos entre relaciones aparejadas.
- PostgreSQL soporta lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL.
- La arquitectura cliente/servidor de PostgreSQL es similar el método del Apache 1.3.x para manejar procesos. Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectar a PostgreSQL. (6)

Sus características técnicas la hacen una de las bases de datos más utilizadas en el 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 al sistema.(7)

PostgreSQL se ejecuta en casi todos los principales sistemas operativos y su documentación está bien organizada, pública y libre. Proporciona soporte a las características de las BD profesionales (Ejemplo: disparadores, procedimientos almacenados, funciones, secuencias, relaciones, reglas, tipos de datos

definidos por usuarios, vistas, vistas materializadas) y es altamente adaptable a las necesidades del cliente. La atomicidad, consistencia, aislamiento y durabilidad son algunas de sus características.

1.2.2 MySQL

MySQL es un Sistema de Gestión de Bases de datos Relacionales distribuido por una licencia dual, lo cual significa que por una parte es libre, y por otra es comercial, es decir que por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. Aparte de las características que definen MySQL como programa *open-source*, existen aspectos que lo diferencian de otros productos como:

1. Posibilidad de crear y configurar usuarios, asignando a cada uno de ellos permisos diferentes.
2. Facilidad de exportación e importación de datos, incluso de la base de datos completa.
3. Posibilidad de ejecutar conjuntos de instrucciones guardadas en ficheros externos a la base de datos.
4. Permite escoger entre múltiples motores de almacenamiento para cada tabla.
5. Agrupación de transacciones, reuniendo múltiples transacciones de varias conexiones para incrementar el número de transacciones por segundo. (8)

MySQL carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. Poco a poco los elementos de los que carecía MySQL han estado siendo incorporados tanto por desarrollos internos, como por desarrolladores de software libre. Entre las características disponibles en las últimas versiones se puede destacar.

- Amplio subconjunto del lenguaje SQL.
- Disponibilidad en gran cantidad de plataformas y sistemas.
- Posibilidad de selección de mecanismos de almacenamiento que ofrecen diferente velocidad de operación, soporte físico, capacidad, distribución geográfica, transacciones...
- Transacciones y claves foráneas.
- Conectividad segura.
- Replicación.

- Búsqueda e indexación de campos de texto.

1.3 Generación de datos ficticios

La generación de datos ficticios consiste en la obtención de datos con consistencia pero sin valor real alguno, que permite simular un entorno parecido a la realidad. La generación de datos ficticios es de suma importancia en el desarrollo de pruebas a bases de datos, pues permite poblarlas con un volumen de datos, que de otra forma, llevarían consigo un gasto considerable de tiempo y de recursos.

En la actualidad generar datos es una práctica muy utilizada por los desarrolladores; y aunque no se ha difundido mucha información sobre el tema, el número de generadores de datos desarrollados sigue creciendo exponencialmente, ya sea para SQL, Oracle, MySQL y muchos otros Sistemas Gestores de Bases de Datos.

1.3.1 Herramientas de generación de datos ficticios.

Una herramienta generadora de datos es una aplicación que genera datos ficticios o de pruebas para las tablas de una o más bases de datos. Mundialmente existen muchas herramientas que realizan esta función. A continuación se muestra en una tabla algunas de estas herramientas con algunas de sus características.

Tabla 1. Herramientas Generadoras de Datos Ficticios.

Herramientas generadoras de datos ficticios	EMS Data Generator para PostgreSQL	EMS Data Generator para MySQL	Postgen	Datanamic Data Generator MultiDB
SGBD	1. PostgreSQL	1. MySQL	1. PostgreSQL	1. Oracle 2. MySQL 3. MS SQL Server 4. PostgreSQL
Plataforma	Desktop	Desktop	Desktop	Web

Ventajas	Ofrece una aplicación de consola, que permite generar datos mediante plantillas de generación. Ofrece una Interfaz de asistente fácil de usar.	Tiene la capacidad de generar datos en scripts en SQL. Permite efectuar una previsualización de los datos que se mostrarán al ejecutar el script.	Es multi-plataforma y de fácil manejo para el usuario.	Puede leer automáticamente la base de datos y detectar cuando una columna o tabla ha sido cambiada o suprimida y luego aplica estos cambios en las tablas del proyecto.
Desventajas	No es de código abierto y utiliza una Licencia hareware. No soporta los datos incluidos en las últimas versiones del gestor. Incompatibilidades con las últimas versiones de PostgreSQL. No tiene en cuenta las relaciones entre	No es libre ni de código abierto por lo hay que pagar por su licencia.	Sólo es capaz de generar datos de tipo texto y numérico.	Es una aplicación privativa.

	tablas, por lo que en el caso de que una tabla tenga una llave extranjera, no podrá ser llenada.			
Tipos de generación.	1- De una lista. 2- Incremental 3- Aleatoria (9)	1- Aleatoria (10)	1- Aleatoria (11)	1- Aleatoria 2- De una lista 3- De un archivo externo. 4- De otra tabla (12)

Teniendo en cuenta las características de las herramientas anteriormente estudiadas se desea desarrollar una herramienta que permita generar datos para las bases de datos de PostgreSQL y MySQL. Esta herramienta debe ser una aplicación desktop teniendo en cuenta que mayoría de las herramientas de este tipo son aplicaciones desktop. Se desea también que la herramienta permita por defecto generar datos de forma aleatoria ya que agiliza la generación pues normalmente los usuarios no desean configurar todos los atributos de cada tabla además las herramientas estudiadas permiten esta forma de generación, aunque también se quiere que genere de otras formas como las que permiten que EMS Data Generator para PostgreSQL y Datanamic Data Generator MultiDB.

1.4 Metodología de desarrollo de software.

El proceso de desarrollo de software no es una tarea fácil, se debe contar con un proceso bien detallado y para esto se necesita aplicar una metodología que sea capaz de llevar a cabo el control total del producto. Las Metodologías de Desarrollo de Software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de software. Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo software, pero los requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software.

Las metodologías de desarrollo de software se dividen en dos grandes grupos, las pesadas o tradicionales y las ágiles. Las metodologías tradicionales se basan en la idea que el éxito del producto se puede lograr si se tiene todo correctamente documentado, mientras las ágiles defienden la idea de que el proceso de desarrollo del software, se centra en el software como tal y no en la documentación alrededor de este, por lo que le da mayor importancia a la programación que a la documentación, aunque no la obvia por completo, sino que toma en cuenta sólo la documentación necesaria y de forma muy sencilla. (13) Es por esta razón que se decide utilizar para el desarrollo del Sistema Generador de Datos una metodología ágil.

Dentro de las metodologías ágiles se encuentra *Extreme Programming* (XP), la cual es una metodología centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del software, promoviendo así el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un clima agradable y amistoso de trabajo.(14)

La metodología XP brinda retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

XP es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizadas para proyectos de corto plazo. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. La metodología XP, según el autor Gerardo Fernández, cuenta con 4 fases de desarrollo de software:

Planificación

La planificación es la primera etapa de desarrollo de XP. En esta fase se debe hacer primero una recopilación de todos los requerimientos del proyecto, también debe haber una interacción con el usuario, y se debe planificar bien entre los desarrolladores del proyecto que es lo que se quiere para el proyecto para así lograr los objetivos finales.

Diseño

Se sugiere que hay que conseguir diseños simples y sencillos, para procurar hacerlo todo lo menos complicado posible para el usuario o cliente y así conseguir un diseño fácil de entender y de implementar que a la larga costará menos tiempo y esfuerzo para desarrollarlo. En esta fase se logrará crear parte del proyecto, la parte física, es decir, la interfaz mediante la cual debe interactuar el usuario o cliente con el proyecto.

Codificación

En esta fase el cliente es una parte más del equipo de desarrollo; su presencia es indispensable en esta y en las demás fases de X.P. A la hora de codificar una historia de usuario su presencia es aún más necesaria. Los clientes son los que crean las historias de usuario y negocian los tiempos en los que serán implementadas. Antes del desarrollo de cada historia de usuario el cliente debe especificar detalladamente lo que ésta hará y también tendrá que estar presente cuando se realicen los test que verifiquen que la historia implementada cumple la funcionalidad especificada. En esta fase de la codificación los clientes y los desarrolladores del proyecto deben estar en comunicación para que los desarrolladores puedan codificar todo los necesario para el proyecto que se requiere, en esta fase está incluido todo lo de codificación o programación por parte de los desarrolladores del proyecto.

Pruebas.

Uno de los pilares de la metodología X.P es el uso de test para comprobar el funcionamiento de los códigos que se vayan implementando. Para esta fase se implementan los test, que son pruebas que se le hacen al proyecto o a los códigos que se vallan implementando.(15)

Fundamento de la selección.

Se decide utilizar la metodología XP para el desarrollo de la herramienta debido a que es la metodología definida por el departamento PostgreSQL y teniendo en cuenta que:

- Se utiliza para la realización de proyectos a corto plazo: para la realización de la solución planteada se definió un tiempo de desarrollo de 9 meses de trabajo.

- Fomenta el desarrollo de equipos pequeños: XP es una metodología pensada para equipos pequeños. Para solucionar el problema de la investigación planteado el equipo cuenta con solo 2 integrantes para todo el ciclo de desarrollo del software.
- El cliente forma parte del equipo de desarrollo: para el desarrollo del Generador de Datos el cliente estará presente y disponible todo el tiempo para el equipo. Gran parte del éxito de la utilización de esta metodología se debe a que es el cliente quien conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada.

1.5 Herramientas y tecnologías a utilizar.

En el entorno competitivo actual, el empleo de herramientas informáticas aumenta la productividad y competitividad de los proyectos que la utilizan. En este sentido, la incorporación de nuevas tecnologías a los procesos de producción o prestación constituye un desafío pero también una necesidad. En el caso de las empresas de software y servicios informáticos, el uso de estas tecnologías es una condición para el desenvolvimiento de sus actividades diarias.

Para el desarrollo del Generador de Datos ficticios sobre las bases de datos PostgreSQL y MySQL se realizó una investigación acerca de las herramientas y tecnologías para el desarrollo de software existentes y a partir de la misma se hizo una selección de las que más se ajustan a las necesidades del proyecto. A continuación se describen las mismas.

1.5.1 Lenguaje de modelado.

El Lenguaje Unificado de Modelado (LUM o UML), por sus siglas en inglés *Unified Modeling Language*, permite visualizar, especificar, construir y documentar un software. Se aplica en el desarrollo de software para dar soporte a una metodología de desarrollo del mismo, sin especificar que metodología usar. Como lenguaje, es usado para la comunicación y un mejor entendimiento entre el diseñador y el programador. (16)

UML es un lenguaje fácil y descriptivo, ya que permite modelar aplicaciones en cualquier dominio y contiene generación automática de código, ajustando su utilización para todas aquellas personas que no

tengan conocimientos de lenguajes de programación. Este provee al usuario, la posibilidad de que reflexione antes de invertir y gastar grandes cantidades de recursos en proyectos de riesgo.

UML es un lenguaje de modelado de sistemas utilizado para entender, diseñar, examinar, configurar, mantener, y controlar la información sobre los sistemas. Permite modelar, construir y documentar los elementos que forman un sistema de software orientado a objetos. Estandariza a 9 tipos de diagramas.

Se utiliza UML en su versión 8.0 puesto que es un lenguaje consolidado en el mundo y también en la UCI. Se ha convertido en poco tiempo en una notación estándar no sólo para la comunidad de investigadores en ingeniería del software, sino también para la industria.

1.5.2 Herramientas CASE (Computer Aided Software Engineering) para el modelado.

Las Herramientas CASE (Computer Aided Software Engineering; y en su traducción al Español significa Ingeniería de Software Asistida por Computación) son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. Como es sabido, los estados en el ciclo de vida de desarrollo de un software son: Investigación Preliminar, Análisis, Diseño, Implementación e Instalación. Estas se pueden ver como la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales.(17)

Ventajas de las herramientas de Ingeniería de Software Asistida por Computación:

Las herramientas CASE poseen grandes ventajas que permiten aumentar la productividad en el desarrollo de un software:

- Verificar el uso de todos los elementos en el sistema diseñado.
- Automatizar el dibujo de diagramas.
- Ayudar en la documentación del sistema.
- Ayudar en la creación de relaciones en la Base de Datos. (17)

Existen diferentes tipos de herramientas CASE, dentro de las cuales se encuentra Visual Paradigm, la cual es una herramienta que es aplicada a lo largo de todo el ciclo de vida de un software. En estas son incluidas actividades como la gestión de proyectos y la estimación. También ofrece un conjunto completo de herramientas de los equipos de desarrollo de software necesario para la captura de requisitos, software de planificación, la planificación de controles, el modelado de clases y de datos, entre otras.

Características de Visual Paradigm.

- Visual Paradigm For UML es una Herramienta Case que soporta las últimas versiones del UML y la Notación y Modelado de Procesos de Negocios.
- En adición al soporte de Modelado UML esta herramienta provee el modelado de procesos de negocios, además de un generador de mapeo de objetos-relacionales para los lenguajes de programación Java, .NET y PHP.
- Para desarrolladores independientes existe una versión llamada Community Edition en la que se caracteriza por ser de uso No Comercial.
- Se integra con las siguientes herramientas Java: Eclipse/IBM WebSphere, JBuilder, NetBeans IDE, Oracle JDeveloper, BEA Weblogic, entre otras.
- Está disponible en varias ediciones, cada una destinada a unas necesidades: Enterprise, Professional, Community, Standard, Modeler y Personal. (18)

Ventajas de Visual Paradigm

- Navegación intuitiva entre código y el modelo.
- Poderoso generador de documentación y reportes UML PDF/HTML/MS Word.
- Demanda en tiempo real, modelo incremental de viaje redondo y sincronización de código fuente.
- Superior entorno de modelado visual.
- Soporte completo de notaciones UML.
- Diagramas de diseño automático sofisticado.
- Proporciona soporte a varios lenguajes en generación de código e ingeniería inversa a través de plataformas java. (18)

Fundamento de la selección.

Se decide utilizar el Visual Paradigm 5.0 ya que ha sido utilizada por el equipo de desarrollo en el diseño de otras aplicaciones por lo que se tiene dominio sobre esta herramienta. Además es muy útil a la hora de elaborar el modelo de dominio del problema y el diagrama de clases, que aunque ambos artefactos no son generados por XP son de utilidad para la comprensión de la herramienta a desarrollar. También esta herramienta brinda la posibilidad de elaborar prototipos de interfaz para la representación de las Historias de Usuario.

1.5.3 Lenguaje de programación.

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones respectivamente, los cuales pueden ser utilizados para controlar el comportamiento de una máquina, especialmente una computadora. (19)

Un lenguaje de programación es un conjunto de sintaxis y reglas semánticas diseñadas para describir el conjunto de acciones consecutivas que un equipo debe ejecutar, es decir es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo. Permite especificar de una manera más precisa sobre qué datos debe operar una computadora, sobre cómo estos datos deben ser almacenados o transmitidos y qué acciones debe tomar bajo una determinada circunstancia.

Actualmente en DATEC el lenguaje mayormente utilizado para el desarrollo de productos software es el C++ utilizando el Entorno de Desarrollo Integrado (IDE) Qt, pero debido a la complejidad de este y al corto tiempo con el que se cuenta para el final de este proyecto se decide evaluar junto a este a otro lenguaje, en este caso Java, debido a ciertas cualidades próximamente enunciadas que garantizan una mayor comodidad y eficiencia en la implementación de la herramienta.

Tabla 2. Comparación entre Java y C++

Características	Java	C++
Sencillez	Si	No
Verificaciones	Si	Menor
Seguridad	Si	Menor

Representación	Alta	Alta
Orientado a Objetos	Si	Si
Portabilidad	Si	Menor

Sencillez

Java tiene una sencillez que no posee C++. Esto es debido a que una de las razones de la creación de Java es la de obtener un lenguaje parecido a C++ pero reduciendo los errores más comunes de la programación, ya que este reduce un 50% los errores que se comenten en C++ entre los que destacan:

- Eliminación de la aritmética de punteros y de las referencias.
- Desaparecen los registros (*struct*), heredados del paradigma estructurado.
- Ya no es necesario liberar memoria (*free o delete*).

Verificaciones

Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución, así se detecten errores lo antes posible, normalmente en el ciclo de desarrollo. Algunas son:

- Verificación del código de byte.
- Gestión de excepciones y errores.
- Comprobación de punteros y de límites de vectores.

C++ no realiza ninguna de estas verificaciones dificultando su uso para desarrollar.

Seguridad

En Java no se permite los accesos ilegales a memoria. En C++ sí se permite. Esto es algo muy importante puesto que este tipo de problema puede ocasionar la propagación de virus y otras clases de programas dañinos por la red. El código Java realiza pruebas antes de ejecutarse en una máquina. El formato del código de byte se comprueba y un probador de teoremas detecta fragmentos de código ilegal, falseo de punteros, violación de derechos de acceso sobre objetos o cambio del tipo o clase de un objeto. Además el código no produce desbordamiento de la pila.

Representación

La obtención de programas con interfaces cómodas e intuitivas, es uno de los objetivos perseguidos mediante el uso de Java. Esto también se permite en C++, aunque con métodos más costosos, y sin interfaces portables como las creadas por Java.

Orientación a Objetos

Aunque C++ es también, como Java, un lenguaje orientado a objetos, la diferencia fundamental es que Java lo es desde su concepción, mientras que C++ se diseñó como un lenguaje compatible con C el cual no cumple con esta característica. Los lenguajes tradicionales no orientados a objetos, como C, trabajan de forma secuencial y basando su funcionamiento en un concepto de procedimiento o función con un alto grado de interrelación y dependencias.

Los lenguajes orientados a objetos se basan en el concepto de objeto, como un elemento de programación, autocontenido y reutilizable. Este posee un conjunto de variables o datos y un conjunto de métodos para manejar los datos. (18)

Ser Orientado a Objetos (OO), permite ciertas comodidades:

- Cada objeto puede ser modificado y mantenido por separado.
- "Ocultamiento de la información". Se pueden mantener en un objeto métodos y variables que no son accesibles desde fuera de él.
- Permite reutilizar porciones de programa ya escritas.

Portabilidad

Los programas Java se compilan a un ``código máquina'' llamado, byte-code, para una Unidad de procesamiento central (CPU) que no existe físicamente: Máquina Virtual de Java (JVM por sus siglas en inglés). Por la red se envía el mismo a todos los clientes que lo solicitan, en cada cliente hay un intérprete que lo transforma al verdadero código máquina que necesita para ejecutar la aplicación. El esquema es viable porque el intérprete que transforma desde byte-code a código máquina es un programa sencillo y fácil de escribir en todas las plataformas. C++ no es portable ya que se limita a ejecutarse en la misma

plataforma que fue compilado a diferencia del byte-code de Java ya que éste es portable a través de diferentes sistemas operativos y procesadores. (20)

Fundamento de la selección.

No es fácil determinar si un lenguaje es más eficiente que otro, hay muchos factores a tener en cuenta. Todos los lenguajes de programación tienen sus ventajas y sus desventajas, y si bien algunos son buenos en cuestiones de eficiencia, no son tan buenos en términos de diseño.

Aunque un programa desarrollado en Java es un programa interpretado, y en principio no es tan rápido como un programa equivalente compilado, las prestaciones de este leguaje son muchísimo mejores que las de cualquier lenguaje interpretado y su funcionamiento en cualquier plataforma independientemente del sistema operativo es una cualidad relevante, de ahí su gran uso internacional. Los programas desarrollados con código Java por lo general suelen ser de código abierto, en este caso particular, se lograría una herramienta que cumpla con esta característica y capaz de ejecutarse tanto en Linux como en Windows.

1.5.4 Entorno de desarrollo.

Un entorno de desarrollo integrado, por sus siglas en inglés *Integrated Development Enviroment* (IDE) es un programa que está compuesto por un conjunto de herramientas hechas para los programadores. Este puede ser usado por varios lenguajes de programación o por uno en específico.(21)

Los IDE son editores de código o constructores de interfaz gráfica que pueden ser aplicaciones independientes o ser parte de aplicaciones que ya existen. Proveen un ambiente de trabajo muy amigable para la mayoría de los lenguajes de programación.(22)

Para el desarrollo de la herramienta, luego de determinar el leguaje a utilizar, es necesario evaluar el Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés), para lograr el objetivo. Los IDE más utilizados en la universidad para el desarrollo de aplicaciones basadas en Java, son el Eclipse y el NetBeans, este último fue seleccionado por los autores por razones explicadas a continuación. Se utiliza además, pero con mucha menos regularidad el Monodevelop, pero sus restricciones y defectos en cuanto

al uso de bibliotecas lo hacen prácticamente omisible para este trabajo. Es importante apreciar que todas estas herramientas se acogen a la tendencia de emigrar a software libre que se lleva a cabo en la Universidad de las Ciencias Informáticas.

La Plataforma NetBeans

La Plataforma NetBeans es una base modular y extensible usada como una estructura de integración para crear aplicaciones de escritorio grandes, con soporte y extensiones adicionales integrables en la plataforma, herramientas y soluciones.
Ofrece servicios comunes a las aplicaciones de escritorio, para que el desarrollador se centre en la lógica de su aplicación, como son:

- Administración de las interfaces de usuario (ej. menús y barras de herramientas).
- Administración de las configuraciones del usuario.
- Administración del almacenamiento (guardando y cargando cualquier tipo de dato).
- Administración de ventanas.
- Framework basado en asistentes.

NetBeans IDE

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso. NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. (23)

NetBeans es una herramienta para programadores que les permite escribir, compilar, depurar y ejecutar sus programas. Todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.(23)

Ofrece servicios comunes a las aplicaciones de escritorio, para que el desarrollador se centre en la lógica de su aplicación, como son:

- Administración de las interfaces de usuario (ej. menús y barras de herramientas).
- Administración de las configuraciones del usuario.
- Administración del almacenamiento (guardando y cargando cualquier tipo de dato).
- Administración de ventanas.
- Framework basado en asistentes.

Fundamento de la selección.

Para el desarrollo de la herramienta se tuvo en cuenta el NetBeans IDE en su versión 7.3.1 dado que fue evaluado por un colectivo internacional de usuarios arrojando resultados gratificantes. Las opiniones más comunes fueron de un grado de usabilidad de 9.4, estabilidad de 10, funciones de 10, instalación de 7.8 y apariencia de 7.7, todo en una escala del 1al 10.(24)

Conclusiones parciales del capítulo

Luego del desarrollo del presente capítulo se puede arribar a las conclusiones siguientes:

- Se analizaron los principales conceptos asociados al tema tales como base de datos, gestor de base de datos y generación datos ficticios.
- Se hizo un estudio de algunas de las herramientas existentes en el mundo para la generación de datos ficticios, EMS Data Generator para PostgreSQL, EMS Data Generator para MySQL Postgen y Datanamic Data Generator MultiDB.
- Se identificaron las principales herramientas y tecnologías a usar para el desarrollo del Generador de Datos, donde se eligió UML 2.0 como lenguaje de modelado, Visual Paradigm 8.0 como herramienta CASE, Java como lenguaje de programación y NetBeans7.3.1 como entorno de desarrollo.

CAPÍTULO 2: DISEÑO Y CARATERÍSTICAS DEL SISTEMA PROPUESTO

Introducción

El presente capítulo es uno de los más importantes pues en el mismo se describe detalladamente el sistema propuesto como solución al problema de investigación identificado. Además se generan los artefactos que propone la metodología XP para las etapas de planificación y diseño, y por último se definen los patrones de diseño y arquitectura que se utilizarán para el desarrollo de la solución propuesta.

2.1 Descripción de la solución propuesta.

Con el objetivo de dar solución al problema de investigación planteado en el capítulo anterior se propone desarrollar un sistema capaz de generar datos ficticios a las bases de datos de PostgreSQL y MySQL. Dicho sistema brindará la opción de seleccionar las bases de datos y las tablas de estas que se desean poblar. Además posibilitará que el usuario pueda especificar la cantidad de datos que serán generados en cada tabla y podrá también decidir de qué forma desea generar los datos, para esto cuenta con varias opciones: generar datos de forma aleatoria, de forma secuencial, a través de un archivo, a través de una consulta y a través de una tabla.

El Generador de Datos para las bases de datos PostgreSQL y MySQL que se desarrollará será funcional sólo para los siguientes tipos de datos, teniendo en cuenta que estos son los datos con los que se trabaja en el centro DATEC.

Datos para MySQL

- Datos numéricos
  - Int o integer
  - Float
  - Double
  - Decimal
  - Bigint

- Datos alfanuméricos
  Char
  Varchar
  Binary

- Datos de fecha y hora
  Date
  Time
  Datetime

Datos para PostgreSQL

- Datos numéricos
  Int o integer
  Float
  Real, double
  Boolean
  Bigint

- Datos alfanuméricos
  Character
  Character varying
  Char
  Text

- Datos de fecha y hora
  Date
  Time

2.2 Modelo de Dominio

Se denomina modelo del dominio a la representación visual de los conceptos u objetos del mundo real en un dominio de interés. Este modelo agrupa los conceptos de un dominio. Es el mecanismo fundamental para comprender el dominio del problema y para establecer conceptos comunes. (25)

Aunque la metodología XP no genera modelo de dominio, a continuación se presenta este artefacto con el objetivo de comprender el proceso que se realiza a la hora de generar datos. Dicho modelo se realizó teniendo en cuenta la definición de modelo de dominio que da el autor Craig Larman en su libro UML y Patrones, donde expone que un modelo de dominio o modelo conceptual es una representación de conceptos en un dominio del problema, no de las entidades del software. (26)

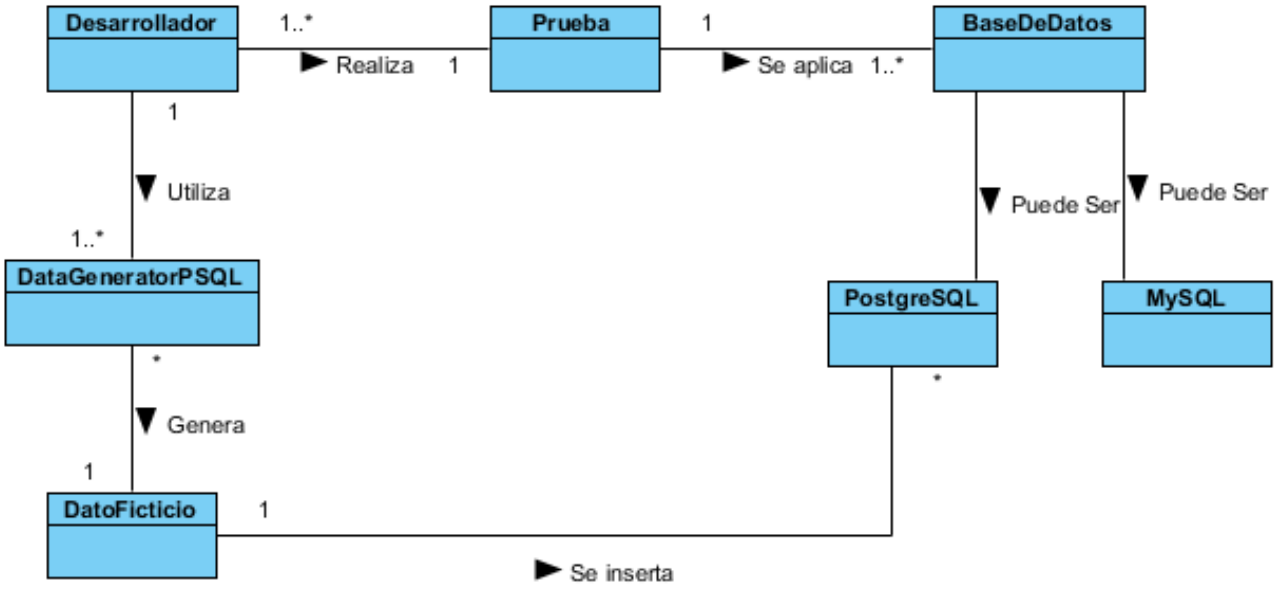


Fig. 1 Modelo de Dominio.

Desarrollador: Persona que realiza las pruebas a la base de datos y utiliza el DataGneratorPostgreSQL: Herramienta existente tomada como punto de partida para modelar los conceptos principales y más comunes de los editores de consultas.

Prueba: Conjunto de acciones que se le aplican a las bases de datos para comprobar su funcionamiento.

Base de Datos: La base de Datos a la cual le serán aplicadas las pruebas.

PostgreSQL: Base datos de PostgreSQL sobre la cual trabajará el usuario.

MySQL: Base datos de MySQL sobre la cual trabajará el usuario.

DatoFicticio: Volumen de datos generado por DataGneratorPostgreSQL que serán insertados en la base de datos PostgreSQL.

2.3 Diseño de la aplicación.

Dentro del proceso de desarrollo de software se encuentra la fase de diseño, la cual es fundamental para lograr una buena producción de software. El objetivo de esta etapa es garantizar la satisfacción del usuario logrando cubrir todas sus necesidades, y ayudar a comprender mejor el sistema que se desarrolla.

2.3.1 Historias de Usuario (HU)

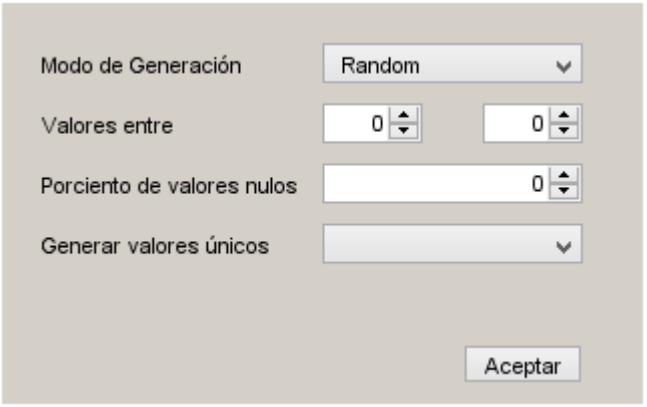
Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento las historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas. (27)

Las historias de usuario tienen el mismo propósito que los casos de uso. Estas son escritas por los propios clientes, tal y como ven ellos las necesidades del sistema. Una HU es similar al empleo de un escenario, con la excepción de que la HU no se limita a la descripción de la interfaz de usuario.

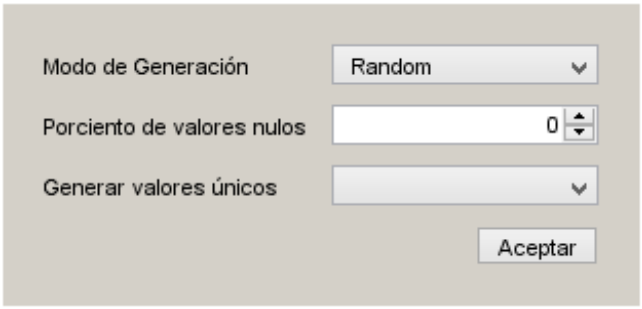
Las HU conducirán el proceso de creación de los test de aceptación (empleados para verificar que las historias de usuario han sido implementadas correctamente). Existen diferencias entre estas y la tradicional especificación de requisitos. La principal diferencia es el nivel de detalle. Las historias de usuario solamente proporcionaran los detalles sobre la estimación del riesgo y cuánto tiempo conllevará la implementación de dicha historia de usuario.

Para el desarrollo del sistema propuesto anteriormente fueron identificadas y documentadas 20 HU. A continuación se muestran un ejemplo de las HU identificadas.

Tabla 3. HU Generar datos ficticios de forma aleatoria para PostgreSQL.

<table>
<tr><th colspan="3">Historia de Usuario</th></tr>
<tr><td>Número: 3</td><td colspan="2">Nombre de la Historia de Usuario: Generar datos ficticios de forma aleatoria para PostgreSQL.</td></tr>
<tr><td colspan="3">Cantidad de modificaciones a la Historia de Usuario: Ninguna</td></tr>
<tr><td colspan="2">Usuario: Yordaika Verdecia Agüero</td><td>Iteración asignada: 1</td></tr>
<tr><td colspan="2">Prioridad en negocio: Muy Alta</td><td>Puntos estimados: 1.0</td></tr>
<tr><td colspan="2">Riesgo en desarrollo: Alta</td><td>Puntos reales: 1.0</td></tr>
<tr><td colspan="3">Descripción: Permite generar datos ficticios de forma aleatoria para la base de datos. Esta forma de generación permite generar para los siguientes tipos de datos: integer, double, text, char, date y float. Para este tipo de generación el usuario podrá trabajar con dos interfaces: una común para los tipos de datos integer, double, date y float, y otra para los tipos de datos char y text.</td></tr>
<tr><td colspan="3">Observaciones: Antes de realizar esta operación el usuario debe asegurarse de haber seleccionado las tablas que desea poblar y especificado la cantidad de tuplas que desea generar.</td></tr>
<tr><td colspan="3">Prototipo de interfaz para los tipos de datos integer, double, date y float
 
</td></tr>
</table>

Prototipo de interfaz para los tipos de datos char y text



Para ver el resto de las HU remitirse a los documentos complementarios, Plantilla de Historias de Usuario del Expediente de proyecto.

2.3.2 Lista de Reserva de Producto

La lista de Reserva del Producto es una tabla que contiene los requisitos funcionales que debe cumplir la aplicación, ordenados por la prioridad de implementación, así como los requisitos no funcionales del sistema a desarrollar con una breve descripción de cada uno de ellos. Además en esta tabla se refleja la estimación de cada uno de los requisitos y su implementación por semanas, definiendo además el rol que realizó la estimación.

La siguiente lista de reserva del producto recoge los requisitos correspondientes al Generador de Datos ficticios sobre las bases de datos PostgreSQL y MySQL a implementar.

Tabla 4. Lista de Reserva del Producto.

Ítem *	Descripción	Estimación	Estimado por
Prioridad: Muy Alta			
1	Establecer conexión con la Base de Datos PostgreSQL.	1	Analista
2	Establecer conexión con la Base de Datos MySQL.	1	Analista
3	Generar datos ficticios de forma aleatoria para PostgreSQL.	1	Analista
4	Generar datos ficticios de forma aleatoria para MySQL.	1	Analista

5	Generar datos ficticios de forma secuencial para PostgreSQL.	1	Analista
6	Generar datos ficticios de forma secuencial para MySQL.	1	Analista
7	Generar datos ficticios a través de un archivo para PostgreSQL.	1	Analista
8	Generar datos ficticios a través de un archivo para MySQL.	1	Analista
9	Generar datos ficticios a través de una consulta para PostgreSQL.	1	Analista
10	Generar datos ficticios a través de una consulta para MySQL.	1	Analista
11	Genera datos ficticios a través de una tabla para PostgreSQL.	1	Analista
12	Genera datos ficticios a través de una tabla para MySQL.	1	Analista
13	Insertar datos en la Base de Datos.	1	Analista
Prioridad: Alta			
14	Seleccionar tablas de la Base de Datos.	1	Analista
15	Mostrar tablas seleccionadas.	1	Analista
16	Mostrar datos.	1	Analista
Prioridad: Media			
17	Generar script.	1	Analista
18	Guardar Proyecto.	1	Analista
19	Abrir proyecto.	0.5	Analista
20	Generar ayuda.	0.5	Analista
Prioridad: Baja			
1	**Usabilidad:** Los enlaces y botones serán fáciles de asociar con las operaciones que realizan. Además la herramienta contará con una ayuda para el usuario donde se indica cómo realizar cada operación.		

2	**Eficiencia:** El tiempo que demorará el sistema en brindar una respuesta a las solicitudes del usuario será no mayor de 10 segundos, al igual que la velocidad de procesamiento de la información.	
4	**Portabilidad:** El sistema debe ser multiplataforma, debe ejecutarse de manera óptima sin importar el sistema operativo que utilice el usuario.	
5	**Soporte:** El sistema será de fácil instalación, configuración y puesta en marcha, contando de esta manera con archivo instalador donde se describirán los pasos a seguir por el usuario.	
6	**Software:** Sistema Operativo: Multiplataforma. Librerías Java. Servidor de bases de datos: SGBD PostgreSQL a partir de su versión 9.1.0 y SGBD MySQL.	

2.3.3 Plan de iteraciones.

Un plan de iteraciones está compuesto por iteraciones que no superan a las tres semanas. Este posee una arquitectura del sistema que puede ser establecida en la primera iteración y utilizada durante la trayectoria del proyecto. Para que se efectúe la creación de esta arquitectura se deben escoger las historias correctas, pero como el cliente es el que decide qué o cuáles historias se implementarán en cada iteración, muchas veces la implementación de esta arquitectura no es posible. El sistema estará listo para su utilización al completarse la última iteración.

Las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido. Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración. Cada historia de usuario se traduce en tareas específicas de programación. De esta misma forma, para cada historia de usuario se establecen las pruebas de aceptación. Estas pruebas se realizan al final del ciclo en el que se desarrollan, pero también al final de cada uno de los ciclos siguientes, para verificar que subsiguientes iteraciones no han afectado a las anteriores. Las pruebas de aceptación que hayan fallado en el ciclo anterior son analizadas para evaluar su corrección, así como para prever que no vuelvan a ocurrir. (28)

En el transcurso de la elaboración del plan de iteraciones se deben tomar en cuenta elementos, tales como:

- Velocidad del proyecto.
- Pruebas de aceptación no superadas de la iteración precedente.
- Tareas no terminadas de la iteración precedente.
- Historias de usuarios no abordadas.

Una vez definidas las HU, es necesario crear un plan de iteraciones con el objetivo de definir cuáles son las tareas con más prioridad y establecer los tiempos de implementación. Para el desarrollo de la solución se definieron 3 iteraciones y para realizar la selección del orden de implementación de las historias de usuario se tuvo en cuenta la prioridad que estas representan para el negocio y la relación de sus funcionalidades.

A continuación se muestra el plan de iteraciones correspondiente al desarrollo del Generador de Datos ficticios sobre las bases de datos PostgreSQL y MySQL.

Tabla 5. Plan de iteraciones.

Release	Descripción de la iteración	Orden de la HU a implementar	Duración total
Iteración 1	En esta primera iteración se implementarán las HU 1, 3, 5, 7, 9, 11 y 13. Estas son HU de prioridad muy alta por tanto son las que mayor incidencia tienen sobre el funcionamiento del Generador de Datos.	1,3,5,7,9,11,13	6 semanas
Iteración 2	En esta iteración serán implementadas las HU 2, 4, 6, 8, 10, 12, 14, 15 y 16 las cuales son de prioridad alta, es decir que aunque no son tan importantes, sí juegan un papel importante en el buen funcionamiento del	2,4,6,8,10,12,14,15,16	7.5 semanas

	sistema propuesto.		
Iteración 3	En esta tercera y última iteración se implementarán las HU de menor prioridad, en este caso las de prioridad baja son las HU 17, 18, 19 y 20.	17,18,19,20	2 semanas

2.3.4 Tarjetas Clases-Responsabilidades-Colaboración

Las tarjetas CRC (Clases-Responsabilidades-Colaboración) son una técnica de modelado orientado a objetos que permite identificar las clases y sus responsabilidades, además de servir de herramienta de ayuda al refinamiento de clases.

Las tarjetas CRC están divididas en tres partes:

1. Clase: Una clase es cualquier persona, cosa, evento, concepto, pantalla o reporte.
2. Responsabilidades: las responsabilidades de una clase son las cosas que la clase conoce y realiza, sus atributos y métodos.
3. Colaboradores: Los colaboradores de una clase son las demás clases con las que trabaja en conjunto.

La característica más sobresaliente de las tarjetas CRC es su simpleza y adaptabilidad. Su principal beneficio es que alientan la disertación animada entre los desarrolladores. Son especialmente eficaces cuando se está en medio de un caso de uso para ver cómo se van a implementar las clases. Los desarrolladores escogen tarjetas a medida que cada clase colabora con la HU. Conforme se van formando ideas sobre las responsabilidades, se pueden escribir en las tarjetas. Es importante pensar en las responsabilidades, ya que evita pensar en las clases como simples depositarias de datos, y ayuda a que el equipo se centre en comprender el comportamiento de alto nivel de cada clase.(29)

El formato físico de las tarjetas CRC facilita la interacción entre clientes y equipo de desarrollo, en sesiones en las que se aplican técnicas de grupos como tormenta de ideas o juego de roles, y se ejecutan escenarios a partir de especificación de requisitos o historias de usuarios. De esta forma, van surgiendo las entidades del sistema junto con sus responsabilidades y colaboraciones. Luego en un estado de

diseño avanzado o ya en la implementación del sistema, las tarjetas CRC se convierten en clases con métodos, atributos, relaciones de herencia, composición o dependencia. A continuación se muestra una de las tarjetas CRC que se elaboraron para la solución propuesta.

Tabla 6. Tarjeta CRC para la clase Control.

Tarjeta CRC	
Clase: Control	
Responsabilidades	Colaboraciones
Se encarga de controlar el sistema. Controla todo el proceso de generación de datos ficticios y la inserción de los mismos en la base de datos.	*Insertar* *VPrincipal* *Tabla*

Para ver el resto de las tarjetas CRC remitirse a los documentos complementarios, Plantilla Modelo de diseño del Expediente de proyecto.

2.3.5 Diagrama de Clases

Para el diseño de aplicaciones informáticas la metodología XP no requiere la presentación del sistema mediante diagramas de clases utilizando notación UML. No obstante el uso de estos diagramas puede aplicarse siempre y cuando influyan en el mejoramiento de la comunicación, no sea un peso su mantenimiento, no sean extensos y se enfoquen en la información importante.

Un diagrama de clases es un diagrama estático que se utiliza para modelar la vista estructural de un sistema. El mismo describe gráficamente las especificaciones de las clases (atributos y métodos) además de visualizar las relaciones entre estas. A continuación se explican brevemente las clases que conforman el diagrama de clases correspondiente al sistema propuesto como solución.

VPrincipal: Contiene el componente visual principal que se le mostrará al usuario. Es la encargada de mostrar las tablas sobre las cuales está trabajando el usuario.
Control: Es la clase controladora del sistema. Se encarga de manejar todas las funcionalidades del sistema.

ConexionDB: Es la clase encargada de realizar la conexión con la base de datos sobre la cual desea trabajar el usuario.

Insertar: Se encarga de insertar en la base de datos, sobre la cual trabaja el usuario, los datos generados.

Tabla: Esta clase contiene toda la información de la tabla seleccionada por el usuario.

Columna: Contiene toda la información sobre un atributo de la tabla seleccionada.

Las clases **Boolean**, **Real**, **Binary**, **Integer**, **Date**, **Double**, **Varchar**, **Date**, **Float**, **Time**, **Datetime**, **Char**, **Decimal** y **Bigint** son encargadas de devolver una lista de datos por cada tipo de generación de datos que se utiliza.

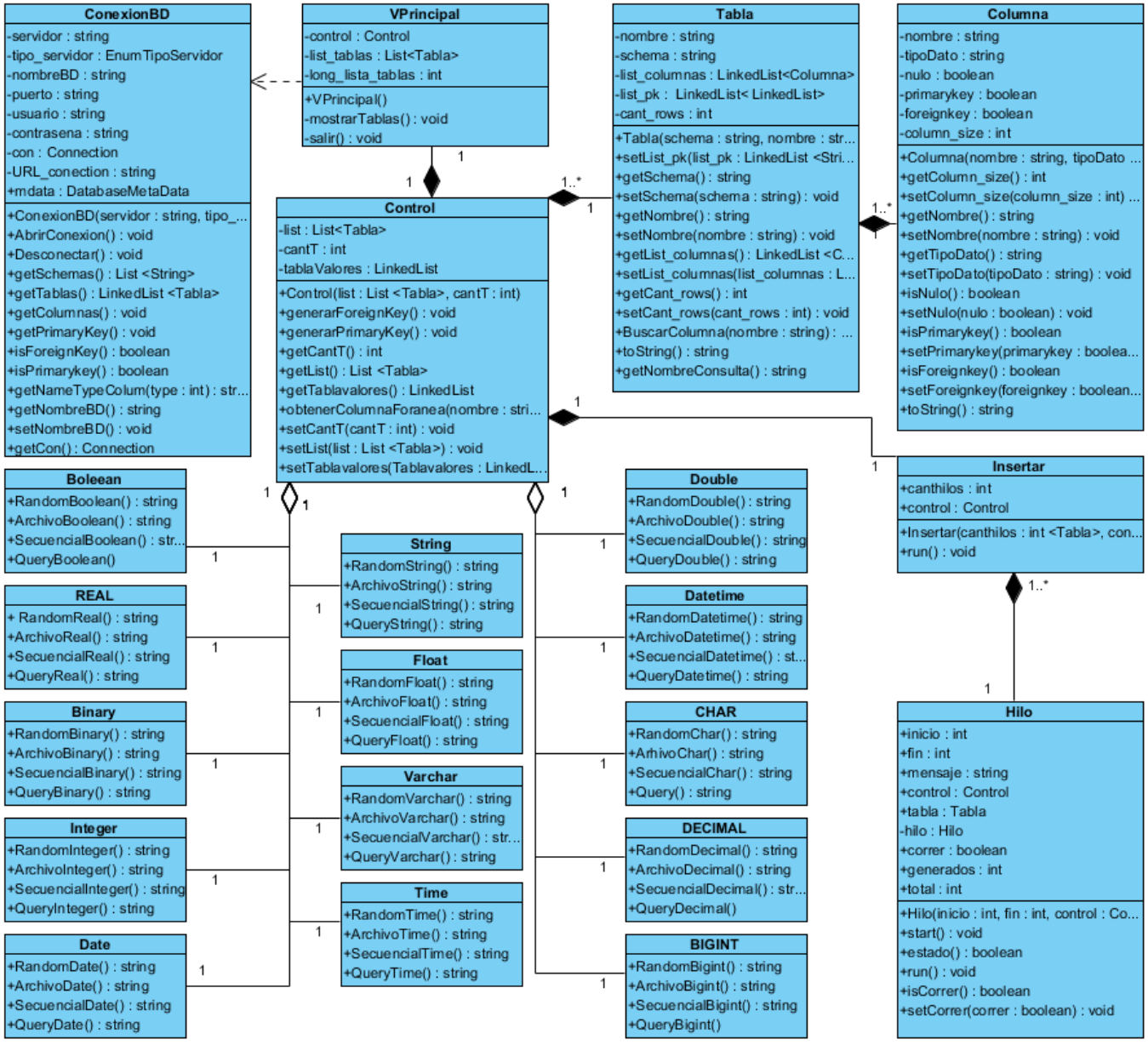


Fig. 2 Diagrama de Clases.

2.4 Arquitectura base de la aplicación

2.4.1 Patrones de arquitectura

Un patrón de arquitectura de software describe un problema particular y recurrente del diseño que surge en un contexto específico, y presenta un esquema genérico y probado de su solución. Para lograr una arquitectura más robusta y consistente de la aplicación se utilizó el patrón Modelo-Vista-Controlador (MVC). Este consiste en separar la lógica de negocio de la interfaz de usuario, dividiendo la aplicación en tres componentes: el modelo, la vista y el controlador.

El **Modelo** contiene los datos y la funcionalidad esencial. Encapsula el núcleo funcional y los datos involucrados. Por su parte la **Vista** despliega la información al usuario y presenta esta información por pantalla y el **Controlador** define la forma en la que debe reaccionar la interfaz del usuario, frente a la entrada de datos del usuario.

Este patrón permite que los sistemas sean flexibles y adaptables. Contiene múltiples vistas para el mismo modelo, vistas sincronizadas y vistas intercambiables. Con la utilización de este patrón es mucho más sencillo agregar múltiples representaciones de datos o información; facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento de las otras capas; facilita además el mantenimiento en caso de errores y ofrece maneras sencillas para probar el correcto funcionamiento del sistema.(30)

Emplear esta arquitectura en la implementación del Generador de Datos ficticios para las bases de datos PostgreSQL y MySQL tiene como ventajas que ofrece a los desarrolladores una mayor flexibilidad para personalizar la presentación de las vistas a partir de la separación de la funcionalidad introducida por esta arquitectura, además proporciona una interfaz de modelo estándar, que permite utilizar una amplia gama de fuentes de datos para ser utilizada con las vistas de elementos existentes. A continuación se muestra como se evidencia el patrón Modelo-Vista-Controlador en el desarrollo de la aplicación.

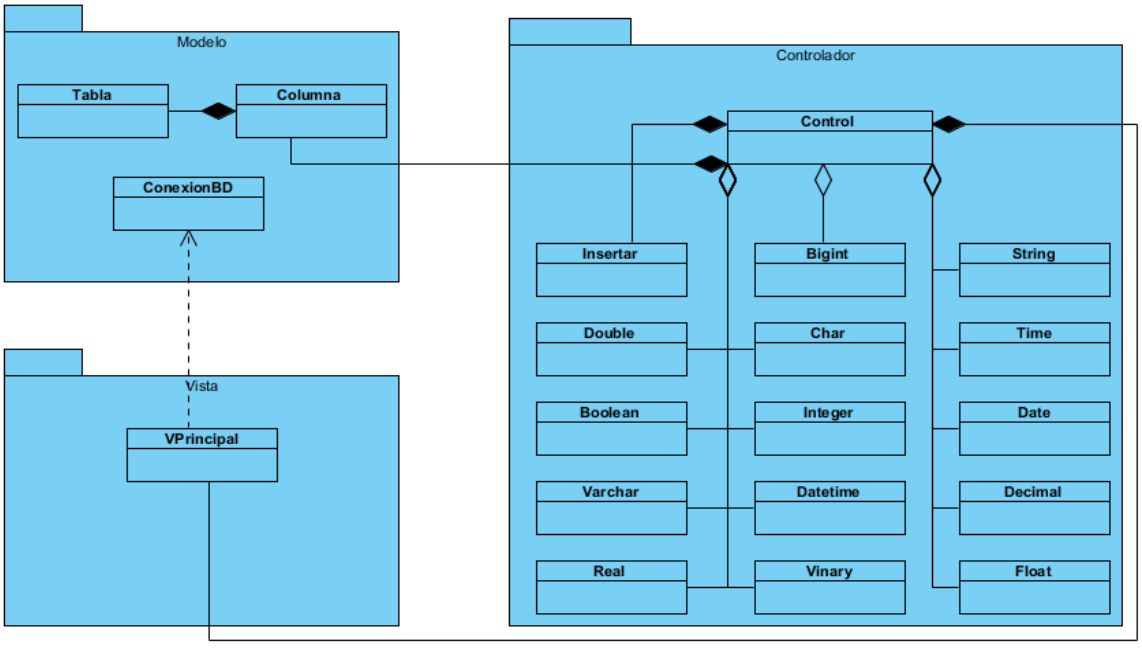


Fig. 3 Patrón de Arquitectura: Modelo-Vista-Controlador.

La clase del componente **Vista**, maneja eventos, captura y visualiza los datos de interés para el usuario. Estos datos son manipulados, validados y controlados por la lógica del negocio, es decir, por las clases agrupadas en el componente **Controlador*;*** tomando la información necesaria de las clases persistentes y de las de acceso a datos, agrupadas en el componente **Modelo*.***

2.4.2 Patrones de diseño

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema general de diseño en un contexto particular.

Los patrones de diseño son un esqueleto de las soluciones a problemas comunes en el desarrollo de software, en otras palabras son respuestas a problemas presentes en el diseño de un producto informático. Deben poseer dos características fundamentales: que su efectividad debe haberse comprobado al utilizarse para resolver problemas similares en ocasiones anteriores y debe adaptarse a disímiles problemas de diseño en distintos escenarios, es decir, que el patrón sea reutilizable. (31)

Los patrones de diseño son los encargados de solucionar un problema en el diseño del algoritmo. Un patrón es una descripción del problema y la esencia de su solución, que se puede reutilizar en casos distintos.

Para resolver los problemas en el diseño del Generador de Datos ficticios se utilizaron los patrones de asignación de responsabilidades o GRASP (*General Responsibility Assignment Software Patterns*, por sus siglas en inglés) y los patrones GoF (*Gang of Four*, "Pandilla de los Cuatro").

Patrones GRASP

Los patrones GRASP (Patrones de Software para la asignación General de Responsabilidad) constituyen un apoyo para la enseñanza, pues ayudan a entender el diseño de objetos. Una de las cosas más complicadas en la programación orientada a objetos consiste en elegir las clases adecuadas y decidir cómo estas clases deben interactuar. Incluso cuando se utilizan metodologías ágiles como XP y se centra el proceso en el desarrollo continuo, es inevitable elegir cuidadosamente las responsabilidades de cada clase en la primera codificación y, fundamentalmente, en la refactorización del programa.(32)

De los diferentes patrones que ofrece GRASP se ha tenido en cuenta para la modelación del Generador de Datos los siguientes:

Experto: Asigna responsabilidades a las clases expertas en información, es decir, las clases que tienen la información necesaria para cumplir con la responsabilidad. (33)

Este patrón se pone de manifiesto con la clase **Tabla** la cual es la única que contiene toda la información de una tabla, de esta forma se convierte en la clase experta y con las cualidades para responder ante cualquier evento relacionado a las tablas.

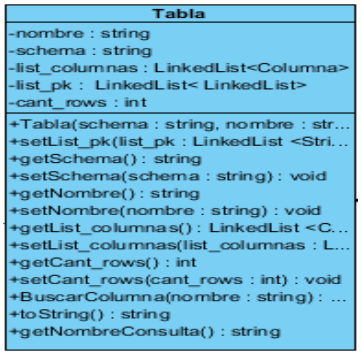


Fig. 4 Fragmento de diagrama de clases donde se evidencia el patrón Experto.

Creador: El patrón creador ayuda a identificar quién debe ser el responsable de la creación de nuevos objetos o clases. La nueva instancia deberá ser creada por la clase que tiene la información necesaria para realizar la creación del objeto, o usa directamente las instancias creadas del objeto. (34)

La clase **Control** es un ejemplo de este patrón ya que posee las cualidades y responsabilidades para crear instancias de objetos de tipo Tabla.

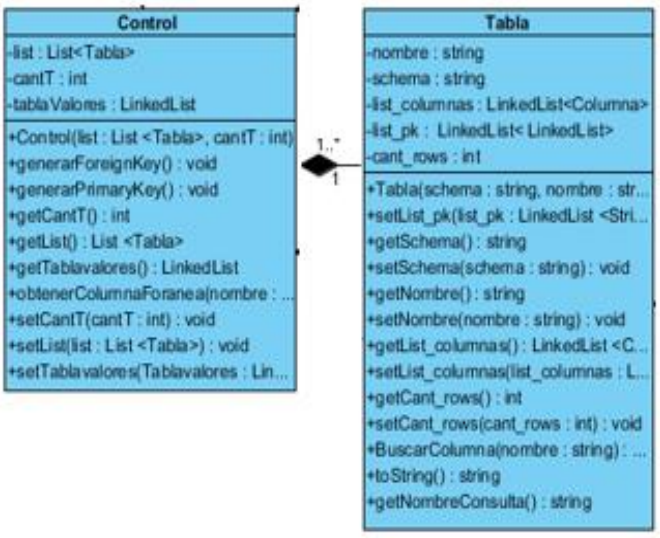


Fig. 5 Fragmento de diagrama de clases donde se evidencia el patrón Creador.

Alta Cohesión: Asigna una responsabilidad de modo que la cohesión siga siendo alta. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. (35)

Este patrón se pone de manifiesto en la clase Insertar PostgreSQL, que es la única encargada de insertar datos en la BD PostgreSQL.

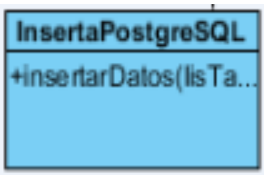


Fig. 6 Fragmento del diagrama de clases donde se evidencia el Alta Cohesión.

Controlador: Asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc.). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. (36)

Un ejemplo de este patrón es la clase Control, la cual se encarga de gestionar las principales operaciones del sistema.

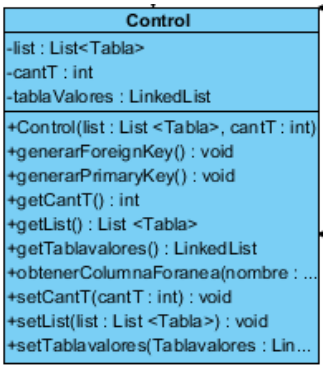


Fig. 7 Fragmento de diagrama de clases donde se evidencia el patrón Controlador.

Patrones GOF

Los patrones GOF recopilan una serie de patrones de diseños, agrupados en tres categorías: de creación, de estructura y de comportamiento. De los diferentes patrones que ofrece GOF se ha tenido en cuenta para la modelación del Generador de Datos los siguientes:

Estructurales: Describen la manera en que se pueden relacionar distintos tipos de objetos, para trabajar unos con otros y formar estructuras de mayor tamaño.

- **Facade (Fachada):** Proporciona una interfaz unificada y de alto nivel para un conjunto de interfaces de un subsistema lográndose que sea más fácil de usar.

El uso de este patrón se pone de manifiesto en la clase VPrincipal la cual es la interfaz que unifica a todas las demás interfaces visuales.

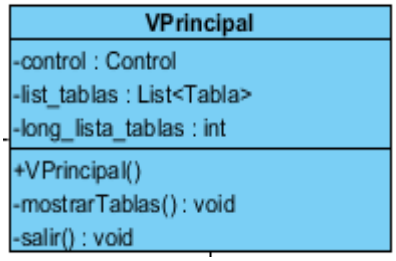


Fig. 8 Fragmento de diagrama de clases donde se evidencia el patrón Facade.

Comportamiento: Contribuyen a definir la comunicación e iteración entre los objetos de un sistema.

- **Observer (Observador):** Define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos.

El uso de este patrón se pone de manifiesto en la clase Tabla y Columna donde al ocurrir un cambio en un objeto de la clase Columna este se actualiza en la clase Tabla.

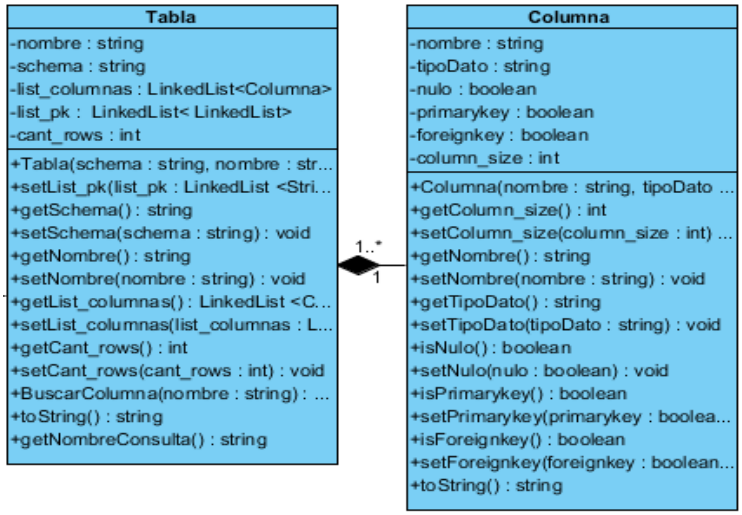


Fig. 9 Fragmento de diagrama de clases donde se evidencia el patrón Observer.

Conclusiones parciales del capítulo

Luego del desarrollo del presente capítulo se puede llegar a las siguientes conclusiones:

- Se hizo una breve descripción de la herramienta propuesta y se definieron 18 HU que debe cumplir dicha aplicación.
- Se elaboró el modelo de dominio del negocio así como el diagrama de clases del negocio con el objetivo de entender mejor la aplicación que se desea desarrollar.
- Fueron generados los artefactos que concibe la metodología XP tales como la lista de reserva del producto, tareas de ingeniería, plan de iteraciones y tarjetas CRC.
- Se definió el patrón arquitectónico Modelo-Vista-Controlador y se hizo uso de los patrones de diseño GRAPS y GOF.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA DEL SISTEMA PROPUESTO

Introducción

El contenido que se aborda en este capítulo está relacionado con la descripción de la implementación del sistema, para darle solución a las Historias de Usuario definidas en el capítulo anterior, se realizan las tareas de Ingeniería, se define el estándar de codificación y se especifican las pruebas a las que fue sometida la aplicación.

3.1 Implementación de la aplicación

La implementación tiene como objetivo principal desarrollar la arquitectura y el sistema como un todo, así como definir la organización del código. Para llevarla a cabo se desglosan en tareas de ingeniería las HU definidas en el capítulo anterior, las cuales guían la implementación, siendo así más fácil el desarrollo del sistema logrando una programación eficiente.

3.1.1 Tareas de ingeniería

Las Tareas de Ingeniería (TI) se usan para describir las tareas que se realizan sobre el proyecto las cuales pueden ser de desarrollo, corrección y mejora. Estas tienen relación con una historia de usuario, aunque de una historia de usuario se puede derivar más de una tarea de ingeniería. A las TI se les especifica la fecha de inicio y fin, se nombra al programador responsable de cumplirla, también se estima el tiempo que tardará en realizar cada una de las tareas y se realiza una pequeña descripción de lo que se tratará de hacer en la tarea.
La tabla que se muestra a continuación contiene una de las tareas correspondientes a la HU 3.

Tabla 7. TI Generar datos de tipo Integer de forma aleatoria para PostgreSQL.

Tarea de Ingeniería	
Número Tarea: 8	**Número Historia de Usuario: 3**
Nombre Tarea: Generar datos ficticios de forma aleatoria para PostgreSQL.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 0.8

Fecha Inicio: 07/02/2014	**Fecha Fin:** 12/02/2014
Programador Responsable: Meyker Rodríguez Leyva	
Descripción: En la vista principal hacer clic sobre un atributo de una tabla. Seleccionar el tipo de generación Random, mostrar el formulario correspondiente a este tipo de generación. Subscribirse a la acción clic del botón Acepar, recoger los datos del formulario y hacer un submit para guardar los datos. Subscribirse a la acción clic del botón Regenerar mostrar los datos generados.	

Para ver el resto de las tareas de ingeniería remitirse a los documentos complementarios, Planilla de Tareas de Ingeniería del Expediente de proyecto.

3.1.2 Estándares de codificación

Para asegurar la calidad de un software uno de los instrumentos que se utilizan es la adopción de estándares de codificación. Los mismos tienen un sin números de ventajas como:

- Asegurar la legibilidad del código entre distintos programadores facilitando el traceo del mismo.
- Proveer una guía para el encargado de mantenimiento/actualización del sistema, con código claro y bien documentado.
- Facilitar la portabilidad entre plataformas y aplicaciones.

Los estándares de codificación permiten entender de manera rápida, fácil y sencilla, el código empleado en el desarrollo de un software. Es de gran importancia el usar técnicas de codificación y realizar buenas prácticas de programación con vistas a generar un código de alta calidad. Si se aplica de forma continua un estándar de codificación bien definido, y posteriormente se efectúan revisiones del código, caben muchas posibilidades de que un proyecto de software se convierta en un sistema fácil de comprender y de mantener, garantizando un mantenimiento óptimo de dicho código por parte del programador. Es por ello que para la implementación del Generador de Datos se definen los requisitos que se presentan a continuación.

Identación: La unidad de identado es de 4 espacios. El uso de la tabulación debe ser evitado porque no existe un estándar que determine con precisión el ancho que va a producir la tabulación.(37)

```
public boolean BuscarColumna(String nombre) {
    boolean encontrado = false;
    for (String nomcolumn : list_pk) {
        if (nomcolumn.equals(nombre)) {
            encontrado = true;
        }
    }
    return encontrado;
}
```

Fig. 10 Ejemplo de Identación evidenciado en la implementación de la aplicación.

Declaración de variables: Cada variable debe de ser declarada en una línea y comentada. También deben aparecer ordenadas alfabéticamente. El nombre de las variables debe de comenzar con letras minúsculas y cada palabra relevante por la que esté compuesta debe ser con letra mayúscula.(38)

```
String nombre; // Nombre de la columna
String tipoDato; // Tipo de dato de la columna
```

Fig. 11 Ejemplo de declaración de variables evidenciado en la implementación de la aplicación.

Declaración de funciones: No debe haber espacio entre el nombre de la función y el paréntesis izquierdo, ni entre este y la lista de parámetros. Debe haber un espacio entre el paréntesis derecho y la llave de comienzo del cuerpo de la función. Las sentencias del cuerpo deben estar en la línea siguiente. La llave que cierra debe estar alineada con la línea de declaración de la función. Los nombres de las funciones se rigen por las mismas características que el de las variables.(39)

```
public void generarAuto() {
    this.Tablavalores = new LinkedList();
    generarPrimaryKey();
    generarForeignKey();
}
```

Fig. 12 Ejemplo de declaración de funciones evidenciado en la implementación de la aplicación.

Nombre de las clases: Las clases comenzarán con mayúscula, si la clase es compuesta empezarán con mayúsculas las dos letras iniciales de cada palabra y estarán separadas por un guión bajo.(39)

```
public class Control
```

Fig. 13 Ejemplo de estándar evidenciado en la implementación de la aplicación.

Sentencia return: Una sentencia return no debe utilizar paréntesis "()" alrededor del valor que se retorna. La expresión cuyo valor se retorna debe comenzar en la misma línea que la palabra reservada return, terminada con un punto y coma.(40)

```
public String getNombre() {
    return nombre;
}
```

Fig. 14 Ejemplo de estándar evidenciado en la implementación de la aplicación.

Sentencia if: La sentencia siempre lleva "{}".(41)

```
if (enu instanceof EnumGenerarDouble) {
    TDouble e = new TDouble((EnumGenerarDouble) enu, obj);
    LinkedList f = (LinkedList) Tablavalores.get(pos_tabla);
    f.set(pos_columna, e.getResult());
}
```

Fig. 15 Ejemplo de sentencia "if" evidenciado en la implementación de la aplicación.

Sentencia for: La sentencia for debe tener la siguiente estructura: (42)

```
for (inicializacion; condicion; actualizacion) {
   sentencias;
}
```

Fig. 16 Estructura de la sentencia "for".

A continuación se muestra un ejemplo de cómo se evidencia este estándar en la implementación de la aplicación.

```
for (int k = 0; k < t.getList_columnas().size(); k++) {
    if (t.getList_columnas().get(k).getNombre().equals(nombre)) {
        list_return = (LinkedList) tb.get(k);
    }
}
```

Fig. 17 Ejemplo de sentencia "for" evidenciado en la implementación.

Espacios en blanco: Las líneas en blanco facilitan la lectura determinando secciones de código lógicamente relacionados. Los espacios en blanco pueden ser (o no deben ser) utilizados en las siguientes circunstancias:

- No se debe utilizar un espacio en blanco entre el identificador de una función y el paréntesis que abre la lista de parámetros. Esto ayuda a distinguir entre palabras reservadas y llamadas a funciones.
- Para cualquier operador binario excepto el punto ".", el paréntesis que abre "(" y el corchete que abre "[" todos deben ser separados por un espacio entre operandos y operador.
- No se debe utilizar el espacio para separar un operador unario de su operando excepto cuando ese operador es una palabra como typeof.
- Cada punto y coma ";" en una sentencia de control debe ser seguido por un espacio.
- Debe dejarse un espacio luego de cada coma ",". (43)

```
public int buscarColumn(List lista, String nombre)
```

Fig. 18 Ejemplo de estándar evidenciado en la implementación de la aplicación.

3.1.3 Interfaces de la aplicación.

Para el desarrollo del Generador de Datos ficticios fueron diseñadas e implementadas varias interfaces de usuario para lograr la comunicación entre el usuario y el sistema. En la Fig. 18 se muestra la interfaz de conexión con la base de datos, donde el usuario podrá especificar la base de datos con la cual desea conectarse. En esta se visualizan los parámetros que debe llenar el usuario para la conexión.

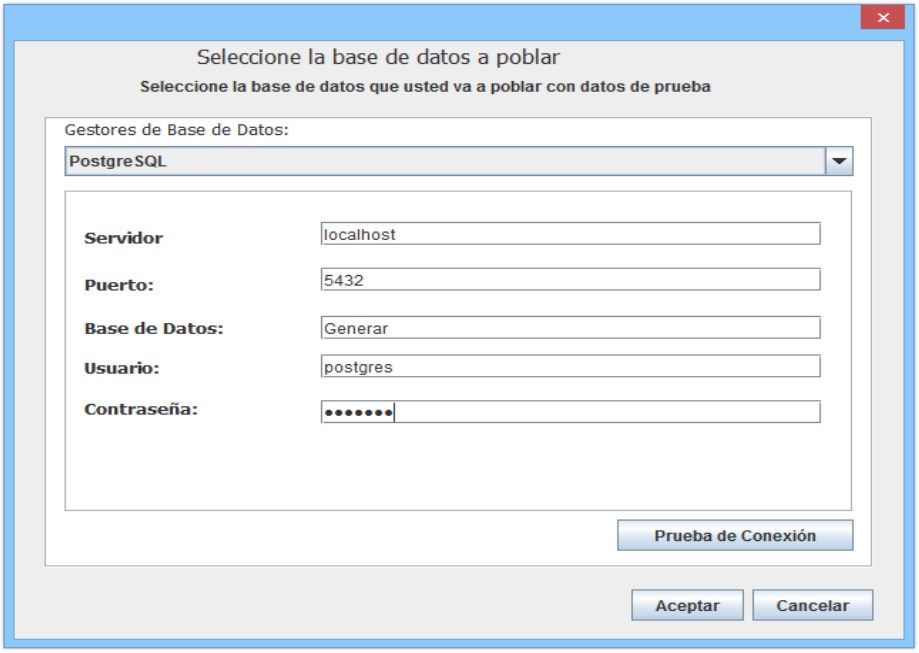


Fig. 19 Interfaz de conexión con la base de datos.

En la siguiente figura se muestra la interfaz de selección de tablas donde se muestran los nombres de las tablas de la base de datos y la cantidad de columnas de cada tabla. En esta interfaz el usuario cuanta con dos opciones, una: seleccionar todas las tablas de la base de datos y dos: seleccionar sólo las que desea poblar.

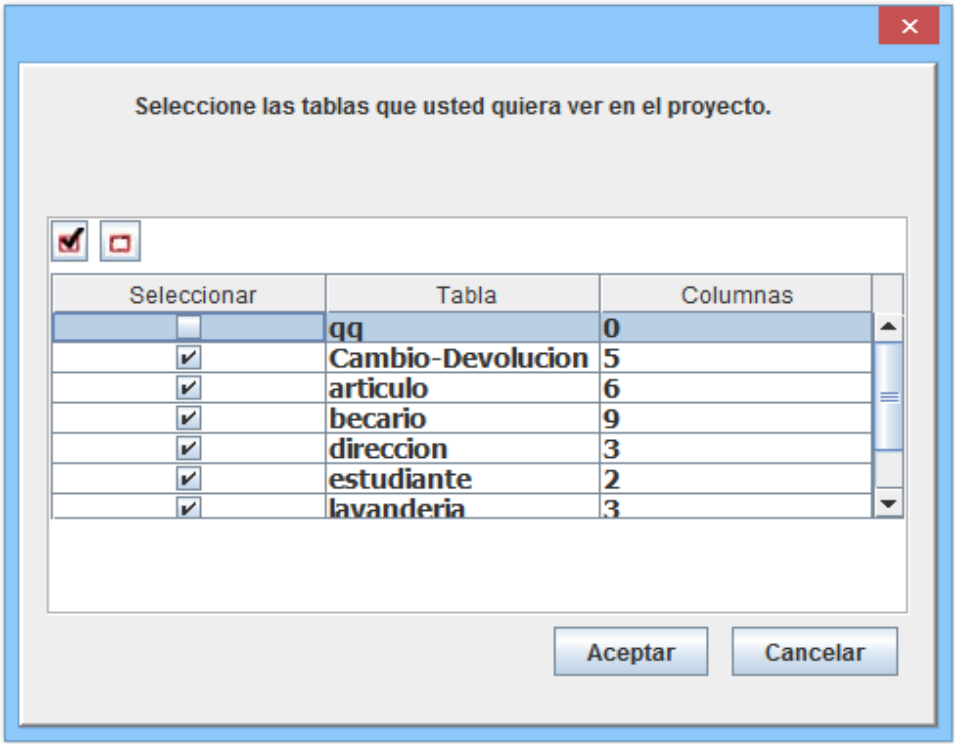


Fig. 20 Interfaz de selección de las tablas que serán pobladas.

En Fig. 20 se muestra la interfaz principal de la aplicación. En esta el usuario tiene la posibilidad de observar las tablas seleccionadas y los tipos de datos de cada una. Además es en esta interfaz donde el usuario tiene la opción de seleccionar el modo de generación de datos que desee y luego llenar los parámetros en dependencia del modo de generación seleccionado. Una vez hecho esto se generan los datos, de los cuales una pequeña parte es mostrada en la parte inferior de la interfaz con el objetivo de brindar al usuario una vista previa de como quedarían los datos en la base de datos.

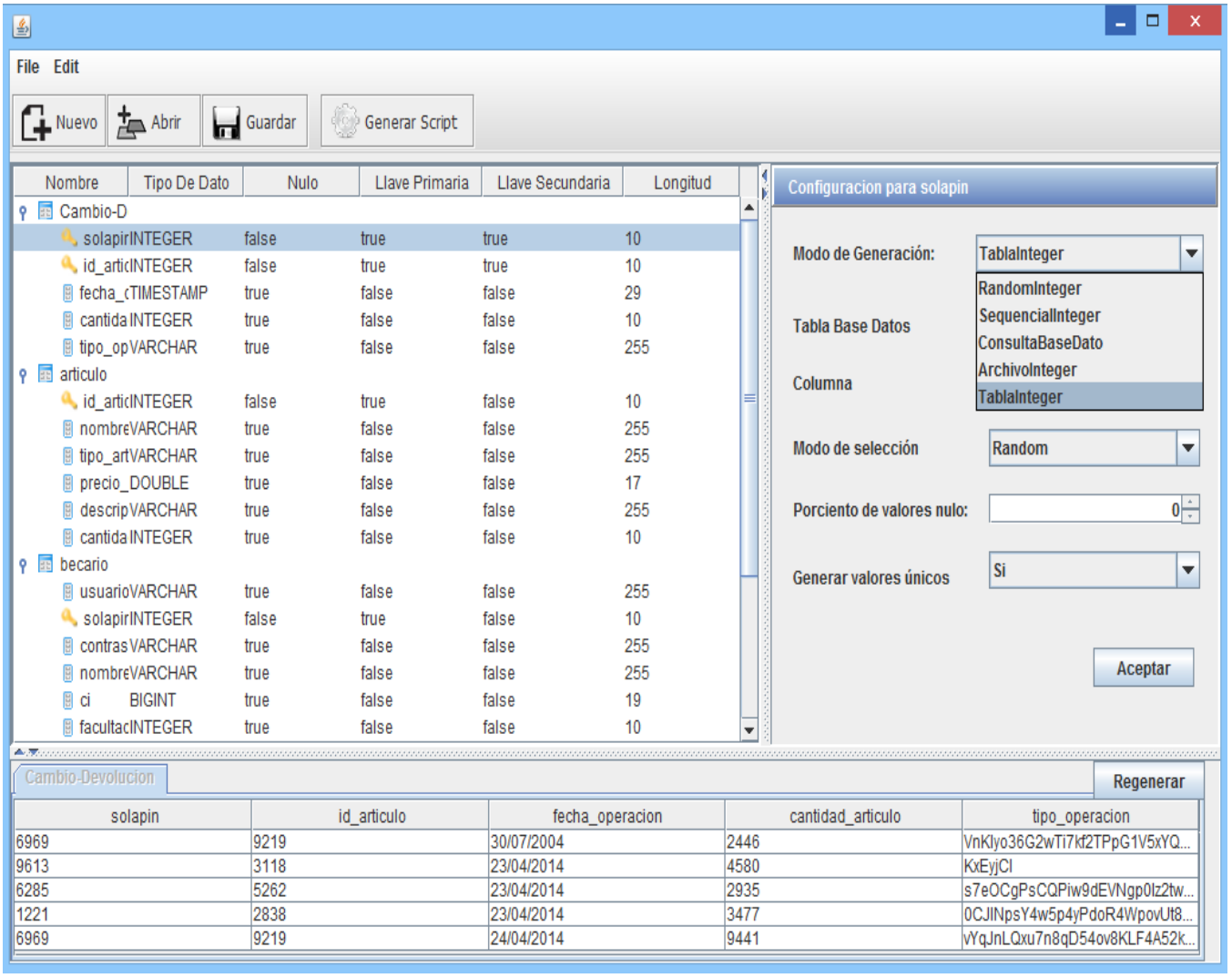


Fig. 21 Interfaz Principal.

3.2 Validación de la solución.

La validación y comprobación del funcionamiento de un producto de software antes de ser liberado permite aumentar su calidad, reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. Permite además aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones.

En XP se definen un grupo de normas para la validación de los productos, logrando que los mismos puedan ser probados para la verificación de su correcto funcionamiento.

3.2.1 Pruebas.

Las pruebas de software juegan un papel decisivo cuando se desea obtener un producto de alta calidad y buen funcionamiento, por lo que su objetivo fundamental es verificar y validar que realmente el software realice lo que se espera de él.

Las pruebas de software constituyen un pilar indispensable para evaluar y determinar la calidad de un software. Concretamente se puede definir pruebas de software como:

- El proceso de ejecución de un programa con la intención de descubrir errores previos a la entrega al usuario final.

- Una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones específicas, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. (44)

La fase de pruebas es una de las fases fundamentales del desarrollo de una aplicación. El objetivo de cada una de las pruebas no es el de prevenir errores sino de detectarlo basándose en técnicas y estrategias empleadas en cada una de las pruebas. Dentro de las estrategias de pruebas existentes pueden mencionarse las pruebas de unidades, las pruebas de integración, las pruebas de sistema, las pruebas de aceptación y pruebas de regresión.

3.2.2 Estrategias de prueba

Un pilar fundamental de la metodología de desarrollo de software XP es el uso de las pruebas para verificar el comportamiento de un programa de forma adecuada.

Como se mencionó anteriormente existen diferentes estrategias de prueba por lo parte de la investigación de este trabajo se ha centrado solamente en el estudio las estrategias correspondientes a la metodología utilizada para el desarrollo de la herramienta generadora de datos, en este caso XP.

XP divide las pruebas en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñadas por los programadores, y pruebas de aceptación, diseñadas por el cliente y destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida.(45)

Pruebas unitarias

Las Pruebas Unitarias son pruebas realizadas a nivel de las clases y métodos construidos para la aplicación. Estas pruebas son implementadas por los mismos programadores en el lenguaje de programación utilizando clases de prueba encargadas de verificar el comportamiento correcto de los métodos en una clase. Para ello es conveniente contar con algún framework de testing unitario que le facilite al equipo de desarrollo la creación, mantenimiento y ejecución de una suite de pruebas automatizadas.(46)

Las pruebas unitarias son una de las piedras angulares de XP. Estas pruebas deben ser definidas antes de realizar el código. Los programadores deben realizar este tipo de prueba cuando la interfaz de un método de la aplicación no es clara, la implementación es complicada, para probar entradas y condiciones inusuales, luego de modificar algo. Éstas deben contemplar cada clase del sistema que pueda generar fallas.

Pruebas de Aceptación

El objetivo de las Pruebas de Aceptación es la verificación de que el software construido en las iteraciones cumple con las funcionalidades solicitadas por el usuario y está listo para ser utilizado en el ambiente del cliente. Estas pruebas revisten gran relevancia porque implican que el cliente da conformidad respecto al sistema desarrollado y lo acepta.(47)

Estas pruebas son definidas por el cliente para cada historia de usuario, y tienen como objetivo asegurar que las funcionalidades del sistema cumplen con lo que se espera de ellas. Las pruebas de aceptación corresponden a una especie de documento de requerimientos en XP, ya que marcan el camino a seguir en cada iteración, indicándole al equipo de desarrollo hacia donde tiene que ir y en qué puntos o funcionalidades debe poner el mayor esfuerzo y atención. Estas pruebas no se realizan durante el desarrollo, pues sería impresentable al cliente; sino que se realizan sobre el producto terminado o pudiera

ser una versión del producto o una iteración pactada previamente con el cliente. Existen dos tipos de pruebas de aceptación:

1. La prueba alfa.

Se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario. Las pruebas alfa se llevan a cabo en un entorno controlado. Para que tengan validez, se debe primero crear un ambiente con las mismas condiciones que se encontrarán en las instalaciones del cliente. Una vez logrado esto, se procede a realizar las pruebas y a documentar los resultados. (48)

2. La prueba beta.

Se lleva a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador.(48)

Como resultado de los problemas informados durante la prueba beta, el desarrollador del software lleva a cabo modificaciones y así prepara una versión del producto de software para toda clase de clientes.

Teniendo en cuenta el estudio realizado se decide llevar a cabo la estrategia de pruebas de aceptación, aplicando los tipos alfa y beta, para verificar que el sistema desarrollado cumple con las funciones específicas para las que se ha creado, debido a que en esta la presentación de resultados es muy importante, en cambio para las pruebas unitarias no tiene mucho sentido ya que siempre se requiere de un total de efectividad en las clases más críticas. Se tiene en cuenta además para la selección de las pruebas de aceptación que es el cliente quien realiza dichas pruebas y en este caso el cliente forma parte del equipo de desarrollo.

3.2.3 Métodos de prueba

Cualquier proyecto que se trace una estrategia de prueba debe contar con métodos para la aplicación de cada una de estas pruebas .A continuación se realiza una breve caracterización de dos métodos importantes para de esta forma seleccionar uno de estos para la correcta realización de las pruebas.

Pruebas de Caja Blanca

Las pruebas de caja blanca realizan un seguimiento del código fuente según se van ejecutando los casos de prueba, de manera que se determinan de forma concreta las instrucciones y bloques en los que existen errores. Conociendo el funcionamiento del producto se pueden desarrollar pruebas de caja blanca, para asegurar que todas las piezas del programa concuerdan, es decir que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada, es decir analiza los caminos de ejecución. Mediante los casos de prueba de caja blanca se pueden derivar casos de prueba que garanticen que:

- Se ejecutan por lo menos una vez todos los caminos independientes de cada módulo.
- Que se ejercitan todas las decisiones lógicas en sus vértices verdaderos y falsos.
- Que se ejecutan todos los bucles en sus límites y con sus límites operacionales.
- Que se ejecutan las estructuras de datos internas para asegurar su validez.

En ocasiones resultan necesarias realizar las pruebas de caja blanca para detectar errores que se producen ante las siguientes situaciones:

- Los errores lógicos y suposiciones, los cuales son proporcionales a la probabilidad de que se ejecute algún cambio en el programa.
- Cuando pensamos que un camino lógico tiene pocas posibilidades de ejecutarse y de hecho se ejecuta de forma normal y esto significa o conlleva a que las suposiciones sobre el flujo de control y datos conlleven a errores de diseño y se encuentran solamente cuando comienza la prueba de camino básico.
- También se pueden encontrar errores tipográficos mediante estas pruebas, lo cual es casi imposible de detectar mediante otro tipo de prueba. (49)

Pruebas de Caja Negra

Teniendo en cuenta la función específica para la que se diseñó el producto, se puede llevar a cabo pruebas que demuestran que cada una de las funciones es totalmente operativa y a la vez buscar errores en cada función. Este tipo de prueba se le llama prueba de caja negra, también conocidas como prueba de funcionalidad. Además permite obtener un conjunto de condiciones de entradas que permitan ejercitar totalmente todos los requisitos funcionales del programa. Estas pruebas se llevan a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa. Mediante los casos de prueba de la caja negra se puede indicar que:

- Las funciones del software son operativas.
- La entrada se acepta de forma adecuada.
- Se produce una salida correcta.
- La integridad de la información externa se mantiene.

La prueba de la caja negra puede descubrir errores de funciones incorrectas o ausentes entre las que se encuentran:

- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.
- Funciones incorrectas o ausentes.

Para desarrollar pruebas de caja negra existen varias técnicas, entre ellas están:

- Técnica de la Partición de Equivalencia: Esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. El objetivo de partición equivalente es reducir el posible conjunto de casos de prueba en uno más pequeño, un conjunto manejable que evalúe bien el software.

- Técnica del Análisis de Valores Límites: Esta Técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables. Los errores tienden a darse más en los límites del campo de entrada que en el centro. Por ello, se ha desarrollado el análisis de valores límites (AVL) como técnica de prueba. El análisis de valores límite lleva a una elección de casos de prueba que ejerciten los valores límite. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL lleva a la elección de casos de prueba en los extremos de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida.
- Técnica de Grafos de Causa-Efecto: Es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones. En este método se debe entender los objetos (objetos de datos, objetos de programa tales como módulos o colecciones de sentencias del lenguaje de programación) que se modelan en el software y las relaciones que conectan a estos objetos. Una vez que se ha llevado a cabo esto, el siguiente paso es definir una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas. (50)

Teniendo en cuenta el estudio realizado sobre los métodos de pruebas antes mencionados se decide aplicar el método de caja negra debido a que es más importante presentarle al cliente que las funcionalidades de la aplicación estén funcionando correctamente y así dejarlo satisfecho y conforme con el producto. Se define emplear además la técnica de partición de equivalencia con el objetivo de probar las entradas en cada caso de prueba.

3.3 Casos de Pruebas basados en Historias de Usuarios.

Para probar las funcionalidades del sistema se diseñaron un total de 20 casos de pruebas correspondientes a las 18 historias de usuarios identificadas. A continuación se muestra un ejemplo de casos de prueba para la HU: Generar datos ficticios de forma aleatoria para PostgreSQL.

Tabla 8. Caso de Prueba: Generar datos ficticios de forma aleatoria para PostgreSQL.

Escenario	Descripción	Valores entre	Porciento de valores	Valores únicos	Respuesta del sistema	Flujo central

			nulos			
EC 1.1 Generar datos ficticios de forma aleatoria para PostgreSQL con datos correctos.	El sistema genera datos aleatorios para el gestor PostgreSQL.	V	V	V		
		20-5000	10	No	El sistema genera los datos correctamente.	El Usuario debe: 1-Seleccionar el modo de generación Random. 2-Especificar el porciento de valores que pueden ser nulos introduciendo un número entero. 3-Seleccionar si desea que los valores sea únicos o no. 4-Hacer clic sobre el botón generar.

A continuación se muestra una tabla donde se describen las variables asociadas al caso de prueba: Generar datos ficticios de forma aleatoria para PostgreSQL.

Tabla 9. Descripción de las variables del caso de prueba: Generar datos ficticios de forma aleatoria para PostgreSQL.

No.	Nombre de campo	Clasificación	Valor nulo	Descripción
Variable 1	Valores entre	Campo de selección (Spinner)	No	Rango donde se encuentran los valores que serán generados.
Variable 2	Porciento de valores nulos	Campo de selección (Text Field)	Si	Por ciento del total de tuplas que pueden ser nulas.
Variable 3	Valores únicos	Campo de selección (Combo Box)	No	Opción de generar valores únicos o no.

Para ver el resto de los casos de prueba con su respectiva descripción de las variables remitirse a los documentos complementarios, Planilla de Casos de Pruebas del Expediente de Proyecto.

3.4 Presentación de los resultados de las pruebas funcionales

Se realizaron 3 iteraciones de pruebas donde en la primera se encontraron en total 12 no conformidades: 6 de validación, 2 de ortografía y 4 de interfaz. En la 2da iteración fueron encontradas 6 no conformidades: 2 de validación, 1 de ortografía, 1 de interfaz y 2 funciones incorrectas. En la 3ra iteración se obtuvieron resultados satisfactorios, encontrándose 0 no conformidades. Antes de pasar a una nueva iteración de prueba se realizaron pruebas de regresión con el objetivo de verificar que las no conformidades

encontradas se habían resuelto o que los cambios realizados no habían provocado nuevas no conformidades.

La siguiente imagen muestra el comportamiento de la cantidad de no conformidades de ortografía, interfaz y validación en las 3 iteraciones.

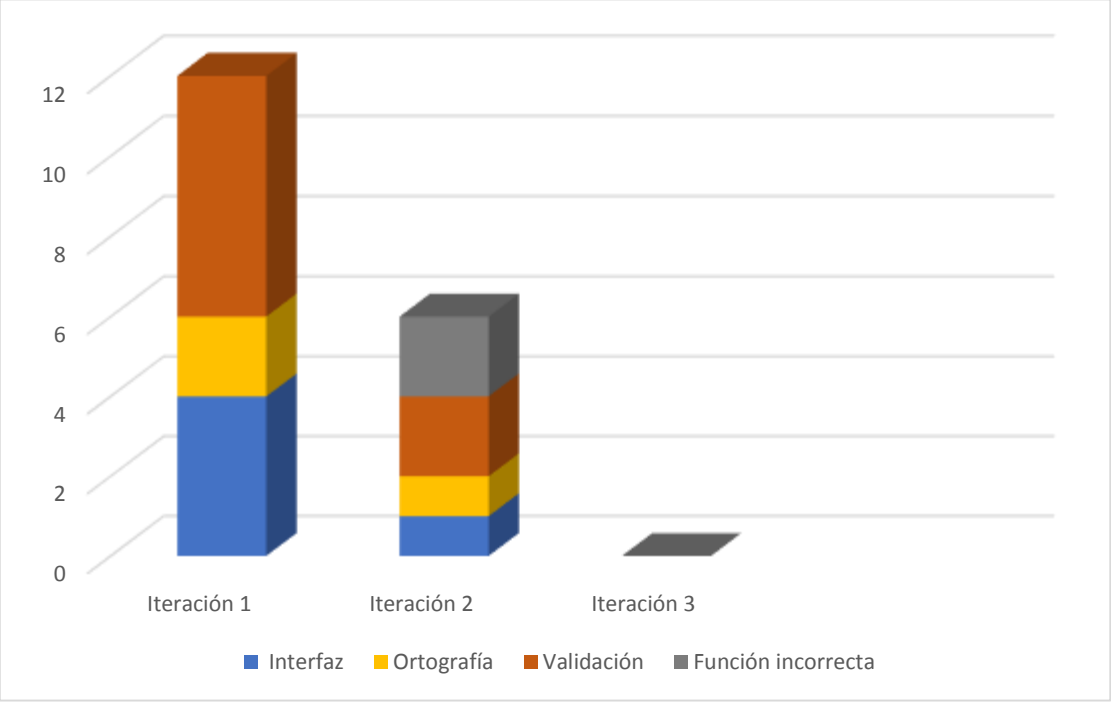


Fig. 22 Representación de las no conformidades encontradas en cada una de las iteraciones de prueba.

Conclusiones parciales del capítulo

Luego del desarrollo del presente capítulo se puede arribar a las conclusiones siguientes:

- Quedaron definidos los estándares de codificación con los que cumple el Generador de Datos desarrollado.
- La realización de las pruebas de caja negra permitió detectar, documentar y corregir las no conformidades existentes en el sistema implementado.
- Al concluir el período de pruebas se obtuvo una aplicación que cumple de forma correcta con la totalidad de las funcionalidades esperadas por el cliente.

CONCLUSIONES GENERALES

- Al realizar el análisis se obtuvieron 20 HU de las que se identificaron 30 requisitos funcionales.
- Se obtuvieron durante el diseño de la herramienta 19 tarjetas CRC así como 21 clases que conformaron el diagrama de clases.
- Se implementó la herramienta MyPostGenerator permitiendo la generación de datos ficticios sobre las base de datos PostgreSQL y MySQL.
- Se validó la solución mediante la confección de 10 casos de prueba, demostrando que la aplicación MyPostGenerator cumple con las funcionalidades identificadas.

RECOMENDACIONES

- Incluir la generación de los tipos de datos: bit varying, bit [(n)], bytea, money, xml.
- Permitir que la herramienta genere datos ficticios para otros gestores de bases de datos relacionales como: SQL Server y Oracle.
- Permitir que la herramienta genere datos ficticios para otros gestores de bases de datos NoSQL como: MongoDB.

REFERENCIAS BIBLIOGRÁFICAS

1. AMBROSIO, D. El concepto de bases de datos. P. 1.

2. Departamento de Ciencias de la Computación e I.A. Introducción a las bases de datos. Fundamentos de diseño de bases de datos. Universidad de Granada.

3. DATE, Christopher J. Introducción a los Sistemas de Bases de Datos 7ma Edición. P. 10.

4. MARQUEZ, Mercedes. Apuntes de Ficheros y Bases de Datos. P. 7.

5. GÓMEZ, Eva, MARTÍNEZ, Patricio, MOREDA, Paloma, SUÁREZ, Armando, MONTOYO, Andrés and SAQUTE, Estela. *Apuntes Bases de Datos 1*. Dpto. de Lenguajes y Sistemas Informáticos Escuela Politécnica Superior Universidad de Alicante, [no date].

6. DOMÍNGUEZ, Rodríguez and TÉLLEZ, Sánchez. Sistema de apoyo a la toma de decisiones en el proceso de negociación comercial. September 2011. P. 10.

7. PostgreSQL. *PostgreSQL* [online]. March 2012. [Accessed 15 October 2013]. Available from: http://www.postgresql.org.es/sobre_postgresql.

8. Sistemas Manejadores de Bases de datos. *Sistemas Manejadores de Bases de datos.* [Online]. February 2011. [Accessed 25 October 2013]. Available from: http://sistemamanejadordebasededatossmbd.blogspot.com/2011/02/diferentes-tipos-de-sistemas-de.html

9. Softpedia. *Softpedia* [online]. Abril de 2012. [Accessed 25 October 2013]. Available from: http://www.softpedia.es/programa-EMS-Data-Generator-for-PostgreSQL-47351.html.

10. programas-gratis.net. [Online]. Available from: http://data-generator-for-mysql.programas-gratis.net/(EMS para MySQL)

11. RIVERA, Bestard and QUIALA, Fonseca. *Generador de datos para PostgreSQL.Postgen.* Universidad de Oriente Santiago de Cuba, [no date].

12. DTM Data Generator. *DTM Data Generator.]* [online]. [Accessed 27 October 2013]. Available from: http://www.sqledit.com/dg/.

13. FIGUEROA, G, SOLÍS, J and CABRERA, A. Metodologías Tradicionales vs. Metodologías Ágiles. P. 1.

14. FIGUEROA, G, SOLÍS, J and CABRERA, A. Metodologías Tradicionales VS. Metodologías Ágiles. P. 5.

15. ESCRIBANO, Fernández Gerardo. Introducción a Extreme Programing. 12 de Diciembre de 2002.

16. RUMBAUGH, James, BOOCH, GRADY and OTROS. *El lenguaje unificado de modelado*. 2000.

17. Lenguaje de Programación. *Lenguaje de Programación* [online]. [Accessed 8 November 2013]. Available from: http://www.lenguajes-de-programacion.com/lenguajes-de-programacion/modelado.shtml.

18. Programming Language Popularity. 2009. P. 16–18.

19. Programacion3. *Programacion3* [online]. [Accessed 17 November 2013]. Available from: http://programacion3.bligoo.com/content/view/3552616/Que-es-lenguaje-Java-y-sus-caracteristicas.html.

20. DOMÍNGUEZ, Dorado. Todo Programación. P. 32–34.

21. Entornos de Programación. [Online]. 2009. Available from: http://lml.ls.fi.upm.es/ep/entornos.html#toc5

22. Programación Desarrollo. [Online]. 2012. Available from: programaciondesarrollo.es/que-es-un-entorno-de-desarrollo-integrado-ide/

23. aprenderaprogramar. *Aprenderaprogramar* [online]. Available from: http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=398:netbeans-eclipse-jcreator-jbuilder-icual-es-el-mejor-entorno-de-desarrollo-ide-para-java-cu00613b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

24. LARMAN, Craig. UML y patrones. Introducción al análisis y diseño orientado a objetos. P. 88.

25. LARMAN, Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos. P. 87.

26. JOSKOWICZ, José. Reglas y Prácticas en eXtreme Programming. February 2008. P. 10.

27. JOSKOWICZ, José. Reglas y Prácticas en Extreme Programming. February 2008. P. 11.

28. Geek the Planet. *Geek the Planet* [online]. 10 May 2011. Available from: http://geektheplanet.net/5461/patrones-de-diseno-%C2%BFque-son-y-para-que-sirven.xhtml.

29. CASTILLO, Oswaldo, FIGUEROA, Daniel and SEVILLA, Héctor. Tripod. [Online]. Available from: http://programacionextrema.tripod.com/index.htm.

30. SCHMULLER, J. Aprendiendo UML en 24 horas. 2000. P. 331.

31. LARMAN, Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos. P. 183–192.

32. LARMAN, Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos. P. 193.

33. LARMAN, Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos. P. 197.

34. LARMAN, Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos. P. 203.

35. LARMAN, Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos. P. 206.

36. CALLEJA, Arias Manuel. Estándares de codificación. P. 2.

37. CALLEJA, Arias Manuel. Estándares de codificación. P. 4.

38. CALLEJA, Arias Manuel. Estándares de codificación. P. 8–9.

39. CALLEJA, Arias Manuel. Estándares de codificación. P. 7.

40. CALLEJA, Arias Manuel. Estándares de codificación. P. 5–6.

41. CALLEJA, Arias Manuel. Estándares de codificación. P. 6.

42. CALLEJA, Arias Manuel. Estándares de codificación. P. 8.

43. JURISCO, N, MORENO, Ana and VEGAS, Sira. *Técnicas de evaluación de software. 2005,* [online]. 2005. Available from: http://eva.uci.cu/file.php/257/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_3/Comun/Tecnicas_de_evaluacion_de_software_Jurisco-Moreno.pdf.

44. GUTIÉRREZ, J, ESCALONA, M, MEJÍAS, M and TORRES, J. *PRUEBAS DEL SISTEMA EN PROGRAMACIÓN EXTREMA* [online]. [no date]. [Accessed 17 May 2014]. Available from: http://www.lsi.us.es/~javierj/investigacion_ficheros/PSISEXTREMA.pdf

45. HERNÁN, Schenone Marcelo. Diseño de una Metodología Ágil de Desarrollo de Software. In: 2004. p. 77.

46. HERNÁN, Schenone Marcelo. Diseño de una Metodología Ágil de Desarrollo de Software. In: 2004. p. 78.

47. Pruebas de Aceptación. *Monografías.com*.

48. COLECTIVO DE AUTORES. Técnicas de prueba. P. 2–3.

49. COLECTIVO DE AUTORES. Técnicas de prueba. P. 9–10.

BIBLIOGRAFÍA

1. AMBROSIO, D. El concepto de bases de datos. P. 1.

2. Departamento de Ciencias de la Computación e I.A. Introducción a las bases de datos. Fundamentos de diseño de bases de datos. Universidad de Granada.

3. DATE, Christopher J. Introducción a los Sistemas de Bases de Datos 7ma Edición. P. 10.

4. MARQUEZ, Mercedes. Apuntes de Ficheros y Bases de Datos. P. 7.

5. GÓMEZ, Eva, MARTÍNEZ, Patricio, MOREDA, Paloma, SUÁREZ, Armando, MONTOYO, Andrés and SAQUTE, Estela. *Apuntes Bases de Datos 1*. Dpto. de Lenguajes y Sistemas Informáticos Escuela Politécnica Superior Universidad de Alicante, [no date].

6. DOMÍNGUEZ, Rodríguez and TÉLLEZ, Sánchez. Sistema de apoyo a la toma de decisiones en el proceso de negociación comercial. September 2011. P. 10.

7. PostgreSQL. *PostgreSQL* [online]. March 2012. [Accessed 15 October 2013]. Available from: http://www.postgresql.org.es/sobre_postgresql.

8. Sistemas Manejadores de Bases de datos. *Sistemas Manejadores de Bases de datos.* [Online]. February 2011. [Accessed 25 October 2013]. Available from: http://sistemamanejadordebasededatossmbd.blogspot.com/2011/02/diferentes-tipos-de-sistemas-de.html

9. Softpedia. *Softpedia* [online]. Abril de 2012. [Accessed 25 October 2013]. Available from: http://www.softpedia.es/programa-EMS-Data-Generator-for-PostgreSQL-47351.html.

10. programas-gratis.net. [Online]. Available from: http://data-generator-for-mysql.programas-gratis.net/(EMS para MySQL)

11. RIVERA, Bestard and QUIALA, Fonseca. *Generador de datos para PostgreSQL.Postgen.* Universidad de Oriente Santiago de Cuba, [no date].

12. DTM Data Generator. *DTM Data Generator.]* [online]. [Accessed 27 October 2013]. Available from: http://www.sqledit.com/dg/.

13. FIGUEROA, G, SOLÍS, J and CABRERA, A. Metodologías Tradicionales vs. Metodologías Ágiles.

14. ESCRIBANO, Fernández Gerardo. Introducción a Extreme Programing. 12 de Diciembre de 2002.

15. RUMBAUGH, James, BOOCH, GRADY and OTROS. *El lenguaje unificado de modelado*. 2000.

16. Lenguaje de Programación. *Lenguaje de Programación* [online]. [Accessed 8 November 2013]. Available from: http://www.lenguajes-de-programacion.com/lenguajes-de-programacion/modelado.shtml.

17. Programming Language Popularity. 2009. P. 16–18.

18. Programacion3. *Programacion3* [online]. [Accessed 17 November 2013]. Available from: http://programacion3.bligoo.com/content/view/3552616/Que-es-lenguaje-Java-y-sus-caracteristicas.html.

19. DOMÍNGUEZ, Dorado. Todo Programación. P. 32–34.

20. Entornos de Programación. [Online]. 2009. Available from: http://lml.ls.fi.upm.es/ep/entornos.html#toc5

21. Programación Desarrollo. [Online]. 2012. Available from: programaciondesarrollo.es/que-es-un-entorno-de-desarrollo-integrado-ide/

22. aprenderaprogramar. *aprenderaprogramar* [online]. Available from: http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=398:netbeans-eclipse-jcreator-jbuilder-icual-es-el-mejor-entorno-de-desarrollo-ide-para-java-cu00613b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

23. LARMAN, Craig. UML y patrones. Introducción al análisis y diseño orientado a objetos.

24. JOSKOWICZ, José. Reglas y Prácticas en eXtreme Programming. February 2008.

25. Geek the Planet. *Geek the Planet* [online]. 10 May 2011. Available from: http://geektheplanet.net/5461/patrones-de-diseno-%C2%BFque-son-y-para-que-sirven.xhtml.

26. CASTILLO, Oswaldo, FIGUEROA, Daniel and SEVILLA, Héctor. Tripod. [Online]. Available from: http://programacionextrema.tripod.com/index.htm.

27. SCHMULLER, J. Aprendiendo UML en 24 horas. 2000. P. 331.

28. LARMAN, Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos.

29. CALLEJA, Arias Manuel. Estándares de codificación.

30. JURISCO, N, MORENO, Ana and VEGAS, Sira. *Técnicas de evaluación de software. 2005,* [online]. 2005. Available from: http://eva.uci.cu/file.php/257/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_3/Comun/Tecnicas_de_evaluacion_de_software_Jurisco-Moreno.pdf.

31. GUTIÉRREZ, J, ESCALONA, M, MEJÍAS, M and TORRES, J. *PRUEBAS DEL SISTEMA EN PROGRAMACIÓN EXTREMA* [online]. [no date]. [Accessed 17 May 2014]. Available from: http://www.lsi.us.es/~javierj/investigacion_ficheros/PSISEXTREMA.pdf

32. HERNÁN, Schenone Marcelo. Diseño de una Metodología Ágil de Desarrollo de Software. In: 2004.

33. Pruebas de Aceptación. *Monografías.com*.

34. COLECTIVO DE AUTORES. Técnicas de prueba.