Universidad de las Ciencias Informáticas

“Facultad 6”

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Título: Plugin de generación de datos para la herramienta de administración de bases de datos HABD

Autor:

Abel Delgado Rodríguez

Tutores:

MSc. Anthony R. Sotolongo León

Ing. Flavio Enrique Roche

La Habana, junio de 2012

"Seamos realistas, hagamos lo imposible"

Che

Declaro ser autor del presente trabajo “Plugin de generación de datos para la herramienta de administración de bases de datos HABD” y reconozco a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ___ días del mes de _____ del año 2012.

_____

Autor

Abel Delgado Rodríguez

_____

Tutor

Msc. Anthony R. Sotolongo León

_____

Tutor

Ing. Flavio E. Roche Rodríguez

DATOS DE CONTACTO

Tutores:

MSc. Anthony R. Sotolongo León

Especialidad de graduación: Ingeniero Informático

Años de experiencia en el tema: 3 años

Años de graduado: 6 años

e-mail: asotolongo@uci.cu

Ing. Flavio Enrique Roche

Especialidad de graduación: Ingeniero en Ciencias Informáticas

Años de experiencia en el tema: 1 años

Años de graduado: 1 años

e-mail: feroche@uci.cu

Agradecimientos

Primero que todo quiero agradecer a la persona más importante y que más quiero en mi vida: mi mamá, porque gracias a ella hoy he llegado hasta aquí.

A mi familia por su apoyo y por de una forma u otra influir en este logro, especialmente a mi abuela Tita, a Rodolfo, a mi papá, a mis hermanos Ale, Luis Mario y Orestico.

Al Papo, más que mi primo, mi hermano, por compartir junto a mí los buenos y malos momentos en la universidad.

A mi novia Rosy, por su cariño y por todo su apoyo en este tiempo tan difícil aun estando tan lejos. Por todos los momentos de felicidad que hemos vivido.

A Héctor, mi hermano de guerra en estos "seis" años.

A mis amigos de la UCI y mis amigos de San Juan.

A Yanelis Benítez, por ser la mejor profesora de la facultad, por ayudarme cuando más lo necesité.

A las personas que sin su ayuda este trabajo no hubiera sido posible, mis tutores Flavio y Anthony, y especialmente a mis amigos Kmilo, Evelyn y Dayné.

A todas las personas que hicieron posible la realización de este trabajo.

A mi mamá, todos mis logros son para ti.

A mi familia.

RESUMEN

Los usuarios que actualmente trabajan con el sistema gestor de Bases de Datos PostgreSQL necesitan realizar pruebas antes del despliegue de una solución, para comprobar el comportamiento de una base de datos. Para que los desarrolladores puedan realizar estas pruebas necesitan un volumen de datos que se asemejen a los reales, crear dicho volumen manualmente llevaría un gasto de tiempo considerable. Es por ello que el objetivo de este trabajo es desarrollar un plugin para la herramienta de administración de bases de datos PostgreSQL HABD, que permita la generación de datos ficticios sobre las tablas de las bases de datos en PostgreSQL. Para ello se realizó un estudio del funcionamiento de las aplicaciones ya existentes que realizan la generación de datos para los distintos gestores de base de datos, así como las arquitecturas y técnicas más utilizadas para este tipo de herramientas. Como resultado se obtuvo una herramienta que permitirá poblar tablas de base de datos, dando la posibilidad al usuario de escoger el tipo de generación (aleatoria o desde una lista) y configurar los parámetros de generación en cada fila. La realización de este plugin constituye una alternativa de gran ayuda en la utilización y prueba de bases de datos tanto en los ambientes productivos, como no productivos.

Palabras clave:

Base de datos, datos ficticios, HABD, PostgreSQL.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La evolución de las tecnologías de la información y las comunicaciones en los últimos años ha tenido un desarrollo acelerado. Actualmente la información constituye uno de los pilares fundamentales en el desarrollo de cualquier país, la necesidad de almacenarla de forma rápida y fácil se convierte en una prioridad para las empresas, pues tenerla organizada permite que esta pueda ser accedida en cualquier momento por todas aquellas personas que la necesiten. Con el desarrollo de la informática se comienzan a crear sistemas de información que con el transcurso del tiempo y el perfeccionamiento de las técnicas se ha llegado a lo que hoy se conoce como base de datos.

Se le llama base de datos justamente a esta colección de datos recopilados y estructurados que existen durante un periodo de tiempo. Estas tienen mayor calidad, flexibilidad y manejo desde la creación de los sistemas gestores de bases de datos, los cuales se diseñan e implementan para gestionar grandes cantidades de información. 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.

Un punto crítico en el trabajo con bases de datos es la realización de pruebas antes del despliegue pues el comportamiento generalmente no es el mismo en desarrollo que en producción. Para que los desarrolladores puedan realizar las pruebas necesitan un volumen de datos que se asemeje a la realidad, crear dicho volumen manualmente llevaría un gasto de tiempo considerable. Existen pocas herramientas capaces de realizar esta función, la más reconocida es el EMS Data Generator para PostgreSQL pero no es libre ni de código abierto lo cual dificulta su utilización, además presenta incompatibilidades con las versiones más recientes del gestor. Otra herramienta es el Postgen la cual es de código abierto y desarrollada en C++ y Qt como framework, pero solo es capaz de generar datos de tipo texto y numérico.

Con la misión de producir software y brindar servicios informáticos a partir de la vinculación estudio - trabajo en el año 2002 surge la Universidad de las Ciencias Informáticas (UCI). En la universidad existen diferentes centros productivos dentro de los que se destaca el Centro de Tecnologías de Gestión de Datos (DATEC), el mismo está compuesto por varios departamentos dentro de los que se encuentra PostgreSQL, el cual tiene como meta fundamental, contribuir a la soberanía tecnológica potenciando el uso de bases de datos libres.

En el curso 2010-2011 en este departamento se presentó el trabajo de diploma Exploración y diseño de la Herramienta de Administración de Bases de Datos para PostgreSQL (HABD). Este trabajo

propone el desarrollo de una nueva herramienta de administración a partir de las investigaciones realizadas sobre los inconvenientes encontrados en las principales herramientas para la administración de bases de datos que existen en el mundo.

La herramienta HABD tiene como característica fundamental que posee una arquitectura basada en plugin, lo que facilita a los desarrolladores incorporar nuevos servicios de manera sencilla por la flexibilidad que esto permite. De esta forma se le pueden agregar un gran número de funcionalidades evitando así que los usuarios necesiten de varias aplicaciones para resolver un único problema.

HABD no posee la funcionalidad de generar datos ficticios sobre las tablas de las bases de datos, elemento fundamental para el desarrollo de las pruebas de los sistemas que utilizan PostgreSQL.

Teniendo en cuenta lo anteriormente expuesto se define como **problema de investigación:** ¿Cómo lograr la generación de datos ficticios sobre las tablas de datos en PostgreSQL en la herramienta de administración HABD?

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

Se identifica como **objetivo general:** Desarrollar un plugin para la herramienta de administración de bases de datos en PostgreSQL HABD, que permita la generación de datos ficticios sobre las tablas de las bases de datos en PostgreSQL.

A partir del objetivo general, se derivan los **objetivos específicos** que se enuncian a continuación:

- ✓ Realizar el análisis de las técnicas y herramientas existentes para la generación de datos.
- ✓ Implementar el plugin de la generación de datos para la herramienta HABD.
- ✓ Realizar pruebas al producto obtenido.

Para dar cumplimiento a los objetivos planteados se realizarán las siguientes **tareas de investigación:**

- ✓ Caracterización de las técnicas y herramientas relacionadas con la generación de datos ficticios para bases de datos.
- ✓ Definición de las tecnologías a utilizar.
- ✓ Identificación y descripción de las funcionalidades del plugin de generación de datos.

- ✓ Implementación de las funcionalidades identificadas.
- ✓ Diseño de casos de pruebas.
- ✓ Aplicación de los casos de pruebas en la herramienta implementada.

Estructura del trabajo

El 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 capítulo.

Capítulo 1: Fundamento teórico

En el capítulo 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: Características del sistema propuesto

En el presente capítulo se describe la propuesta de solución para el problema de la investigación. Se determinan los requisitos no funcionales y las historias de usuario con los que deberá cumplir el plugin. Además se generan los artefactos que propone para la etapa de diseño la metodología XP. Se definen los patrones de arquitectura y diseño que se utilizarán.

Capítulo 3: Implementación y pruebas

En este capítulo se describen las funcionalidades principales del plugin, así como las pruebas realizadas para verificar que responde correctamente y está listo para ser utilizado.

CAPÍTULO 1: Fundamento Teórico

Introducción

En este capítulo serán abordados los principales conceptos referentes al dominio del problema. Se seleccionará la metodología de desarrollo de software que más se ajuste a las necesidades del trabajo, así como las herramientas y tecnologías.

1.1 Conceptos asociados a la investigación

Con el paso del tiempo y el avance tecnológico que trae consigo, la información que se genera es cada vez mayor. Las bases de datos dejaron de ser una aplicación más y se convirtieron en una parte fundamental de cualquier sistema. Hoy en día son imprescindibles para almacenar ese gran volumen de datos en crecimiento.

Base de datos

El Glosario IEEE de Ingeniería del Software (IEEE 1990) define el término base de datos de la siguiente forma: Una colección de datos interrelacionados almacenados conjuntamente en uno o más ficheros de computadora (1).

Una base de datos es una colección de datos estructurados según un modelo que refleje las relaciones y restricciones existentes en el mundo real. Los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de éstas, y su definición y descripción han de ser únicas estando almacenadas junto a los mismos. Por último, los tratamientos que sufran estos datos tendrán que conservar la integridad y seguridad de éstos (2).

Otro de los conceptos es aportado por el Dr. Christopher J. Date, quien define a las bases de datos como: un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa dada (3).

A partir de los conceptos brindados por una serie de autores estudiosos del tema se puede llegar a la conclusión que una base de datos es un conjunto de datos de un mismo contexto, organizados de tal forma que puedan ser accedidos de forma rápida y sencilla.

Sistema gestor de bases de datos

Las bases de datos necesitan de una herramienta que sea capaz de administrarlas y controlarlas, surgen así los sistemas gestores de base de datos.

El concepto de sistema gestor de bases de datos se define en el Glosario IEEE de Ingeniería del Software de la siguiente forma: Un sistema informático compuesto por hardware, software o ambos, que proporciona una técnica sistemática para la creación, el almacenamiento, el procesamiento y la consulta de la información almacenada en base de datos. Un sistema gestor de bases de datos actúa como un intermediario entre las aplicaciones y los datos, o bien entre los datos y la base de datos. (1)

Un sistema gestor de base de datos es un conjunto de programas que administran y gestionan la información contenida en una base de datos. Ayudando a realizar las siguientes acciones (4):

- ✓ Definición de los datos.
- ✓ Mantenimiento de la integridad de los datos dentro de la base de datos.
- ✓ Control de la seguridad y privacidad de los datos.
- ✓ Manipulación de los datos.

Es posible definir un sistema gestor de bases de datos como un conjunto de aplicaciones capaces de administrar las bases de datos, que permiten la interacción entre estas y los usuarios, haciendo posible acceder de forma organizada a los datos almacenados.

Uno de los sistemas gestores de bases de datos más usados hoy en día es PostgreSQL, este es distribuido bajo licencia BSD y su código fuente se encuentra disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarle a otras bases de datos comerciales.

Sus características técnicas la hacen una de las bases de datos más potentes y robustas del mercado. Su desarrollo comenzó hace más de 16 años, y durante este tiempo, estabilidad, potencia, robustez, facilidad de administración e implementación de estándares han sido las características que más se han tenido en cuenta durante su desarrollo. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo al sistema (5).

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

Plugin

Un plugin es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande (6).

El uso de plugins se ha convertido en una gran tendencia. Muchas empresas productoras de software crean aplicaciones basadas en esta tecnología aprovechando sus ventajas, tales como:

- ✓ Bajo acoplamiento: Permite modificar el código sin afectar el funcionamiento de otros componentes ni del sistema en general.
- ✓ Simplificación de pruebas: Es posible probar cada plugin o funcionalidad nueva antes de probar el sistema.
- ✓ Mejoras continuas: Aún terminado y desplegado el software es posible seguir trabajando en el mismo agregando nuevas funcionalidades.

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.

1.2 Herramientas de generación de datos

Contar con una base de datos poblada con un gran volumen de datos, es imprescindible a la hora de probar un software que implique la utilización de bases de datos. Existen diferentes herramientas que realizan esta función. Específicamente para PostgreSQL las herramientas se encuentran:

EMS Data Generator: Es una aplicación para generar datos de prueba para varias tablas de bases de datos de PostgreSQL a la vez.

La aplicación de asistente permite definir tablas y campos para generar datos, establecer intervalos de valores, generar campos de gráfico por máscara, cargar valores de campos BLOB desde archivos, obtener listas de valores de consultas SQL y muchas otras características para generar datos de prueba en una forma simple y directa.

También ofrece una aplicación de consola, que permite generar datos mediante plantillas de generación (7).

EMS Data Generator no tiene en cuenta las relaciones entre tablas, por lo que en el caso de que una tabla tenga una llave extranjera, no podrá ser llenada.

La aplicación tiene como limitación principal que no es de código abierto y tiene una licencia Shareware, por lo que permite evaluar de forma gratuita el producto, pero hay que pagar para usar de forma completa el software. Además no es compatible con las últimas versiones del gestor.

Datanamic Data Generator MultiDB: Es un generador de datos que permite a los desarrolladores poblar bases de datos fácilmente con miles de filas de datos de prueba sin significado y sintácticamente correctos. El mismo se conecta a la base de datos y muestra las tablas y columnas con sus respectivos parámetros de configuración. La herramienta se puede utilizar para generar datos de prueba partiendo de cero o de datos existentes y es posible además generar un script de inserción, en el caso de que se quiera poblar la base de datos en otro momento.

Como su nombre lo indica: MultiDB, esta herramienta es capaz de trabajar con múltiples sistemas gestores de bases de datos: Oracle, MySQL, MS SQL Server, PostgreSQL y bases de datos de MS Access (8).

Al igual que EMS Data Generator, Datanamic Data Generator MultiDB es privativo y solo posee gratis una versión de prueba.

Postgen: Fue creado en el Centro de Desarrollo de Software de Santiago de Cuba a partir de la necesidad de una herramienta de código abierto, para generar datos de una base de datos. Este es multi-plataforma, de fácil manejo para el usuario. Permite configurar valores por rangos, insertar datos en varias tablas y el control automático sobre la integridad de referencia para la generación ligada de los datos de las tablas.

Actualmente está en su primera versión y tiene como deficiencia fundamental que solo es capaz de generar datos de tipo texto y numérico.

Tanto EMS Data Generator, Datanamic Data Generator MultiDB y Postgen presentan el inconveniente adicional de no presentar la arquitectura adecuada para ser integrados como plugin al HABD.

1.3 Metodología de desarrollo de software

Una metodología de desarrollo de software es un conjunto de pasos y procedimientos que deben seguirse para desarrollar software. Elegir la mejor metodología para un determinado proyecto es de suma importancia, pues de ello depende en gran medida el éxito o el fracaso del mismo.

Las metodologías pueden seguir uno o diferentes modelos del ciclo de vida de un software, es decir que esto indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no cómo hacerlo. Tienen como principal objetivo obtener un producto final con gran calidad y principalmente que cumpla con todas las especificaciones descritas por el cliente. Estas especificaciones o requisitos como también se les puede llamar tienden a variar mucho con el transcurso del tiempo, es por esto que han surgido diferentes metodologías de desarrollo, dentro de estas se destaca la metodología de desarrollo de software Extreme Programming (XP), la cual es ágil y tiene como meta satisfacer las necesidades del cliente, es decir tratar de brindar un software como el cliente lo necesite y en el momento que lo necesite.

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.

La metodología XP se basa en (9)**:**

- ✓ Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos con el objetivo de detectar futuros errores.
- ✓ Re fabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- ✓ Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

Se selecciona XP como la metodología a utilizar, ya que es la que más se ajusta a las necesidades del proyecto. Además de ser la metodología ágil predefinida por el departamento y el proyecto en general.

1.4 Tecnologías y herramientas a utilizar

Para lograr un mejor perfeccionamiento del plugin a desarrollar, se hizo una investigación acerca de las tecnologías y herramientas existentes en el mundo, para de esta forma seleccionar las que más se ajusten a las necesidades del proyecto. A continuación se describen las mismas:

1.4.1 Lenguaje de modelado

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios (10).

Algunas de las propiedades de UML como lenguaje de modelado estándar son (10):

- ✓ Concurrencia, es un lenguaje distribuido y adecuado a las necesidades de conectividad actuales y futuras.
- ✓ Ampliamente utilizado por la industria.
- ✓ Reemplaza a decenas de notaciones empleadas con otros lenguajes.
- ✓ Modela estructuras complejas.
- ✓ Las estructuras más importantes que soporta tienen su fundamento en las tecnologías orientadas a objetos, tales como: objetos, clases, componentes y nodos.
- ✓ Emplea operaciones abstractas como guía para variaciones futuras, añadiendo variables si es necesario.
- ✓ Comportamiento del sistema: casos de uso, diagramas de secuencia y de colaboración, que sirven para evaluar el estado de las máquinas.

1.4.2 Herramienta de Ingeniería de Software Asistida por Computación

Las herramientas CASE son un conjunto de métodos, utilidades y técnicas que dan asistencia a los analistas, ingenieros de software y desarrolladores, destinadas a facilitar el desarrollo de software aumentando su productividad y reduciendo el coste de las mismas en términos de tiempo y de dinero. CASE es una sigla que corresponde a las iniciales de: Computer Aided Software Engineering y en su traducción al español significa Ingeniería de Software Asistida por Computación.

Las herramientas CASE se han venido ampliando y desarrollando, entre estas se destaca Visual Paradigm.

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, a un menor coste.

Ventajas de usar Visual Paradigm (11)**:**

- ✓ Permite insertar los dibujos de Microsoft Visio en cualquier diagrama UML.
- ✓ Perfecta integración con los principales IDEs, incluyendo Microsoft Visual Studio, Borland Jbuilder, Eclipse y NetBeans.
- ✓ Integración completa con Microsoft Office.
- ✓ Es multiplataforma. Visual Paradigm está disponible y probado en diferentes sistemas operativos incluyendo Windows, Linux y Mac OS X.
- ✓ Análisis textual para la identificación de requerimientos del sistema y clases candidatas.
- ✓ Soporte para tarjetas CRC.
- ✓ Conversión instantánea de código fuente y ejecutables a modelos. Soporta lenguajes y formatos como XML, .NET, archivos .exe, código, clases y .jar de Java, código C++ y CORBA IDL.
- ✓ Diseño de diagramas de forma automática.

Se escogió Visual Paradigm 8.0 como herramienta CASE a utilizar ya que la misma soporta el ciclo de vida de desarrollo de un software, además de garantizar la reproducción de código para la mayoría de

los lenguajes de programación como C++ en la cual está implementada la herramienta a la que se le integrará dicho plugin, además de ser la herramienta definida por el proyecto.

1.4.3 Framework de desarrollo

Un framework no es más que una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que se añaden las últimas piezas para construir una aplicación concreta.

Estos tienen como principal objetivo apresurar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones. Existen diferentes frameworks que facilitan el trabajo con el lenguaje de programación C++, dentro los cuales se destaca Qt, el cual es un framework multiplataforma utilizado para la creación de aplicaciones y está diseñado en dicho lenguaje. Este agrega algunas características a C++ como son: bucle foreach, sentencia forever, entre otras.

Estas son algunas de las ventajas de usar Qt (12):

- ✓ Usa un editor visual: Las herramientas de desarrollo visuales permiten a los diseñadores crear interfaces más amigables y fluidas. Además automatiza una gran parte del trabajo tedioso que requiere la especificación de las dimensiones y características de los elementos de la interfaz de usuario.
- ✓ Es gratuito y de código abierto: Todas las herramientas de Qt son gratuitas y está permitido usarlas para crear proyectos comerciales. Además nuevas herramientas y complementos aparecen de forma regular como beneficio de pertenecer a la comunidad Open Source (Código Abierto).
- ✓ Multi-plataforma: Desarrollar aplicaciones que funcionen tanto en MacOS como en Windows es bastante complicado sin Qt. Por eso algunas aplicaciones populares como Google Earth o Skype lo utilizan.
- ✓ Estabilidad y calidad: Qt existe desde 1992 y desde entonces todo ha sido probado, usado profesionalmente y mejorado muchas veces, resultando en muy buena fiabilidad y facilidad de uso.

- ✓ Gran comunidad online: Ante cualquier problema o duda siempre existirán personas bien informadas y dispuestas a ayudar.
- ✓ Programadores con experiencia pueden usar C++: Qt no es un nuevo lenguaje de programación, con conocimientos de C++ y/o Java no resulta difícil utilizar Qt ya que la mayoría de los proyectos de Qt están escritos en C++.
- ✓ Documentación: Desde su surgimiento hasta el día de hoy se ha acumulado una gran cantidad de documentación. Tanto tutoriales como fragmentos de código.

Para la implementación del trabajo se decidió utilizar el Framework Qt en su versión 4.7 ya que es el framework que se utiliza actualmente en el departamento PostgreSQL para el desarrollo de plugins, además de las grandes posibilidades que brinda para los diseños gráficos y su gran completamiento de código.

1.4.4 Lenguaje de programación

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes (13).

Los lenguajes de programación brindan un modo práctico para que los seres humanos puedan dar instrucciones a un equipo. Estos facilitan la tarea de programación, ya que disponen de formas apropiadas que permiten ser leídas y escritas por personas, es correcto decir que estos son independientes del modelo de computador a utilizar.

Un lenguaje de programación puede ser de alto o bajo nivel. En los de bajo nivel las instrucciones son simples y cercanas al funcionamiento de la máquina, como por ejemplo el código máquina y el ensamblador. Dentro de los lenguajes de bajo nivel se encuentra el lenguaje C++, el cual es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original (14).

C++ se comenzó a desarrollar en 1980 y su autor fue B. Stroustrup. Al comienzo era una extensión del lenguaje C que fue denominada C with classes (C con clases). Este nuevo lenguaje comenzó a ser

utilizado fuera de la ATT en 1983. El nombre C++ es también de ese año, y hace referencia al carácter del operador incremento de C (++). Ante la gran difusión y éxito que iba obteniendo en el mundo de los programadores, la ATT comenzó a estandarizarlo internamente en 1987. En 1989 se formó un comité ANSI (seguido algún tiempo después por un comité ISO) para estandarizarlo a nivel americano e internacional (15).

- ✓ Es un lenguaje compilado:

  C++ compila directamente al código nativo de la máquina, lo que le permite ser uno de los lenguajes más rápidos del mundo.

- ✓ Es un lenguaje fuertemente tipado:

  C++ es un lenguaje que espera que el programador sepa lo que está haciendo, facilitándole un gran control como consecuencia de ello.

- ✓ Es compatible con la comprobación de tipo estático y dinámico:

  C++ permite verificar las conversiones de tipos, en tiempo de compilación o en tiempo de ejecución. Sin embargo la mayor parte del tipo de comprobación de C++ es estática.

- ✓ Posibilidad de usar diferentes paradigmas:

  C++ ofrece un apoyo notable para los paradigmas de programación procedurales, genérico y orientado a objetos.

- ✓ Portabilidad:

  Como uno de los lenguajes más utilizados en el mundo y como un lenguaje abierto, C++ tiene una amplia gama de compiladores que se ejecutan en diferentes plataformas. Esto posibilita que un código escrito en C++ pueda ser compilado en casi todo tipo de ordenadores y sistemas operativos.

- ✓ Gran soporte de librerías:

  En el sitio web SourceForge es posible encontrar más de tres mil librerías para C++.

Se utiliza C++ por ser el lenguaje para el cual está creado el framework Qt, además el plugin que se desarrollará requiere ser integrado a la herramienta de administración de base de datos HABD, la cual está desarrollada en dicho lenguaje.

1.4.5 Entorno de desarrollo

Un entorno de desarrollo integrado o en inglés Integrated Development Environment (IDE) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un solo lenguaje de programación, o bien poder utilizarse para varios.

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.

El IDE utilizado es QtCreator, es multiplataforma y es distribuido bajo tres licencias:

- ✓ Qt Commercial Developer License
- ✓ Qt GNU LGPL v.2.1
- ✓ Qt GNU GPL v.3.0

El principal objetivo de QtCreator es facilitar una plataforma, con un completo entorno de desarrollo para desarrollar proyectos de Qt. Está disponible para Linux, Mac OS X y las plataformas de Windows.

Entre sus características se destacan:

- ✓ Avanzado editor de código C++.
- ✓ Soporta los lenguajes C#, .NET, Python, Ada, Pascal, Perl, PHP y Ruby.
- ✓ Posee una interfaz gráfica integrada y un diseñador de formularios.
- ✓ Herramientas para la administración de proyectos.
- ✓ Ayuda sensible al contexto.
- ✓ Depurador visual.
- ✓ Resaltado y autocompletado de código.

De los IDE existentes se decidió utilizar QtCreator 2.2 por ser potente y capaz de crearle una interfaz gráfica a una aplicación realizada en C++. Posibilita la integración de la herramienta desarrollada con la herramienta de administración de base de datos HABD, además de ser el IDE de desarrollo predefinido por el departamento PostgreSQL.

Conclusiones del capítulo

En este capítulo se realizó un estudio de las principales herramientas existentes en el mundo para la generación de datos ficticios permitiendo lograr un mejor entendimiento de este proceso y específicamente detallar cada una de estas, encontrando sus inconvenientes y deficiencias así como sus potencialidades. Se seleccionaron las herramientas y tecnologías para el desarrollo del plugin, todas de alto nivel y eficiencia. Se determinó usar XP como metodología de desarrollo, UML 2.0 como lenguaje de modelado y Visual Paradigm 8.0 como herramienta CASE. Para la implementación del plugin C++ como lenguaje de programación, Qt 4.7 como framework y QtCreator 2.2 como IDE de desarrollo.

CAPÍTULO 2: Características del sistema propuesto

Introducción

En el presente capítulo se hace una descripción detallada de la propuesta de solución para el problema de la investigación identificado. Se determinan los requisitos no funcionales y las Historias de Usuario con los que deberá cumplir el plugin desarrollado. Además se generan los artefactos que propone la metodología de desarrollo de software XP para la etapa de diseño. Se definen los patrones de arquitectura y diseño que se utilizarán.

2.1 Descripción de la solución propuesta

Para dar solución al problema de la investigación identificado se propone el desarrollo de un plugin capaz de integrarse a la herramienta de administración de base de datos HABD que posibilite la generación de datos ficticios sobre las tablas de bases de datos PostgreSQL.

El usuario podrá seleccionar qué base de datos y qué tablas de la base de datos seleccionada serán pobladas, permitiendo especificar qué cantidad de datos serán generados para cada tabla. De cada una de las tablas el usuario podrá escoger que campos de la misma desea poblar y si prefiere hacerlo con datos existentes en una lista (la cual podrá crear) o con datos aleatorios. La selección del tipo de generación se especificará a nivel de cada campo de la tabla. Si selecciona de forma aleatoria, podrá configurar la forma en que se generarán los datos en dependencia de su tipo.

El plugin será capaz de generar los tipos de datos más usados de PostgreSQL, como son:

- ✓ Número
  - Integer
  - Smallint
  - Bigint
  - Float
- ✓ Texto
  - Text

- Character
- Character varying

✓ Cadena de bits

- Bit
- Bit varying

✓ Boolean

✓ Date

2.2 Modelo de dominio

El Modelo de Dominio (MD) es un artefacto que se genera en la fase de análisis de un proyecto, mediante este, se analizan los conceptos que utilizan los usuarios y con los que deberá trabajar una aplicación. Este es utilizado por el analista como intermediario para comprender el negocio del sistema a realizar.

La metodología XP no genera precisamente este artefacto, pero se decidió hacerlo porque mediante el mismo se agiliza el proceso de entendimiento de las clases conceptuales, de la forma más fácil para aquellas personas que interactúen con el plugin.

A continuación se presenta (Figura 1) el modelo de dominio que se realizó para comprender mejor el funcionamiento del plugin:

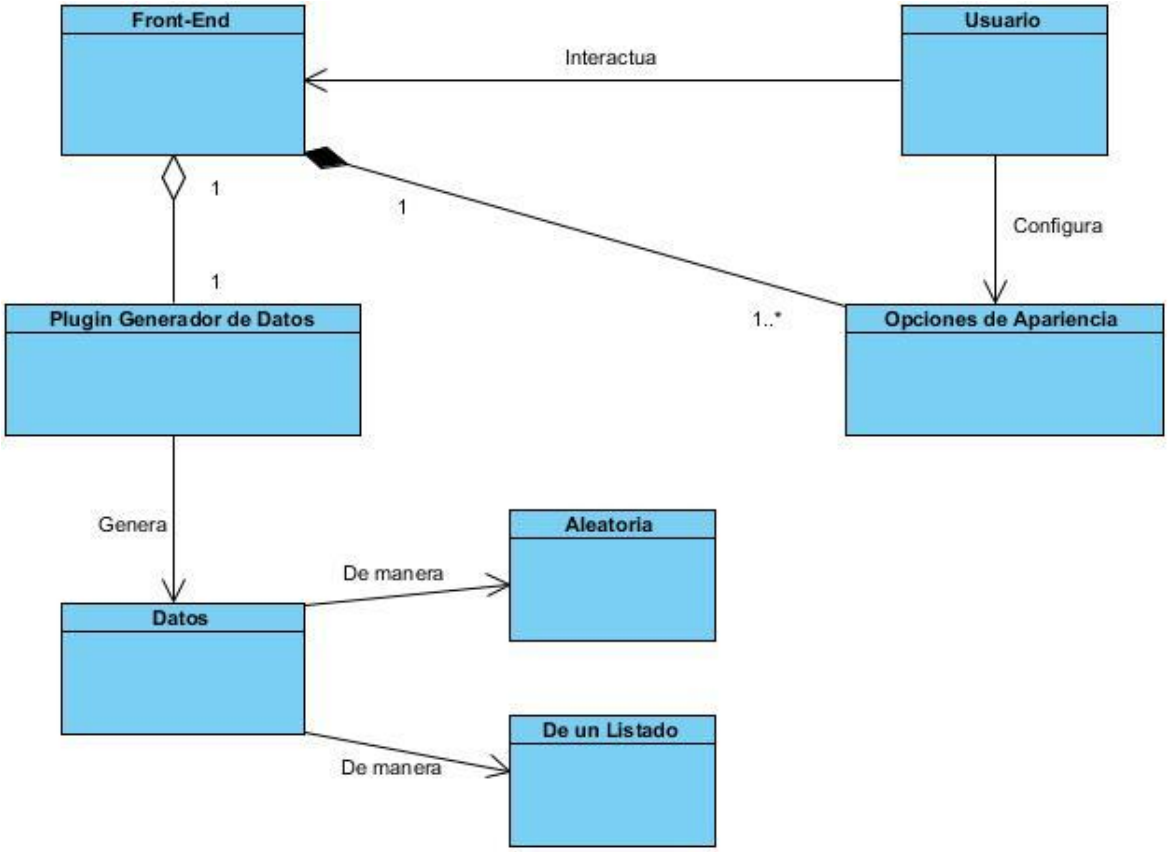


Figura 1: Modelo de Dominio

2.3 Historias de usuarios

Las Historias de Usuario (HU) describen funcionalidades que aportan por si misma información específica de gran importancia para el cliente.

Estas son una 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 HU es muy dinámico y flexible, en cualquier momento 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 (16).

Las HU son escritas por el cliente en un lenguaje fácil de entender, es decir sin hacer mucho énfasis en los detalles. Son usadas mayormente para estimar el tiempo de desarrollo de la aplicación que las

mismas describen. El tiempo de desarrollo ideal para una historia de usuario es entre una y tres semanas, si esta se pasa de las tres semanas hay que dividirla y formar una o más historias. Además se pueden descomponer en tareas de programación y asignarlas a los programadores para ser realizadas durante una iteración.

Fueron identificadas y documentadas 5 HU, las cuales se muestran a continuación:

Tabla 1: HU Obtener parámetros para la generación de datos

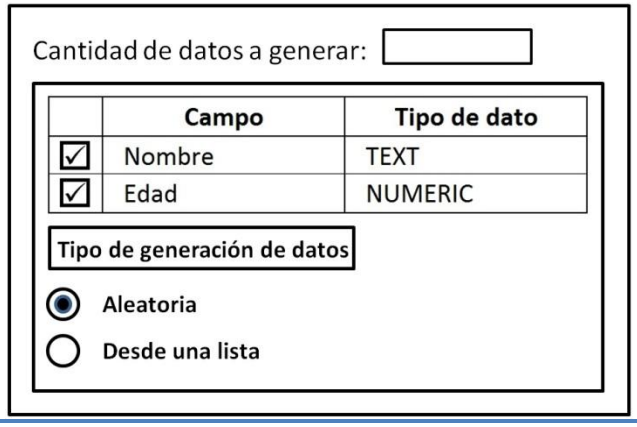
<table>
<tr><td colspan="3">Historia de Usuario</td></tr>
<tr><td>Número: 1</td><td colspan="2">Nombre de la Historia de Usuario: Obtener parámetros para la generación de datos.</td></tr>
<tr><td colspan="3">Cantidad de modificaciones a la Historia de Usuario: 1</td></tr>
<tr><td colspan="2">Usuario: usuario registrado en la base de datos</td><td>Iteración asignada: 1era iteración.</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: Muy Alto</td><td>Puntos reales: 1 semana</td></tr>
<tr><td colspan="3">Descripción: De cada tabla seleccionada el usuario deberá especificar qué cantidad de datos desea sean generados. Además deberá seleccionar para qué atributos se generarán datos. Para cada uno de estos atributos se especificará el tipo de generación de datos (aleatoria o desde una lista).</td></tr>
<tr><td colspan="3">Observaciones: Los atributos NOT NULL tendrán que estar marcados obligatoriamente para que se les aplique la generación de datos.</td></tr>
<tr><td colspan="3">Prototipo de interfaces:
 
</td></tr>
</table>

Tabla 2: HU Generar datos aleatorios

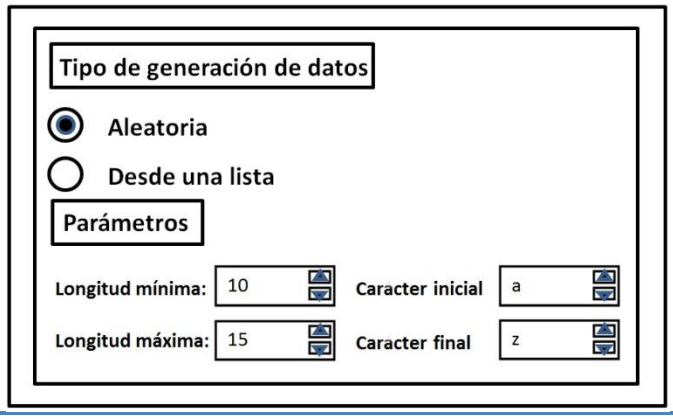
Historia de Usuario	
Número: 2	**Nombre de la Historia de Usuario:** Generar datos aleatorios.
Cantidad de modificaciones a la Historia de Usuario: 1	
Usuario: usuario registrado en la base de datos	**Iteración asignada:** 1era iteración
Prioridad en negocio: Muy Alta	**Puntos estimados:** 3,0
Riesgo en desarrollo: Muy Alto	**Puntos reales:** 3 semanas
Descripción: El usuario deberá especificar parámetros necesarios para cada atributo en dependencia del tipo de dato que este posea. Al realizar el proceso de generación de datos los cambios se guardarán en la base de datos a la cual se está conectado. Al concluir el proceso el sistema muestra un mensaje de éxito y el resultado se podrá visualizar en la herramienta HABD.	
Observaciones:	
Prototipo de interfaces: 	

Tabla 3: HU Generar datos desde una lista

Historia de Usuario	
Número: 3	**Nombre de la Historia de Usuario:** Generar datos desde una lista.
Cantidad de modificaciones a la Historia de Usuario: 1	

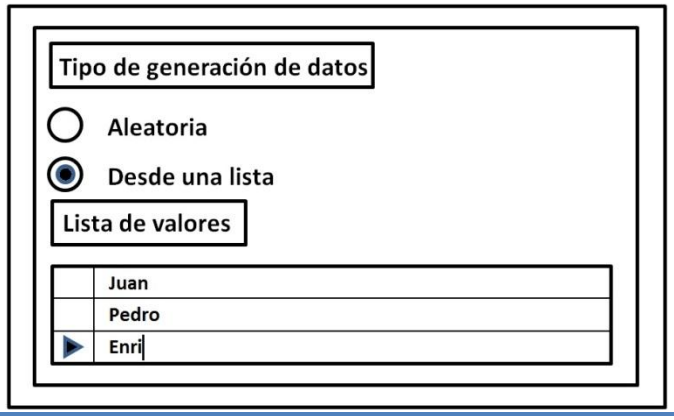
Usuario: usuario registrado en la base de datos	**Iteración asignada:** 1era iteración
Prioridad en negocio: Muy Alta	**Puntos estimados:** 3,0
Riesgo en desarrollo: Muy Alto	**Puntos reales:** 3 semanas
Descripción: El sistema brindará la posibilidad de adicionar los elementos que componen la lista que se usará para generar los datos. Al realizar el proceso de generación de datos los cambios se guardarán en la base de datos a la cual está conectado. Al concluir el proceso el sistema muestra un mensaje de éxito y el resultado se podrá visualizar en la herramienta HABD.	
Observaciones:	
Prototipo de interfaces: 	

Tabla 4: HU Mostrar campos de las tablas seleccionadas

	Historia de Usuario
Número: 4	**Nombre de la Historia de Usuario:** Mostrar campos de las tablas seleccionadas.
Cantidad de modificaciones a la Historia de Usuario: 1	
Usuario: usuario registrado en la base de datos	**Iteración asignada:** 2da iteración
Prioridad en negocio: Alta	**Puntos estimados:** 1,0
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Cuando el usuario selecciona la tabla sobre la cual desea generar los datos se	

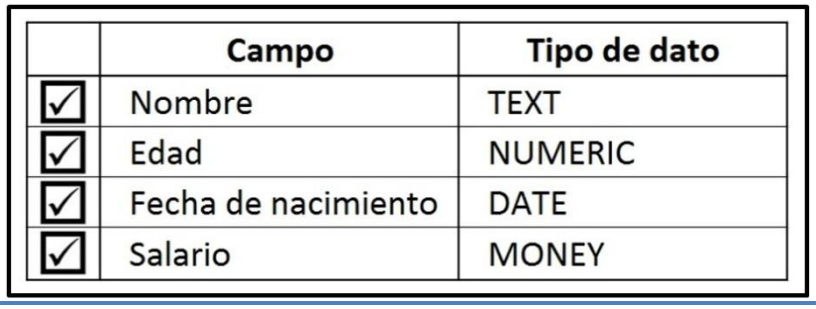
muestra un listado con el nombre y el tipo de dato de todos los atributos que contiene dicha tabla a los cuales se les puede realizar el proceso de generación.
Observaciones:
Prototipo de interfaces: 

Tabla 5: HU Generar ayuda

Historia de Usuario	
Número: 5	**Nombre de la Historia de Usuario:** Generar ayuda.
Cantidad de modificaciones a la Historia de Usuario: 1	
Usuario: usuario registrado en la base de datos	**Iteración asignada:** 2da iteración
Prioridad en negocio: Media	**Puntos estimados:** 1,0
Riesgo en desarrollo: Medio	**Puntos reales:** 1 semana
Descripción: El usuario debe presionar el botón de ayuda que se encuentra en la parte superior derecha de la interfaz, con ícono de color azul y un símbolo de signo de interrogación.	
Observaciones: El plugin tiene que estar integrado al HABD.	
Prototipo de interfaces: 	

2.4 Lista de Reserva del Producto

La Lista de Reserva del Producto (LRP) es uno de los artefactos generados por la metodología XP, el cual contiene el listado de las HU representadas anteriormente, ordenadas por prioridad de implementación: Muy Alta, Alta, Media y Baja. Los requisitos no funcionales están clasificados como de prioridad baja. La LRP puede ampliarse y modificarse a medida que se avanza en la implementación.

Tabla 6: Lista de reserva del producto

Ítem *	Descripción	Estimación	Estimado por
Prioridad: Muy Alta			
1	Obtener parámetros para la generación de datos.	1	Analista
2	Generar datos aleatorios.	3	Analista
3	Generar datos desde una lista.	3	Analista
Prioridad: Alta			
4	Mostrar campos de las tablas seleccionadas.	1	Analista
Prioridad: Media			
5	Generar ayuda	1	Analista
Prioridad: Baja			
1	**Apariencia o interfaz externa:** Interfaz intuitiva, amigable, organizada, con una navegabilidad flexible y de fácil comprensión, de forma que el objetivo del sistema para con los clientes, se pueda conseguir rápidamente. Debe estar diseñada para verse en cualquier resolución igual o inferior a 1024x768. El diseño gráfico debe estar acorde con las pautas de diseño de la Universidad.		
2	**Facilidad de uso:** Para utilizar el sistema es necesario poseer conocimientos elementales de computación, así como de bases de datos.		
3	**Soporte**: Responde a los requisitos de soporte de HABD los cuales se encuentran en el expediente de proyecto.		
4	**Software**: Sistema Operativo: Multiplataforma. Librerías QT. Servidor de Bases de datos: PostgreSQL 8.4.x o versiones superiores.		

5	**Hardware:** Se necesita 128 MB de memoria RAM como mínimo, 1GB de espacio libre en el disco duro para su instalación y el micro a 300 MHz.	
6	**Seguridad:** Responde a los requisitos de seguridad de HABD los cuales se encuentran en el expediente de proyecto.	

2.5 Tareas de la Ingeniería

Las tareas de la ingeniería (task card) describen las tareas que se realizan sobre el proyecto. Las mismas pueden ser de desarrollo, corrección y mejora. Estas están relacionadas directamente con una historia de usuario, aunque una HU puede ser desglosada en varias tareas. Además se especifica la fecha de inicio y fin de la tarea, el programador responsable de cumplirla y una breve descripción de la misma. A continuación se muestra en las tablas 7 y 8 las tareas de la ingeniería #1 y #2, pertenecientes a la historia de usuario #1, el resto se encuentra en el expediente de proyecto.

Tabla 7: TI Diseñar interfaz con los parámetros para la generación de datos

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario: 1**
Nombre Tarea: Diseñar interfaz con los parámetros para la generación de datos.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 0,4 semanas
Fecha Inicio: 12-03-2012	**Fecha Fin:** 13-03-2012
Programador Responsable: Abel Delgado Rodríguez	
Descripción: Esta interfaz permitirá introducir la cantidad de datos a generar por cada tabla. Además dará la posibilidad de escoger el tipo de generación (aleatoria o desde una lista) de cada atributo.	

Tabla 8: TI Obtener parámetros para la generación de datos

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario: 1**
Nombre Tarea: Obtener parámetros para la generación de datos.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 0,6 semanas
Fecha Inicio: 14-03-2012	**Fecha Fin:** 16-03-2012
Programador Responsable: Abel Delgado Rodríguez	
Descripción: Los parámetros que se obtendrán serán: la cantidad de datos a generar por tablas; y el tipo de generación (aleatoria o desde una lista).	

2.6 Plan de iteraciones

El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción.

Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: HU no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores (17).

Tabla 9: Plan de iteraciones

Release	Descripción de la iteración	Orden de la HU a implementar	Duración total
Iteración 1	En esta iteración se implementarán las HU 1, 2 y 3. Estas historias son de prioridad muy alta pues son las que mayor incidencia tienen en la funcionalidad del plugin. Por tal motivo son desarrolladas en esta iteración.	HU #1 HU #2 HU #3	7 semanas
Iteración 2	En la 2da iteración se desarrollarán las HU 4 y 5, las cuales son de menor importancia. Estas HU tienen prioridad Alta y Media respectivamente.	HU #4 HU #5	2 semanas

2.7 Modelo de diseño

La metodología XP hace especial énfasis en los diseños simples y claros, pues un diseño simple se implementa más rápidamente que uno complejo. El objetivo principal del diseño es especificar una solución que pueda ser fácilmente convertida a código fuente y construir una arquitectura fuerte simple y extensible.

2.7.1 Diagrama de Clases

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.

Este es un artefacto definido en la metodología RUP (Rational Unified Process) y que XP no posee, pero se decide incorporar a la investigación en aras de lograr una descripción más detallada y fácil de entender del sistema. A continuación se muestra el diagrama de clases perteneciente al plugin:

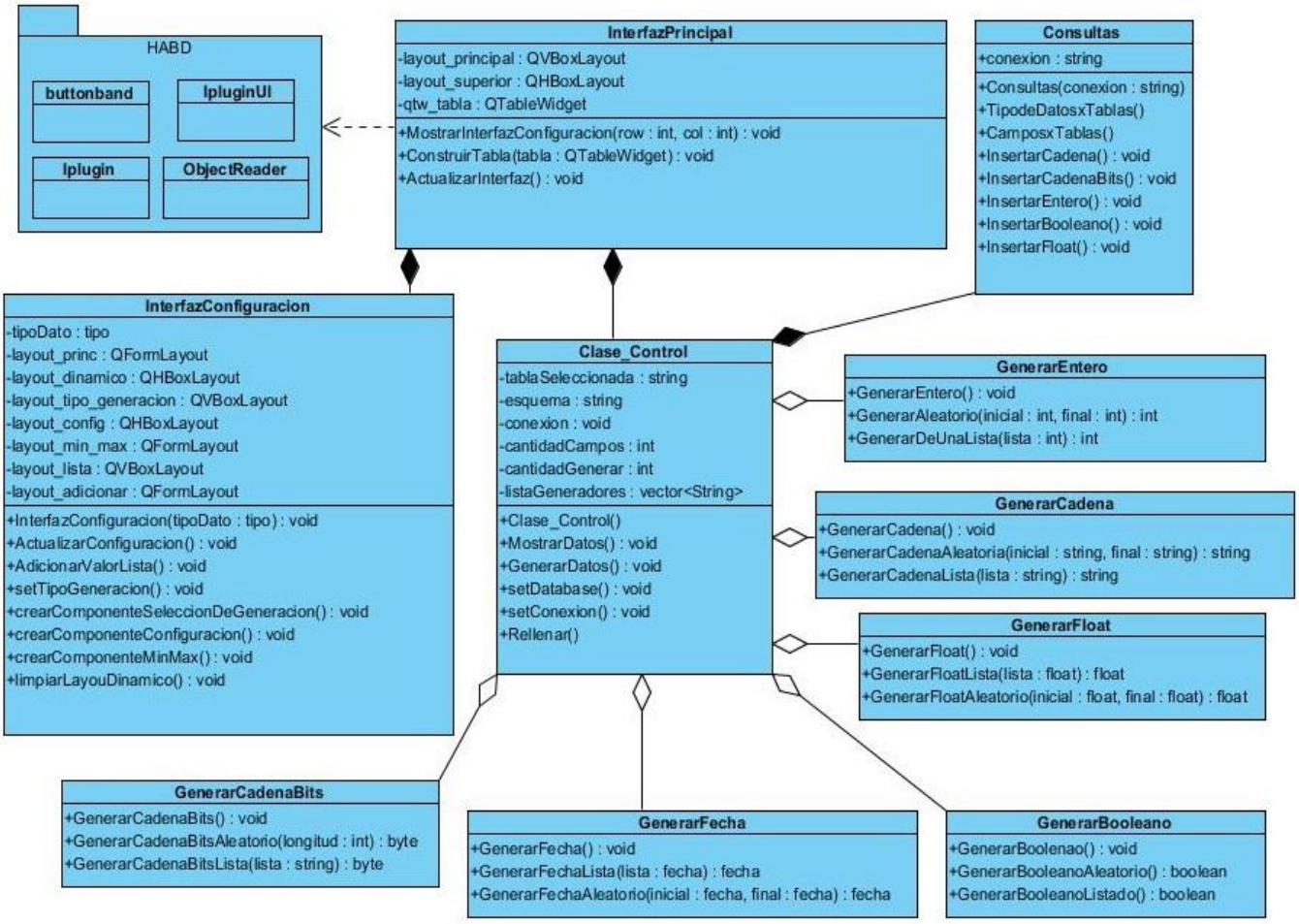


Figura 2: Diagrama de clases

2.7.2 Tarjetas Clase-Responsabilidades-Colaboración

El uso de las tarjetas C.R.C (Class-Responsabilities-Collaboration) permite al programador centrarse y apreciar el desarrollo orientado a objetos olvidándose de los malos hábitos de la programación procedural clásica.

Las tarjetas C.R.C representan objetos; la clase a la que pertenece el objeto se puede escribir en la parte de arriba de la tarjeta, en una columna a la izquierda se pueden escribir las responsabilidades u objetivos que debe cumplir el objeto y a la derecha, las clases que colaboran con cada responsabilidad (18).

Durante el proceso de diseño del plugin se elaboraron un total de 10 tarjetas CRC. A continuación se muestran algunas de las más significativas, el resto está plasmado en el expediente de proyecto de la investigación.

Tabla 10: Tarjeta CRC para la clase InterfazPrincipal.

Tarjeta CRC	
Clase: InterfazPrincipal	
Responsabilidades	**Colaboraciones**
Encargada de construir la interfaz del plugin. Construye la tabla con el nombre y el tipo de dato de cada atributo. Muestra la interfaz de configuración. Actualiza la interfaz.	*InterfazConfiguración* *HABD*

Tabla 11: Tarjeta CRC para la clase InterfazConfiguración.

Tarjeta CRC	
Clase: InterfazConfiguración	
Responsabilidades	**Colaboraciones**
Crea los componentes de configuración, selección de generación, y valores mínimos y máximos. Actualiza la interfaz de configuración y limpia el layout dinámico. Adiciona los valores insertados a la lista.	

Tabla 12: Tarjeta CRC para la clase Clase_Control.

Tarjeta CRC	
Clase: Clase_Control	
Responsabilidades	**Colaboraciones**
Encargada de controlar el sistema. Controla el proceso de generación de datos y su inserción en la base de datos.	*GenerarEntero* *GenerarCadena* *GenerarFloat* *GenerarBooleano* *GenerarFecha* *GenerarCadenaBits* *InterfazPrincipal* *Consultas*

Tabla 13: Tarjeta CRC para la clase GenerarCadena.

Tarjeta CRC	
Clase: GenerarCadena	
Responsabilidades	**Colaboraciones**
Genera cadenas de caracteres, tanto de forma aleatoria como a partir de una lista previamente introducida por el usuario.	

2.8 Patrón Arquitectónico

Un patrón de arquitectura de software es un esquema genérico probado para solucionar un problema particular, el cual es recurrente dentro de un cierto contexto. Este esquema se especifica describiendo los componentes, con sus responsabilidades y relaciones (19).

Patrón Modelo - Vista - Controlador

El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes: el Modelo, las Vistas y el Controlador (20).

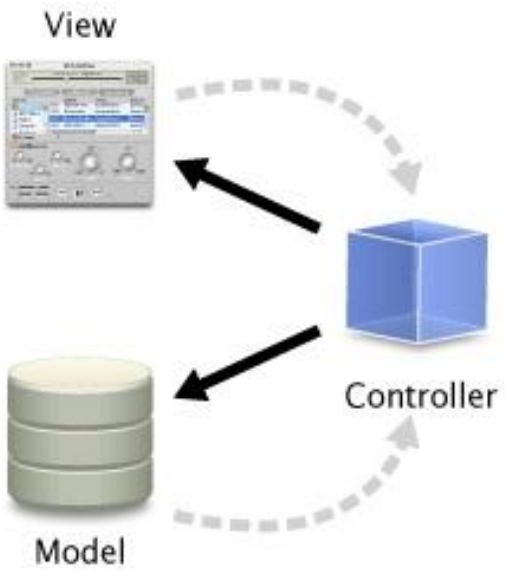


Figura 3: Patrón Modelo-Vista-Controlador

Modelo

El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar, sin tomar en cuenta la forma en la que esa información va a ser mostrada ni los mecanismos que hacen que esos datos estén dentro del modelo, es decir, sin tener relación con ninguna otra entidad dentro de la aplicación.

Vista

Las vistas son el conjunto de clases que se encargan de mostrar al usuario la información contenida en el modelo. Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo.

Controlador

El controlador es un objeto que se encarga de dirigir el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él. A partir de estos mensajes, el controlador se encarga de modificar el modelo o de abrir y cerrar vistas. El controlador tiene acceso al modelo y a las vistas, pero las vistas y el modelo no conocen de la existencia del controlador (21).

A continuación se muestra (Figura 4) como se evidencia el patrón Modelo-Vista-Controlador en el plugin:

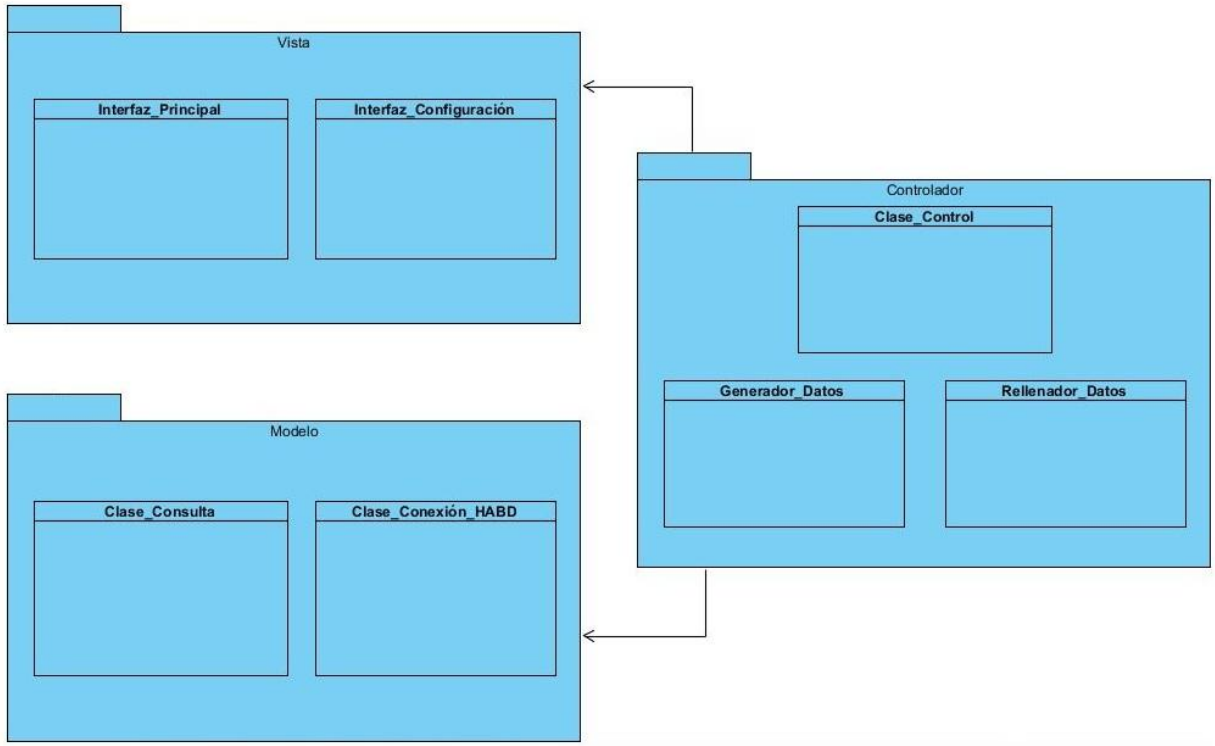


Figura 4: Modelo-Vista-Controlador

2.9 Patrones de diseño

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares.

Se debe tener presente los siguientes elementos de un patrón: su nombre, el problema (cuándo aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios) (22).

Patrones Grasp

GRASP es el acrónimo de General Responsibility Assignment Software Patterns (patrones generales de software para asignar responsabilidades). 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 (23).

Patrones GRASP que se utilizaron para el desarrollo del plugin:

Bajo acoplamiento: Se deben asignar las responsabilidades de forma tal que cada clase se comunique con el menor número de clases, minimizando el nivel de dependencia.

Alta cohesión: Asignar a las clases responsabilidades que trabajen sobre una misma área de la aplicación y que no tengan mucha complejidad, evitando así que una clase sea la única responsable de muchas tareas en áreas funcionales muy heterogéneas.

Ambos patrones se pueden apreciar en el plugin en las clases Generadoras de datos, donde existe una clase que genera cada tipo de dato (GenerarFecha, GenerarCadena, GenerarEntero, GenerarFloat y GenerarBoolean) aplicando el patrón alta cohesión. Dichas clases no se relacionan entre sí, sino que se relacionan directamente con la clase Clase_Control, poniendo en práctica el patrón bajo acoplamiento.

Experto: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para realizar la tarea. Este patrón se evidencia en las clases generadoras de datos las cuales son las encargadas de generar cada tipo de dato.

Creador: Este patrón ayuda a responder la pregunta de qué clase debe ser la responsable de crear nuevos objetos (instancias de alguna clase). En el plugin se pone de manifiesto este patrón, por ejemplo cuando la clase Interfaz_Principal crea una instancia de InterfazConfiguración.
interfaz_config = new InterfazConfiguración (rellenador->getGenerador(row), tipo);

Controlador: Asignar 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. En el plugin se evidencia este patrón de diseño en la clase Clase_Control, la cual se encarga de controlar el flujo de información dentro de la misma.

2.10 Estándares de Codificación

La metodología XP enfatiza la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación (del equipo, de la organización u otros estándares reconocidos para los lenguajes de programación utilizados). Los estándares de programación mantienen el código legible para los miembros del equipo, facilitando los cambios.

El estándar de codificación que se utiliza es el definido anteriormente en el desarrollo de la herramienta HABD, se utiliza para lograr un mejor entendimiento en aquellas personas que deseen trabajar con el plugin, además de permitir optimizar el código y tenerlo más organizado. El mismo se describe a continuación.

Identación

La unidad de identado es de 3 espacios. El uso de la tabulación debe ser evitado porque (tal como se escribía en el siglo pasado) no existe un estándar que determine con precisión el ancho que va a producir la tabulación.

Longitud de la Línea

Evitar líneas con más de 80 caracteres. Cuando una sentencia sobrepase una línea simple del editor, esta deber ser separada en otras. Realice la separación después de un operador, preferentemente luego de una coma. Realizar la ruptura después de un operador disminuye la probabilidad de que al realizar un copiado del código se cometa un error por olvido de la inserción del punto y coma. La siguiente línea debe ser identada con 5 espacios.

```
QString consulta = "INSERT INTO "+table_schema+"."+table_name+" VALUES ( ";
```

Comentarios

Es conveniente dejar información que pueda ser leída tiempo después por personas (posiblemente usted mismo) que necesitan entender que fue lo que se hizo en el fragmento de código. Los comentarios deben ser escritos correctamente y claros.
Generalmente deben usarse comentarios de una sola línea. Reserve los comentarios de bloques para la documentación formal o para comentar porciones de código.

```
cantidad_generada = cont;//No incluye valores nulls
```

Declaración de variables

Cada variable debe ser declarada en una línea y comentada. También deben aparecer ordenadas alfabéticamente:

```
int longitud_maxima;//longitud máxima de la cadena
int longitud_minima;//longitud mínima de la cadena
int max;//caracter final del rango de caracteres
int min;//caracter inicial del rango de caracteres
```

El nombre de las variables debe comenzar con letras minúsculas y cada palabra relevante por la que esté compuesta debe ser con letra mayúscula, o pueden separarse por el caracter "_".

```
cantidad_generada = cont;//No incluye valores nulls
```

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.

```
long  Generador_Int::genNumAleatorio(long min, long max){
    unsigned  long long rango = abs((long long)max - (long long)min);
    unsigned  long long aleatorio = rand() % (rango + 1);
    unsigned long long estiramiento = rango / RAND_MAX;
    if(estiramiento != 0){
        aleatorio*=estiramiento;
    }
    return std::min(min,max) + aleatorio;
}
```

Identificadores

Los identificadores pueden estar formados por cualesquiera de las 26 letras minúsculas o mayúsculas (A .. Z, a .. z), los 10 dígitos (0 .. 9) y el carácter subrayado "_". Debe evitarse el uso de caracteres internacionales (ej: ñ, ü) porque no siempre pueden ser leídos o entendidos correctamente en todos los lugares. No se debe usar el símbolo dólar "$" o la barra invertida "\" en los identificadores.

```
cantidad_generada = cont;//No incluye valores nulls
```

Sentencias Simples

Cada línea debe contener como máximo una sentencia. Debe poner un punto y coma ";" al final de cada sentencia simple. Tenga en cuenta que una sentencia de asignación puede resultar en la asignación de una función o de un objeto como literal y en todos los casos como sentencia de asignación debe estar finalizada con un punto y coma.

```
layout_principal->addRow(layout_dinamico);
```

Sentencias Compuestas

Las sentencias compuestas son aquellas sentencias que contienen una lista de sentencias encerradas entre llaves:

- ✓ Las sentencias encerradas deben ser identada a 4 espacios.
- ✓ La llave que inicia la lista de sentencias debe estar al final de línea de la sentencia compuesta.
- ✓ La llave que termina la lista de sentencias debe estar al comienzo de una línea y guardar la misma identación que la sentencia compuesta en correspondencia con la llave que inicia.
- ✓ Las llaves siempre serán usadas para listar todas las sentencias, aunque se trate de una sola, cuando son parte de una estructura de control como if o for. Ello facilita agregar nuevas sentencias sin la introducción accidental de errores.

```
void Rellenador::Inicializar() {
    vector<Generador_Dato*>::iterator it = generadores.begin();
    for(;it != generadores.end(); it++) {
        *it = 0;
    }
}
```

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.

```
return min_bd;
```

Sentencia if

```
if(tipo_gener == ALEATORIO) {
    valor = Util::genCadenaAleatoria(longitud_minima,longitud_maxima,min,max);
}
else {
    assert(lista.size() > 0);
    int indice_aleatorio = Util::genNumAleatorio(0,lista.size()-1);
    valor = lista[indice_aleatorio];
}
```

Estructuras repetitivas

```
for(int i = inicio+1; i<fin; i++) {
    llave_foranea+=definicion[i];
}
```

```
while(longitud_aleatoria-- > 0) {
    int bit = genNumAleatorio(0,1);
    cadena_bits += A_Cadena(bit);
}
```

```
do {
    if(tipo_gener == ALEATORIO) {
        valor = Util::genCadenaAleatoria(longitud_minima,longitud_maxima,min,max);
    }
    else {
        assert(lista.size() > 0);
        int indice_aleatorio = Util::genNumAleatorio(0,lista.size()-1);
        valor = lista[indice_aleatorio];
    }

}
while(find( valores_bd_.begin(), valores_bd_.end(), Dato(valor)) != valores_bd_.end());
```

Espacios en blanco

- ✓ No se debe utilizar un espacio en blanco entre el identificador de una función y el paréntesis que abre a la lista de parámetros. Ello 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 ",".

```
static QDate genFechaAleatoria(QDate fecha_inicial, QDate fecha_final);
```

2.11 Interfaz de la aplicación

En la Figura 5 se muestra la interfaz principal con la que interactúa el usuario una vez que se encuentra el plugin integrado a la herramienta HABD. Específicamente se visualiza como el usuario una vez seleccionada una tabla de la base de datos, puede especificar la cantidad de datos que desea generar, el tipo de generación (en este caso generación aleatoria) y las opciones de generación que dependen del tipo de dato del atributo que tenga seleccionado.

En caso de que el usuario seleccione tipo de generación de una lista, se mostrará la interfaz correspondiente a este tipo de generación, ver Figura 6.

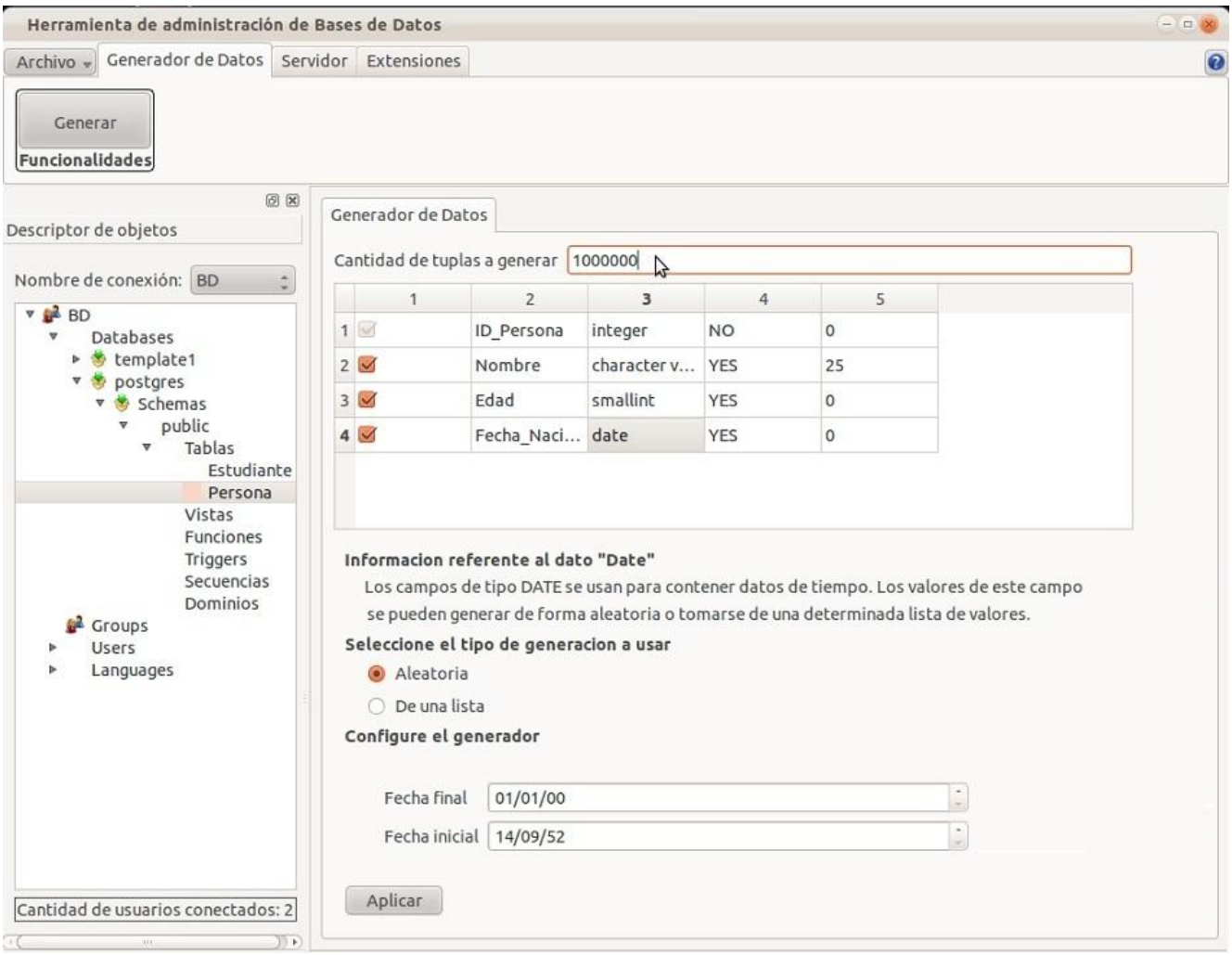


Figura 5: Interfaz de la aplicación (Generar Aleatorio)

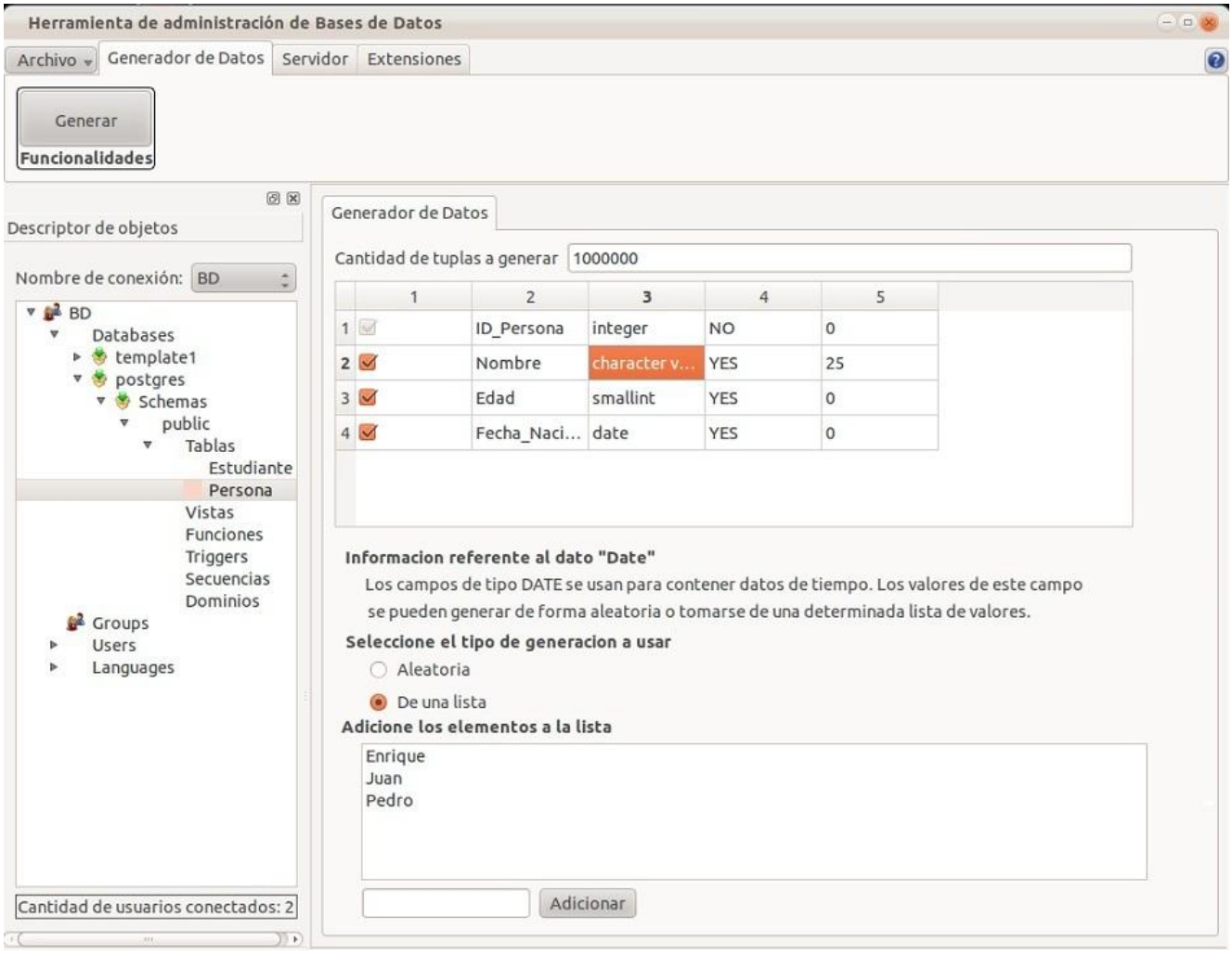


Figura 6: Interfaz de la aplicación (Generar desde una lista)

Conclusiones del capítulo

En el capítulo se analizaron los procesos fundamentales que están relacionados con la solución informática que se debe desarrollar. Se describió la solución propuesta para el problema de la investigación. Además se identificaron 5 HU y los requisitos no funcionales elementos fundamentales para la creación de un plugin con calidad. Fueron generados los artefactos que propone la metodología de desarrollo: la lista de reserva del producto, las tareas de la ingeniería y el plan de iteraciones, así como otros artefactos que facilitan el entendimiento del desarrollo de la aplicación. El uso de los patrones de diseño y de arquitectura; y de las tarjetas CRC, permiten que la implementación sea sencilla y correcta.

CAPÍTULO 3: Implementación y pruebas

Introducción

Una vez desarrolladas las funcionalidades del plugin es necesario realizar pruebas a la misma, debido a que su correcto funcionamiento no quiere decir que esta no contenga errores. Es por ello que este capítulo se centra en las pruebas, en validar cada una de las soluciones logradas.

3.1 Descripción de los principales métodos implementados

A continuación se muestran algunos ejemplos del código de los principales métodos implementados en el desarrollo del plugin con una breve descripción de la funcionalidad de los mismos.

Método Rellenar

Se encuentra en la clase Clase_Control.

```
void Clase_Control::Rellenar(){
    int c = cantidad_a_generar;
    while(c-- > 0){
        QString consulta = "INSERT INTO "+table_schema+"."+table_name+" VALUES ( ";
        vector<Generador_Dato*>::iterator it = generadores.begin();
        vector<Generador_Dato*>::iterator aux;
        QString valores;
        for(;it != generadores.end();it++){
            Generador_Dato* g = *it;
            aux = it;
            aux++;
            if(g != 0){
                Dato d = g->generar_();
                if(d.getTipo() != ENTERO){
                    valores+="'";
                }
                valores += QString::fromStdString(d.aCadena());
                if(d.getTipo() != ENTERO){
                    valores+="'";
                }
                if(aux != generadores.end()){
                    valores+=",";
                }
            }
            else{
                if(aux != generadores.end()){
                    valores+=",";
                }
            }
        }
        consulta+=valores;
        consulta+=");";
        QSqlQuery consultador(consulta, con);
    }
}
```

Figura 7: Ejemplo de código

Este método se encarga insertar en la base de datos. Para ello recorrer el vector de generadores haciendo uso de iteradores implementados en las STL. Llama en cada objeto el método generar para obtener el valor de los atributos de la consulta "INSERT" que se crea. Si un generador no fue inicializado el método asigna en el atributo correspondiente de la consulta la palabra reservada “Null”.

Método genNumAleatorio

Se encuentra en la clase Generador_Int

```
long  Generador_Int::genNumAleatorio(long min, long max){
    unsigned  long long rango = abs((long long)max - (long long)min);
    unsigned  long long aleatorio = rand() % (rango + 1);
    unsigned long long estiramiento = rango / RAND_MAX;
    if(estiramiento != 0){
        aleatorio*=estiramiento;
    }
    return std::min(min,max) + aleatorio;
}
```

Figura 8: Ejemplo de código

Este método genera un número aleatorio entre un valor mínimo y máximo. Para definir el valor aleatorio se usó el tipo long con el objetivo de poder crear valores aleatorios de un mayor rango. Este método hace uso del método rand de la biblioteca Cstdlib de C para crear un valor aleatorio. Para lograr que este método implementado en C se adaptase al rango de valores requerido se implementó el "estiramiento". Este último permite generar enteros de 32 bits.

Método genCadenaAleatoria

Se encuentra en la clase Generador_Cadena

```
string Generador_Cadena::genCadenaAleatoria(int long_min, int long_max, char min, char max){
    int longitud_aleatoria = Generador_Int::genNumAleatorio(long_min, long_max);
    string cadena = "";
    while(longitud_aleatoria-- > 0){
        char c = Generador_Int::genNumAleatorio(min, max);
        cadena+=c;
    }
    return cadena;
}
```

Figura 9: Ejemplo de código

Este método se encarga de generar una cadena aleatoria entre una longitud mínima y máxima con caracteres que oscilan entre un caracter inicial y un caracter final. Para definir tanto la longitud como los valores internos de la cadena de forma aleatoria se hace uso del método estático genNumAleatorio de la clase Generador_Int**.**

Método genFloatAleatorio

El método se encuentra en la clase Generador_Float

```
float Generador_Float::genFloatAleatorio(float min, float max){
    if(Generador_Int::genNumAleatorio(0,1) == 0){
        return -1*(rand()*(max-min)/RAND_MAX + min);
    }
    return rand()*(max-min)/RAND_MAX + min;
}
```

Figura 10: Ejemplo de código

Este método se encarga de generar un número flotante de 32 bits que se encuentra entre un mínimo y un máximo. El mismo hace uso del método estático genNumAleatorio de la clase Generador_Int para definir el signo del número flotante.

Método genFechaAleatoria

El método se encuentra en la clase Generador_Fecha

```
QDate Generador_Fecha::genFechaAleatoria(QDate fecha_inicial, QDate fecha_final){
    QDate fecha_aleatoria = fecha_inicial;
    int rango = fecha_inicial.daysTo(fecha_final);
    fecha_aleatoria = fecha_aleatoria.addDays(Generador_Int::genNumAleatorio(0,rango));
    return fecha_aleatoria;
}
```

Este método se encarga de generar una fecha aleatoria que se encuentra entre una fecha inicial y una fecha final. Para abstraer el comportamiento de una fecha se hace uso de la clase QDate del framework Qt. Para generar la fecha aleatoria se hace uso del método estático genNumAleatorio de la clase Generador_Int.

3.2 Enfoques de diseños de pruebas

En cada una de las etapas de desarrollo de un software las pruebas son de vital importancia, ya que mediante estas es posible controlar que los productos cumplan los requisitos de operatividad, y se puede garantizar su calidad. Las pruebas constituyen una actividad en la cual un sistema es ejecutado bajo condiciones específicas, los resultados son observados y registrados a fin de determinar si los errores ocurren cuando no tendrían que ocurrir, realizando a partir de aquí una evaluación del estado del sistema.

Existen tres enfoques de diseños de casos de pruebas, los cuáles son (24):

- ✓ Enfoque estructural o de caja blanca: esta consiste en centrarse en la estructura interna (implementación) del programa para elegir los casos de prueba.
- ✓ Enfoque funcional o de caja negra: consiste en estudiar la especificación de las funciones, la entrada y la salida para derivar los casos.
- ✓ Enfoque aleatorio: consiste en utilizar modelos (en muchas ocasiones estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba.

3.2.1 Método seleccionado

Uno de los pilares de la metodología XP es el uso de pruebas para comprobar el funcionamiento de los códigos que se van implementando. Esta metodología propone la realización de dos tipos de pruebas: unitarias y de aceptación.

Las pruebas unitarias son realizadas a pequeñas porciones de código, generalmente de la ejecución de estas pruebas se encargan los desarrolladores, ya que estos tienen un mayor conocimiento del código.

Las pruebas de aceptación son creadas en base a las HU, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada. (25) Estas pruebas son puramente funcionales.

Se denominan pruebas funcionales o Functional Testing, a las pruebas de software que tienen por objetivo probar que los sistemas desarrollados, cumplan con las funciones específicas para los cuales han sido creados. Es común que este tipo de pruebas sean desarrolladas por analistas de pruebas con apoyo de algunos usuarios finales, esta suele ser la última etapa de pruebas al software antes de proceder a su despliegue (26).

A este tipo de pruebas se les denomina también pruebas de comportamiento o pruebas de caja negra, ya que los probadores o analistas de pruebas, no enfocan su atención a como se generan las respuestas del sistema, básicamente el enfoque de este tipo de prueba se basa en el análisis de los datos de entrada y en los de salida, esto generalmente se define en los casos de prueba preparados antes del inicio de las pruebas (26).

Para la solución se propone realizar las pruebas de aceptación mediante el método de caja negra, en resumen en las pruebas de caja negra, los casos de prueba pretenden demostrar que las funciones del

software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta.

3.3 Casos de pruebas basadas en HU

El software se probó con el diseño de cinco casos de pruebas, los cuales se corresponden directamente con cada una de las HU identificadas:

Condición de ejecución: El plugin tiene que estar integrado al HABD y debe existir una conexión a un servidor de base de datos.

A continuación se muestra la descripción de las variables asociadas al caso de prueba Obtener Parámetros:

Tabla 14: Descripción de variables del caso de prueba obtener parámetros

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
Variable 1	Cantidad de datos	Campo de entrada (Textfield)	No	Es la cantidad de tuplas que se generarán y se insertarán en la tabla.
Variable 2	Atributos	Campo de selección (Checkbox)	No	Son los atributos que pueden ser generados.
Variable 3	Tipo de generación	Campo de selección (Radiobutton)	No	Las dos formas en que se pueden generar los datos.

Tabla 15: Caso de prueba Obtener parámetros

Escenario	Descripción	Variable 1	Variable 2	Variable 3	Respuesta del sistema	Flujo central
EC 1.1 Obtienen los datos para la generación con valores correctos.	Se obtienen los valores correctamen-te	V	V	V	Se recogen los datos satisfactoria-mente.	El usuario debe: 1- Introducir un número entero positivo. 2- Seleccionar uno o más campos de la tabla para generar. 3- Seleccionar un tipo de generación.
		1000	Se seleccionan uno o más campos	Se selecciona: Aleatoria		
EC 1.2 Obtienen los datos para la generación con valores incorrectos.		I	V	V	El sistema muestra un mensaje de error: "Debe especificar una cantidad de datos a generar"	
		No se introduce ningún valor	Se seleccionan uno o más campos	Se selecciona: Aleatoria		
		V	I	V	El sistema muestra un mensaje de error: "Debe seleccionar al menos un campo".	
		1000	No se selecciona ningún campo	Se selecciona: Aleatoria		

El resto de los casos de prueba con sus respectivas descripciones de las variables se encuentran en el expediente de proyecto.

Los resultados no satisfactorios obtenidos de las pruebas realizadas pasaron a ser no conformidades y se recogieron en el registro de defectos y dificultades detectados. En la tabla siguiente se muestran las no conformidades que se encontraron, además de la fecha en la que se detectaron y se les dio solución.

Tabla 16: No conformidades

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Clasificación	Estado NC	Respuesta Equipo Desarrollo
Aplicación	1	Al integrar el plugin a HABD, la interfaz no aparece completa en monitores con resoluciones pequeñas.	Interfaz Principal	Prueba	Significativa	28/05/2012 PD 30/05/2012 RA	Se rediseñó la interfaz de usuario logrando una mejor distribución de los componentes visuales.
Aplicación	2	Al desmarcar un campo para que el mismo no se genere se muestra el componente visual de configuración del mismo.	Interfaz de configuración	Prueba	Significativa	01/06/2012 PD 02/06/2012 RA	Se corrigió la acción correspon-diente al evento desmarcar checkbox.
Aplicación	3	En la ventana principal del plugin las letras acentuadas salen con errores de codificación.	Interfaz Principal	Prueba	Significativa	14/06/2012 PD 15/06/2012 RA	Se agregó un código a la aplicación que corrige el error de los caracters
Aplicación	4	En el EC 1.2 Generar los datos aleatorios con valores incorrectos no se muestra ningún mensaje cuando el valor de la variable 1 se deja en blanco.	Interfaz Principal	Prueba	Significativa	14/06/2012 PD 15/06/2012 RA	Se corrigió el error y se muestra el mensaje de error correctamen-te.

Conclusiones del capítulo

La realización de las pruebas de caja negra permitió detectar, documentar y solucionar los errores existentes en el sistema implementado. Concluir el período de pruebas permitió obtener una aplicación que responde correctamente, cumpliendo en su totalidad con los objetivos planteados para el plugin de generación de datos ficticios.

CONCLUSIONES GENERALES

Al finalizar la investigación se puede concluir que:

- ✓ El proceso de desarrollo del plugin estuvo guiado por la metodología de desarrollo XP, utilizando C++ como lenguaje de programación y Qt como framework para la implementación del plugin, apoyados en el potente y completo IDE Qt Creator.
- ✓ Se implementó un plugin para la generación de datos ficticios, el cual ayudará en el proceso de pruebas a bases de datos PostgreSQL.
- ✓ Las pruebas de aceptación aplicadas para validar las funcionalidades del plugin implementado, demostraron que el sistema cumple satisfactoriamente con los requisitos que garantizan su correcto funcionamiento.

Como resultado de la investigación se obtuvo un plugin para la herramienta de administración de bases de datos HABD que permite la generación de datos ficticios sobre las tablas de datos en PostgreSQL.

RECOMENDACIONES

Luego de haber logrado los objetivos que se trazaron en este trabajo y como la aplicación se encuentra en su primera versión, se recomienda:

- ✓ Añadir al plugin la funcionalidad de generar datos ficticios; ello permitirá al usuario seleccionar mediante una consulta SQL los datos a generar.
- ✓ Permitir que el plugin genere otros tipos de datos de los que soporta el sistema gestor de base de datos PostgreSQL: network, enumerativo, XML y arreglo.

REFERENCIAS BIBLIOGRÁFICAS

1. **El Glosario IEEE de Ingeniería del Software.** Connexions. *Connexions.* [En línea] [Citado el: 26 de marzo de 2012.] http://cnx.org/content/m17423/latest/.

2. **Juan Carlos Casamayor, Matilde Celma, Laura Mota.** 1999.

3. **Date, Christopher J.** *Introducción a los Sistemas de Bases de Datos.* Séptima Edición. 2001.

4. **Alvarez, Sara.** Desarrollo Web. *Desarrollo Web.* [En línea] [Citado el: 26 de marzo de 2012.] http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.

5. PostgreSQL. *PostgreSQL.* [En línea] [Citado el: 30 de marzo de 2012.] http://www.postgresql.org.es/sobre_postgresql.

6. **Harvey M.Deitel, P.J.Deitel.** *Como programar en C/C++ y Java.*

7. Softpedia. *Softpedia.* [En línea] [Citado el: 2 de abril de 2012.] http://www.softpedia.es/programa-EMS-Data-Generator-for-PostgreSQL-47351.html.

8. Datanamic. *Tools for database developers.* [En línea] [Citado el: 31 de mayo de 2012.] http://www.datanamic.com/datagenerator/index.html.

9. EcuRed. *EcuRed.* [En línea] [Citado el: 5 de abril de 2012.] http://www.ecured.cu/index.php/EXtreme_Programming.

10. **Rumbaugh, James y Booch, Grady y Jacobson, Ivar.** *El lenguaje unificado de modelado.Manual de Referencia.* 2000.

11. **Visual Paradigm.** Visual Paradigm International. [En línea] [Citado el: 18 de abril de 2012.] http://www.visual-paradigm.com/aboutus/10reasons.jsp.

12. **Nokia.** Conversations by Nokia. [En línea] 10 de marzo de 2011. [Citado el: 27 de abril de 2012.] http://conversations.nokia.com/2011/03/10/reasons-to-get-stuck-on-qt-a-bakers-dozen/.

13. Mi tecnológico. [En línea] [Citado el: 25 de abril de 2012.] http://www.mitecnologico.com/Main/DefinicionDeLenguajeDeProgramacion.

14. **Jalón, Javier García de.** *Aprenda C++ como si estuviera en primero.*

15. cplusplus.com. [En línea] 2011. [Citado el: 26 de abril de 2012.] http://www.cplusplus.com/info/description/.

16. **Addison-Wesley.** *Extreme Programming Installed.* 2001.

17. —. *Extreme Programming Explored.* 2002.

18. programacionextrema.tripod.com. *programacionextrema.tripod.com.* [En línea] [Citado el: 16 de mayo de 2012.] http://programacionextrema.tripod.com/fases.htm.

19. **Ochoa, Sergio.** *Introducción a los Patrones.* 2005.

20. **Burbeck, Steve.** [En línea] [Citado el: 22 de mayo de 2012.] http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

21. **Pantoja, Ernesto Bascón.** *El patrón de diseño Modelo-Vista-Controlador.* 2004.

22. MSDN Magazine. *MSDN Magazine.* [En línea] [Citado el: 3 de mayo de 2012.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.

23. **Mora, Roberto Canales.** *Tutorial: Patrones Grasp.* 2003.

24. Alarcos. *Alarcos.* [En línea] [Citado el: 8 de mayo de 2012.] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf.

25. **Joskowicz, José.** *Reglas y Prácticas en eXtreme Programing.* 2008.

26. CalidadySoftware.com. *Quality Assurance & Software Testing.* [En línea] 2009. [Citado el: 11 de mayo de 2012.] http://www.calidadysoftware.com/testing/pruebas_funcionales.php.

BIBLIOGRAFÍA

1. **El Glosario IEEE de Ingeniería del Software.** Connexions. *Connexions.* [En línea] [Citado el: 26 de marzo de 2012.] http://cnx.org/content/m17423/latest/.

2. **Juan Carlos Casamayor, Matilde Celma, Laura Mota.** 1999.

3. **Date, Christopher J.** *Introducción a los Sistemas de Bases de Datos.* Séptima Edición. 2001.

4. **Alvarez, Sara.** Desarrollo Web. *Desarrollo Web.* [En línea] [Citado el: 26 de marzo de 2012.] http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.

5. PostgreSQL. *PostgreSQL.* [En línea] [Citado el: 30 de marzo de 2012.] http://www.postgresql.org.es/sobre_postgresql.

6. **Harvey M.Deitel, P.J.Deitel.** *Como programar en C/C++ y Java.*

7. Softpedia. *Softpedia.* [En línea] [Citado el: 2 de abril de 2012.] http://www.softpedia.es/programa-EMS-Data-Generator-for-PostgreSQL-47351.html.

8. Datanamic. *Tools for database developers.* [En línea] [Citado el: 31 de mayo de 2012.] http://www.datanamic.com/datagenerator/index.html.

9. EcuRed. *EcuRed.* [En línea] [Citado el: 5 de abril de 2012.] http://www.ecured.cu/index.php/EXtreme_Programming.

10. **Rumbaugh, James y Booch, Grady y Jacobson, Ivar.** *El lenguaje unificado de modelado.Manual de Referencia.* 2000.

11. **Visual Paradigm.** Visual Paradigm International. [En línea] [Citado el: 18 de abril de 2012.] http://www.visual-paradigm.com/aboutus/10reasons.jsp.

12. **Nokia.** Conversations by Nokia. [En línea] 10 de marzo de 2011. [Citado el: 27 de abril de 2012.] http://conversations.nokia.com/2011/03/10/reasons-to-get-stuck-on-qt-a-bakers-dozen/.

13. Mi tecnológico. [En línea] [Citado el: 25 de abril de 2012.] http://www.mitecnologico.com/Main/DefinicionDeLenguajeDeProgramacion.

14. **Jalón, Javier García de.** *Aprenda C++ como si estuviera en primero.*

15. cplusplus.com. [En línea] 2011. [Citado el: 26 de abril de 2012.] http://www.cplusplus.com/info/description/.

16. **Addison-Wesley.** *Extreme Programming Installed.* 2001.

17. —. *Extreme Programming Explored.* 2002.

18. programacionextrema.tripod.com. *programacionextrema.tripod.com.* [En línea] [Citado el: 16 de mayo de 2012.] http://programacionextrema.tripod.com/fases.htm.

19. **Ochoa, Sergio.** *Introducción a los Patrones.* 2005.

20. **Burbeck, Steve.** [En línea] [Citado el: 22 de mayo de 2012.] http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

21. **Pantoja, Ernesto Bascón.** *El patrón de diseño Modelo-Vista-Controlador.* 2004.

22. MSDN Magazine. *MSDN Magazine.* [En línea] [Citado el: 3 de mayo de 2012.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.

23. **Mora, Roberto Canales.** *Tutorial: Patrones Grasp.* 2003.

24. Alarcos. *Alarcos.* [En línea] [Citado el: 8 de mayo de 2012.] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf.

25. **Joskowicz, José.** *Reglas y Prácticas en eXtreme Programing.* 2008.

26. CalidadySoftware.com. *Quality Assurance & Software Testing.* [En línea] 2009. [Citado el: 11 de mayo de 2012.] http://www.calidadysoftware.com/testing/pruebas_funcionales.php.

27. **León, Anthony R. Sotolongo.** *Compendio de consultas útiles al catálogo de postgreSQL.* La Habana, Cuba : s.n., 2011.

28. **The PostgreSQL Global Development Group.** *PostgreSQL 9.1.0 Documentation.* California : s.n., 1996-2011.

29. Comunidad Técnica Cubana de PostgreSQL. *Comunidad Técnica Cubana de PostgreSQL.* [En línea] http://postgresql.uci.cu/.

30. PostgreSQL-es. *Portal en español sobre PostgreSQL.* [En línea] 2009-2012. http://www.postgresql.org.es/.

31. **Nokia Corporation.** Qt Developer Network. *Qt Developer Network.* [En línea] 2011. http://qt-project.org/.

32. Zator Systems. *Zator Systems.* [En línea] 1990-2012. http://www.zator.com/.

33. Zona Qt. *Zona Qt.* [En línea] http://www.zonaqt.com/.

ANEXOS

Anexo 1: Tarea de la ingeniería #2: Obtener los parámetros para la generación de datos.

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario: 1**
Nombre Tarea: Obtener los parámetros para la generación de datos.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 0,6 semanas
Fecha Inicio: 14-03-2012	**Fecha Fin:** 16-03-2012
Programador Responsable: Abel Delgado Rodríguez	
Descripción: Los parámetros que se obtendrán serán: la cantidad de datos a generar por tablas; y el tipo de generación (aleatoria o desde una lista).	

Anexo 2: Tarea de la ingeniería #3: Diseñar interfaz con los parámetros de cada tipo de atributo.

Tarea de Ingeniería	
Número Tarea: 3	**Número Historia de Usuario: 2**
Nombre Tarea: Diseñar interfaz con los parámetros de cada tipo de atributo.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1 semana
Fecha Inicio: 19-03-2012	**Fecha Fin:** 23-03-2012
Programador Responsable: Abel Delgado Rodríguez	
Descripción: Esta interfaz en dependencia del tipo de datos del atributo que se seleccione dará la posibilidad de especificar los valores mínimo y máximo para la generación.	

Anexo 3: Tarea de la ingeniería #4: Obtener parámetros de cada atributo.

Tarea de Ingeniería	
Número Tarea: 4	**Número Historia de Usuario: 2**
Nombre Tarea: Obtener parámetros de cada atributo.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1 semana
Fecha Inicio: 26-03-2012	**Fecha Fin:** 30-03-2012
Programador Responsable: Abel Delgado Rodríguez	
Descripción: Se obtendrán los parámetros introducidos en la interfaz anterior.	

Anexo 4: Tarea de la ingeniería #5: Generar datos aleatorios.

Tarea de Ingeniería	
Número Tarea: 5	**Número Historia de Usuario: 2**
Nombre Tarea: Generar datos aleatorios.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1 semana
Fecha Inicio: 02-04-2012	**Fecha Fin:** 06-04-2012
Programador Responsable: Abel Delgado Rodríguez	
Descripción: En dependencia de los parámetros obtenidos y el tipo de datos de cada atributo se generarán los datos teniendo en cuenta también la cantidad especificada por el usuario.	

Anexo 5: Tarea de la ingeniería #6: Insertar los datos en la base de datos.

Tarea de Ingeniería	
Número Tarea: 6	**Número Historia de Usuario: 2**
Nombre Tarea: Insertar los datos en la base de datos.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1 semana
Fecha Inicio: 09-04-2012	**Fecha Fin:** 13-04-2012
Programador Responsable: Abel Delgado Rodríguez	
Descripción: Los datos generados se insertarán en la base de datos.	

Anexo 6: Tarjeta CRC para la clase GenerarCadenaBits.

Tarjeta CRC	
Clase: GenerarCadenaBits	
Responsabilidades	**Colaboraciones**
Genera cadenas de bits, tanto de forma aleatoria como a partir de una lista previamente introducida por el usuario.	

Anexo 7: Tarjeta CRC para la clase GenerarFecha.

Tarjeta CRC	
Clase: GenerarFecha	
Responsabilidades	**Colaboraciones**
Genera fechas, tanto de forma aleatoria como a partir de una lista previamente introducida por el usuario.	

Anexo 8: Tarjeta CRC para la clase GenerarBooleano.

Tarjeta CRC	
Clase: GenerarBooleano.	
Responsabilidades	**Colaboraciones**
Genera valores booleanos de forma aleatoria.	

GLOSARIO DE TÉRMINOS

APIs: (Application Programming Interface - Interfaz de Programación de Aplicaciones) es un conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación.

Código abierto: Es el término con el que se conoce al software distribuido y desarrollado libremente.

Cstdlib: C Standard General Utilities Library (Biblioteca estándar de C de propósito general).

Fase: Son los pasos en que se descomponen las metodologías. Cada fase puede o no estar subordinada a otra fase, pudiendo existir entre ellas relaciones de dependencia.

Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje de scripting (lenguaje interpretado) entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Hardware: Componentes físicos que constituyen las Computadoras y demás dispositivos periféricos.

Herramienta: Subprograma o módulo encargado de funciones específicas y afines entre sí para realizar una tarea. Una aplicación o programa puede contar con múltiples herramientas a su disposición. Por ejemplo, el corrector ortográfico puede ser una herramienta en una aplicación para redactar documentos, pero no es una aplicación en sí misma.

Identación: anglicismo (de la palabra inglesa indentation) de uso común en informática que significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores para separarlo del texto adyacente, lo que en el ámbito de la imprenta se ha denominado siempre como sangrado o sangría. En los lenguajes de programación de computadoras, la identación se utiliza para mejorar la legibilidad del código fuente por parte de los programadores, teniendo en cuenta que los compiladores o intérpretes raramente consideran los espacios en blanco entre las sentencias de un programa.

IEEE: Corresponde a las siglas del Instituto de Ingenieros Eléctricos y Electrónicos. Es una asociación estadounidense dedicada a la estandarización internacional sin fines de lucro formada por profesionales de las nuevas tecnologías.

Multihilo: Es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente. Un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea.

Plugin: Es una aplicación informática que añade funcionalidades específicas a un programa principal. Su nombre significa enchufable y su presencia es muy habitual en los navegadores web, en reproductores de música y en sistemas de gestión de contenidos. Los plugins no son parches ni actualizaciones, sino propiedades añadidas a los programas originales, aparecidas por primera vez a mediados de los años 70.

Software: Software es un término genérico que designa al conjunto de programas de distinto tipo (sistema operativo y aplicaciones diversas) que hacen posible operar con el ordenador.

STL: ("Standard Templete Library"). Parte de la LE (librería estándar) genuina de C++ (que no es heredada de C) y que responde a la forma "++" de hacer las cosas.

Tecnología: Abarca un conjunto de técnicas, conocimientos y procesos para el diseño y construcción de objetos para satisfacer necesidades humanas. La tecnología puede referirse a objetos que usa la humanidad (como máquinas, utensilios, hardware), pero también abarca sistemas, métodos de organización y técnicas, se basa en aportes científicos.