Universidad de las Ciencias Informáticas

Facultad 10

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Título: Guía de Buenas Prácticas para obtener mantenibilidad en bases de datos desarrolladas en PostgreSQL

Autoras: *Dayrene Ronquillo García*

Yadira Fuentes Cano

Tutoras: *Ing. Sonia Guerrero Lambert*

Ing. Yusleydi Fernández del Monte

CO-Tutor: *Ing. Liermes Ferriol Men*

Ciudad de La Habana, Cuba. Junio, 2010.

"Es bueno tener en costumbre algunos vicios como pueden ser fumar, comer cerdo, beber alguna sobrecopa o no hacer gimnasia, para que si algún día cae uno enfermo tenga el médico algo que prohibir y uno sane. Pero si uno es todo virtud, en cayendo enfermo morirá, por impotencia de mejora".

Declaración de Autoría

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

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

_____	_____
Dayrene Ronquillo García	Yadira Fuentes Cano
Autora	Autora
_____	_____
Ing. Sonia Guerrero Lambert.	Ing. Yusleydi Fernández del Monte.
Tutora	Tutora

Tutor: Ing. Yusleydi Fernández del Monte.

- Graduada en el año 2008 de Ingeniero en Ciencias Informáticas.
- Tiene 5 años de experiencia en el trabajo vinculado con la Calidad de Software.
- Tiene 2 años de experiencia en el trabajo vinculado con la Gestión de Conocimientos.
- Tiene 2 años de experiencia en la docencia universitaria.
- Ha publicado y realizado presentaciones en eventos nacionales e internacionales sobre temas de gestión de conocimientos y calidad de software.
- Se desempeña como profesora de la facultad 10.

Tutor: Ing. Sonia Guerrero Lambert.

- Graduada en el año 2008 de Ingeniero en Ciencias Informáticas.
- Tiene 5 años de experiencia en el trabajo vinculado con la Calidad de Software.
- Tiene 2 años de experiencia en el trabajo vinculado con la Gestión de Conocimientos.
- Tiene 2 años de experiencia en la docencia universitaria.
- Ha publicado y realizado presentaciones en eventos nacionales e internacionales sobre temas de gestión de conocimientos y calidad de software.
- Se desempeña como profesora de la facultad 10.

CoTutor: Ing. Liermes Ferriol Mena.

- Graduado en el año 2008 de Ingeniero en Ciencias Informáticas.
- Tiene 4 años de experiencia en el trabajo vinculado con las Bases de Datos.
- Tiene 2 años de experiencia como Arquitecto de Datos.
- Se desempeña como especialista en informática del UCID y presta servicios en el proyecto ERP Cuba en la línea de planificación.

Agradecimientos

A mi mamá y mi papá: Mi mamá por ser la mejor madre del mundo, por luchar junto a mí desde el inicio de mi vida y a mi papá por confiar en mis posibilidades de salir adelante; a los dos, por estar conmigo en los momentos más importantes de mi vida.

A mi gorda: por defenderme y ayudarme siempre que lo he necesitado, por ser tan especial conmigo, en fin por quererme tanto.

A toda mi familia: mis tías (Yane, Are y Belky) que para mí son mi otra mamita; Yane que a pesar de estar tan lejos siempre ha estado muy cerca ayudándome y aconsejándome, a Are por darme tantas fuerzas para seguir y a Belky por ser mi tía Pochi. También a mis tíos (Luisi y Aure) y a todas mis primas y primos.

A Misael: a ti mi amor, por estar siempre junto a mí, por ayudarme y comprenderme tanto, por ser tan especial y por darme tanto amor.

A Mely y Aniu: por ser mis hermanos y por estar junto a mí en todo momento.

A Ernesto: por haberme ayudando y aconsejado tanto en mis primeros años.

A la virgen de la Caridad del Cobre: por protegerme y hacer que mis deseos se cumplan.

A todos mis amigos: los que me soportaron durante estos 5 años (Lisbet, Cary, Ailyn, Ailén) y los que han compartido todo conmigo desde niña (Ramón, Juve, Maidi y Alaín); siempre han estado ahí para alentarme en aquellos momentos en que el ánimo y la perseverancia no me acompañaban.

A mis compañeros de aula: Yoa, Dariel, Dalvis, Eilen, Frangel y Gleyser.

A mis tutoras y co-tutor: por estar siempre pendientes y brindarme lo mejor de ellos.

A mi compañera de tesis: por su ayuda, comprensión y sacrificio.

A mis vecinos y amigos: (Negra, Pichi y Mongo) por ser como mi familia y ayudarme tanto, al igual que (Grani y Siomara).

A mis maestros: (Meri, Santo y Maye), es imposible que los olvide, gracias a ellos hoy soy lo que soy.

Dayrene

Agradecimientos

Hoy es un día muy especial para mí, es un momento en el que se resume toda mi vida de estudiante llena de sacrificios, alegrías, sueños, tristezas, para enfrentarme al mundo como una profesional. Muchas son las personas que de manera directa o indirecta me han ayudado en la realización de este trabajo, por lo que quiero dejar constancia de todas ellas y agradecerles con sinceridad su ayuda.

A mi mamá que ha sido toda mi inspiración y es por ella por la que me he sacrificado, gracias por estar siempre a mi lado dándome ánimo, fuerzas y aliento, para ti este regalo.

A mi papito que sé que está más que orgulloso de si hijita.

De gran importancia es para mí mencionar la inmensa gratitud que debo a mis abuelas, dos tesoros que me ha regalado la vida, por apoyarme en todo cuanto hizo falta para que me sintiera tranquila y con ánimos para seguir adelante.

A mis tías, tíos y primos que de una forma u otra me han apoyado.

A mi tía Tita que siempre ha sido y será para mí un ejemplo como madre, amiga, compañera, es un ser al que le debo mucho respeto admiración.

A mi prima Yanet que más que prima es como mi hermana. Gracias Yane por haberme dado un ahijado bello y adorable, que además es mi otro pedacito de vida.

Mención especial merece la familia Mederos Fornel, que durante tres años de mi carrera han formado parte de mi vida, me han apoyado y guiado y sobre todo me han cuidado como si fuera su hija. A Jorge Mario Mederos Fornel que sobre todas las cosas le debo mi vida entera para agradecerle todo lo que ha hecho por mí, decirle que lo quiero mucho.

A las tutoras por contribuir con el trabajo y darnos siempre ánimos para seguir adelante.

A todos mis compañeros de la universidad que durante todo este tiempo me han ayudado mucho y han sido parte de mi vida a diario en la Universidad (Eilen, Ana, Dalvis, Ailin, Claudia, Dayana, Yudeisy).

Yadira

Dedicatoria

A mi mamá; por ser mi faro, la luz que me guía cuando pienso que el mundo me cae encima, por darme tantas fuerzas para seguir adelante. A ti, con todo mi amor te regalo el presente y futuro que una vez soñaste para mí.

A mi papá; por confiar en mis posibilidades de triunfar y por estar tan seguro siempre de mí. A ti, para que siempre estés orgulloso de mí.

A mi abuela Olimpia y mi tío Migue; porque aunque ya no están conmigo estoy segura que este momento habría sido de enorme felicidad para ustedes y sé que estuvieran muy orgullosos de mí.

Dayrene

Dedico este Trabajo de Diploma a toda mi familia por haberse sacrificado durante todos estos años, principalmente a mi mama Niurka Cano Pedroso por todo el amor infinito que me ha dado, por sus consejos, por siempre guiarme por un camino correcto, por su dedicación, paciencia, por sus noches sin dormir cuando tenía pruebas, cuando estaba enferma, por su preocupación cuando no me sentía bien y porque sé además que nunca dejaré de ser su niña.

A Dayrene García Ronquillo, compañera de tesis, por todo su empeño y sacrificio para la realización de este trabajo.

A mis amigos del alma Ailen, Lisbet y Yoanni, quiero que sepan que me sirvió de mucho haberlos conocido, les agradezco infinitamente el haber compartido conmigo momentos buenos y malos y sobre todo decirles que en mí siempre habrá un lugarcito para cada uno de ustedes, sea cual sea la distancia.

Yadira

El mantenimiento es una actividad primordial en el proceso de desarrollo del software, cada vez se hace más imprescindible realizar productos que se adapten rápidamente a los cambios que se les realiza luego de su puesta en funcionamiento. El Centro de Desarrollo de Tecnologías de Datos (DATEC) está presentando problemas en la realización de bases de datos altamente mantenibles; por lo que en este trabajo se presenta una guía de buenas prácticas para elevar la mantenibilidad en las bases de datos realizadas en PostgreSQL desde su proceso de creación en los proyectos productivos de DATEC. La guía está basada en buenas prácticas reconocidas a nivel mundial así como de la experiencia de varios administradores de bases de datos. Para comprobar la necesidad de la misma se realizaron entrevistas en varios centros laborables a nivel nacional demostrando que su creación es imprescindible para optimizar el trabajo. Se utilizan además métodos teóricos como el histórico – lógico y el analitico – sintético. Para evaluar el resultado obtenido se aplica el método Delphi a once especialistas en el tema, constatando que la guía presenta una buena estructura y que abarca los contenidos necesarios.

Palabras claves*: Mantenimiento de software, Mantenibilidad, Bases de datos, PostgreSQL.*

Tabla de Contenidos

Antiguamente, el concepto de calidad se encontraba en un segundo plano, lo importante era producir. Pasados los años llegó el momento en que la oferta superó la demanda, es decir, los clientes tenían dónde elegir y es entonces cuando la preocupación por la calidad supera la preocupación por producir. Desde aquel momento, el mundo empresarial se encuentra marcado por el acelerado ritmo del desarrollo tecnológico, donde las organizaciones están cada vez más conscientes de la importancia que presentan las bases de datos para alcanzar sus metas y satisfacer las necesidades de los clientes.

A menudo, se hace imprescindible realizarles mejoras o modificaciones a las bases de datos debido a los cambios que ocurren en los requisitos y necesidades de la organización que interactúa con ellas, es decir, existe una necesidad de mantenimiento del software. Este proceso de crear y mantener la información de la base de datos actualizada, con la calidad requerida y con todos lo errores depurados es muy importante, pues ayuda a mantener la lealtad de los usuarios o clientes.

Según estudios realizados por Francisco Ruiz y Macario Polo[1] el mantenimiento de software implica alrededor del 80% de los costes en el ciclo de vida de un producto, o sea, que este es la parte más costosa del software y estadísticamente está comprobado. A pesar de esto, la mayor parte de las organizaciones desarrolladoras de software no prestan la suficiente atención a este proceso. La tendencia es creciente con el paso del tiempo, tal como se muestra en la figura1:[1]

Referencia	Fechas	% Mantenimiento
[Pressman, 1993]	años 70	35%-40%
[Lientz y Swanson, 1980]	1976	60%
[Pigoski, 1997]	1980-1984	55%
[Pressman, 1993]	Años 80	60%
[Rock-Evans y Hales, 1990]	1987	67%
[Schach, 1990]	1987	67%
[Pigoski, 1997]	1985-1989	75%
[Frazer, 1992]	1990	80%
[Pressman, 1993]	Años 90 (prev.)	90%

Figura1. Tendencia creciente del costo del mantenimiento del software.

[1] Pertenecen al Departamento de Informática del Grupo Alarcos de la "Universidad de Castilla-La Mancha", situada en la Ciudad Real.

Estos costes de mantenimiento de software se pueden reducir si se produce un software de mejor calidad. Igualmente, también se pueden reducir los esfuerzos de mantenimiento de software si este se realiza utilizando técnicas que mejoren alguna de sus características y contribuyan a elevar la calidad del mismo.

En Cuba se realizan esfuerzos para llevar a cabo la informatización de la sociedad, comenzando fundamentalmente por las empresas y organizaciones, donde este aspecto se ha convertido en elemento primordial a tener en cuenta. En los centros de desarrollo de software, cada vez se aboga más por establecer procedimientos que faciliten la fase de mantenimiento del software.

Según entrevistas realizadas a especialistas en el tema, en empresas tales como la Terminal de Contenedores de La Habana (TCH), Softel, proyectos de la Universidad de las Ciencias Informáticas (UCI), entre otros, se detectaron elementos relacionados con la calidad del proceso de mantenimiento. Todos los profesionales entrevistados coinciden con la alta prioridad que se le debe asignar a este proceso y destacan que la mantenibilidad es un factor que no tienen cómo concedérselo a su producto. También señalan que la calidad de dicho proceso se ve debilitada por los siguientes elementos:

- La ejecución del proceso de mantenimiento es mediante un estilo libre, establecido por el propio programador.
- Características internas del sistema considerablemente hostil, como un diseño pobre de las estructuras de datos, mala codificación y lógica defectuosa.
- Poca disponibilidad de documentación y especificaciones de diseño, o la existente es incomprensible, incorrecta o insuficiente.
- No existe una correcta comprensión del código del programa.

En la Universidad, no existe un modelo que le garantice a los encargados de desarrollar bases de datos del Centro de Desarrollo de Tecnologías de Datos (DATEC) lograr una facilidad de mantenimiento en las bases de datos realizadas en PostgreSQL, es decir, elevar la mantenibilidad de las mismas. Estos desarrolladores tienen un pobre conocimiento de cómo obtener este factor en su trabajo, lo que dificulta que este se realice con la calidad requerida. Los clientes actuales no exigen mantenibilidad, sin embargo, si esto llegase a ocurrir no habría forma de elevar el nivel de mantenibilidad en las bases de datos.

Luego de este análisis y tomando en cuenta la situación actual, se enuncia el siguiente **problema científico:** ¿Cómo elevar la mantenibilidad en bases de datos que se desarrollan en PostgreSQL?

Para dar solución al problema científico se plantea el siguiente **objeto de estudio:** Procesos de obtención de mantenibilidad en las bases de datos, y como **campo de acción:** Procesos de obtención de bases de datos realizadas en PostgreSQL teniendo en cuenta la mantenibilidad durante el Diseño, Configuración, Arquitectura y Codificación.

El **objetivo** de la investigación es: Elaborar una guía que contenga un conjunto de buenas prácticas para elevar la mantenibilidad en las bases de datos realizadas en PostgreSQL. La investigación se sustenta en la siguiente **idea a defender:** Con la utilización de la guía de buenas prácticas para obtener mantenibilidad se pueden lograr bases de datos mantenibles en PostgreSQL. Para dar cumplimiento al objetivo de la investigación se definieron una serie de **tareas de investigación**:

1. Consultar la evolución y tendencia de la mantenibilidad en bases de datos para conocer el entorno en que se ha desarrollado esta característica en las mismas.
2. Aplicar entrevistas y encuestas para conocer cómo se desarrollan las bases de datos actualmente en DATEC.
3. Consultar los aspectos de las bases de datos para conocer las actividades que pueden o no hacerse para aumentar la mantenibilidad en las mismas.
4. Elaborar una guía compuesta por un conjunto de buenas prácticas para cumplir con el objetivo de la investigación.
5. Aplicar el método Delphi para pronosticar el impacto de la guía destinada a obtener bases de datos mantenibles según especialistas en el tema.

Para la realización de la investigación se utilizan diferentes métodos científicos: **Teóricos, Empíricos** y **Estadísticos**. Los métodos teóricos se usan para proporcionar la interpretación y fundamentación de los diferentes datos recopilados a lo largo del proceso de investigación, los métodos empíricos se emplean para posibilitar la recogida de información relacionada con los procesos de obtención de mantenibilidad en las bases de datos y los estadísticos para obtener datos cuantitativos con el objetivo de demostrar la efectividad de la propuesta.

Dentro de los **métodos teóricos** se emplean el **Análisis histórico - lógico** y el **Analítico - sintético**. El primero permite realizar un estudio de la trayectoria de las bases de datos realizadas en PostgreSQL, para conocer los principales hitos de su evolución, así como las tendencias de su proceso de desarrollo

permitiendo confirmar la necesidad de la mantenibilidad en las mismas. El segundo se utiliza para analizar los documentos y teorías sobre las bases de datos realizadas en PostgreSQL, su utilización en la antigüedad y en la actualidad, buscar sus características y elementos fundamentales para luego seleccionar las buenas prácticas que permiten obtener mantenibilidad en las bases de datos y adaptarlas a DATEC.

Los **métodos empíricos** utilizados fueron la **Entrevista** y la **Encuesta**. Ambos métodos se aplican a especialistas en esta área de conocimientos y a encargados de desarrollar bases de datos utilizando el gestor PostgreSQL, con el propósito de obtener los elementos necesarios para lograr un nivel de información sobre la situación actual que presenta la Universidad en el desarrollo de las bases de datos mantenibles. Además, a través de las encuestas se logra determinar el nivel de correctitud de la guía de buenas prácticas.

El **método estadístico** utilizado es el **Delphi** basado en el criterio de especialistas, a los cuales se les presenta la guía de buenas prácticas para obtener mantenibilidad en bases de datos desarrolladas en PostgreSQL y se obtienen datos cuantitativos que muestran el nivel de aceptación de la misma.

En esta investigación la **población** está compuesta por las personas que trabajan en DATEC realizando bases de datos en PostgreSQL, de ella se obtiene una **muestra** significativa seleccionada mediante la técnica de muestreo no probabilístico utilizando el muestreo intencional.

El presente trabajo de diploma se encuentra estructurado en **tres capítulos**, donde cada uno consta de introducción, desarrollo y conclusiones. En el **primer capítulo** titulado "*Fundamentación Teórica de la Guía de Buenas Prácticas para obtener Mantenibilidad* ", se presenta una investigación sobre las bases de datos, los procesos de mantenimiento y la mantenibilidad. Incluye un estudio del tema tratado en el ámbito internacional, nacional y en la Universidad; además de las metodologías y normas existentes que de una forma u otra dan solución a los problemas de mantenibilidad en las bases de datos desarrolladas en PostgreSQL. El **segundo capítulo** nombrado "*Guía de Buenas Prácticas para obtener Mantenibilidad*" consiste en la presentación de la guía de buenas prácticas para obtener mantenibilidad en las bases de datos. El **tercer capítulo** titulado "*Evaluación de la Guía de Buenas Prácticas para obtener Mantenibilidad*", se realiza el análisis y evaluación de la propuesta de buenas prácticas.

1.1 Introducción

En este capítulo se exponen elementos relacionados con el mantenimiento y la mantenibilidad del software, se precisan términos importantes para la comprensión de la investigación, tales como: el proceso de mantenimiento, algunas metodologías, procedimientos y estándares. También se abordan elementos referentes a las características de los diferentes gestores de bases de datos y en especial de PostgreSQL, además de las principales tendencias actuales.

1.2 Conceptos

1.2.1 Mantenimiento del software (MS)

El IEEE[2] define **mantenimiento del software** como: *"Es el proceso de modificar un componente o sistema de software después de su entrega para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarlo a cambios en el entorno".* [2]

Según la definición recogida en el estándar IEEE 1219 se entiende por **mantenimiento del software**: "*La modificación de un producto de software después de la entrega para corregir fallos, para mejorar el rendimiento u otros atributos, o para adaptar el producto a un entorno modificado*". [3]

El ANSI/IEEE brinda la siguiente definición de **mantenimiento del software**: "*Las modificaciones de los productos software después de su entrega para corregir fallos, mejorar rendimiento u otros atributos o adaptar el producto a un cambio de entorno".* [1]

Una definición similar de **mantenimiento del software** es dada por ISO/IEC: "*Un producto software soporta una modificación en el código y su documentación asociada para la solución de un problema o por la necesidad de una mejora. Su objetivo es mejorar el software existente manteniendo su integridad.*" [4]

Pressman (1998) define: "*La fase de **mantenimiento del software** se centra en el cambio que va a asociado a la corrección de errores, a las adaptaciones requeridas a medida que evoluciona el entorno del software, y a cambios debidos a las mejoras producidas por los requisitos cambiantes del cliente*". [5]

[2] Institute of Electrical and Electronics Engineers, es decir, Instituto de Ingenieros Eléctricos y Electrónicos

En la investigación se considera que **mantenimiento de software** se refiere al: "*Cambio o modificación a realizar sobre un producto de software ya sea para corregir una falla, para mejorar algún elemento o realizar una adaptación al entorno cambiante".*

1.2.2 Mantenibilidad

El IEEE (1990) define **mantenibilidad** como: "*La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno"*. Esta definición está directamente conectada con la definición del IEEE para **mantenimiento del software** que se expuso anteriormente. El atributo de calidad que influye de forma directa en los costes y necesidades del **mantenimiento de software** es la **mantenibilidad**, debido a que a mayor mantenibilidad (*facilidad de mantenimiento*), menores costes de mantenimiento, y viceversa.

La ISO[3] 9126 define el concepto de **mantenibilidad** como: "*La facilidad con que una modificación puede ser realizada"* y está indicada por los siguientes subatributos:

- *Facilidad de análisis*: Capacidad del producto software para diagnosticar deficiencias o causas de fallos en el software.
- *Facilidad de cambio*: Capacidad del producto software que permite la ejecución de una modificación específica en ella misma.
- *Estabilidad*: Capacidad del producto de software para evitar defectos no esperados debidos a modificaciones en el mismo.
- *Facilidad de prueba*: Capacidad del producto software que permite al software que ha sido modificado ser evaluado. [6]

La ISO 9126 (1991): brinda otra definición de **mantenibilidad:** *"Es el esfuerzo necesario para realizar modificaciones específicas"*. [7]

Pressman (1998) define la **mantenibilidad** como: *"Medida cualitativa de la facilidad de comprender, corregir, adaptar y/o mejorar el software"*. [8]

[3] Internatinal Organization for Standardization, es decir, Organización Internacional para la Estandarización.

Se considera en la investigación que la **mantenibilidad** se refiere a*: "La facilidad con que un producto o componente de software puede recibir cambios o modificaciones".* Estos cambios o modificaciones pueden incluir correcciones, mejoras o adaptación del software a cambios en el entorno, en los requerimientos o en las especificaciones funcionales.

1.3 Mantenimiento del software

Según las definiciones analizadas de mantenimiento, se puede observar que existen varios tipos de mantenimiento según el tipo de modificación que se desee realizar sobre el software. Cuando se refiere en la definición a corregir defectos se asocia al mantenimiento correctivo, mejorar el rendimiento al mantenimiento perfectivo, para conseguir una mejora del software el preventivo y adaptar a cambios del entorno a mantenimiento adaptativo, quedando referidos los 4 tipos de mantenimiento existentes:

- Mantenimiento CORRECTIVO
- Mantenimiento PERFECTIVO
- Mantenimiento ADAPTATIVO
- Mantenimiento PREVENTIVO

A continuación se hace un breve desglose sobre la finalidad que persigue cada uno de ellos.

¿Qué es mantenimiento correctivo?

A pesar de las pruebas y verificaciones que aparecen en las etapas anteriores del ciclo de vida del software, los programas pueden tener defectos. El **mantenimiento correctivo** tiene por objetivo localizar y eliminar los posibles defectos de los programas. Un defecto en un sistema es una característica del sistema con el potencial de causar un fallo. Un fallo ocurre cuando el comportamiento de un sistema es diferente del establecido en la especificación. Este tipo de mantenimiento suele ser necesario en alguna de las siguientes ocasiones:

- Cuando el programa falla o aborta.
- Un programa produce un resultado que no es acorde con los requisitos.
- Los diseños y requisitos no están acordes con el software que los soporta.
- La documentación de usuario lleva a conclusiones erróneas al propio usuario provocando la ejecución de actividades que provocan resultados incorrectos o fallos en el sistema.

En el proceso de desarrollo del software, son diversas las causas que pueden originar los defectos, aunque según algunos estudios realizados en la Universidad “Rey Juan Carlos”, la mayor parte se origina durante la especificación de requisitos y de diseño, como muestra la figura 2. [5]

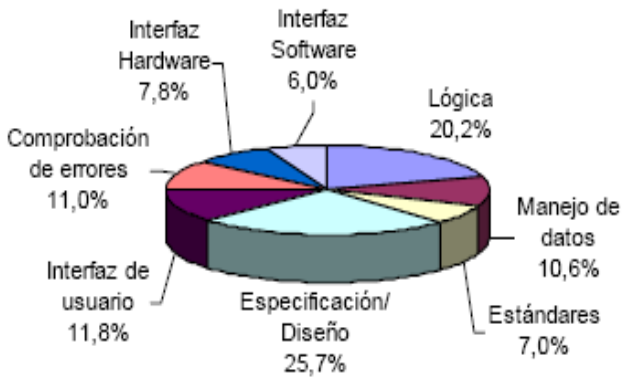


Figura2. Origen de los defectos del software.

¿Qué es mantenimiento perfectivo?

Mantenimiento perfectivo es un método para pulir o refinar la calidad del software y su documentación. El objetivo principal del mantenimiento perfectivo es mejorar la calidad del software para hacerlo más mantenible y reducir el coste e impacto de los cambios de los procesos de mantenimiento.

¿Qué es mantenimiento adaptativo?

Se define **mantenimiento adaptativo** como el proceso para mejorar la funcionalidad del software, hardware y su documentación a un entorno cambiado. Cuando el software ya está en explotación, puede ocurrir que se produzca algún cambio en el entorno que se ejecuta, este tipo de mantenimiento responde a esta situación. Sus pasos son:

- *Definición de requisitos*: Revisar y entender los requisitos de nuevos cambios.
- *Diseño del sistema*: Determina dónde y cuándo implementar los cambios a nivel de sistema.
- *Diseño de datos*: Los cambios en las estructuras de datos deben ser diseñados.
- *Diseño de programa*: Analizar los documentos del diseño del programa para determinar dónde añadir, modificar y borrar las funciones que implementan los cambios propuestos.

- *Diseño del módulo*: Analizar los documentos para determinar cómo integrar cambios en un módulo existente o bien crear uno nuevo.

Este tipo de mantenimiento es cada vez más frecuente, debido principalmente a los cambios que pueden ocurrir en los diversos aspectos de la informática como son: nuevas generaciones de hardware, nuevos sistemas operativos o versiones de los antiguos y mejoras en los periféricos o en otros elementos del sistema.

¿Qué es mantenimiento preventivo?

Mantenimiento preventivo es el que se ejecuta para prevenir fallos antes de que estos ocurran. Por ejemplo, un chequeo antes de compilar y ejecutar una aplicación. Sus técnicas son:

- *Análisis de complejidad*: Medir la complejidad de los módulos.
- *Análisis de funcionalidad*: Medir la funcionalidad de un sistema ayuda a estimar su valor.
- *Ingeniería inversa/recuperación de diseño*: Debido a que el código no esté acorde con la documentación.
- *Reingeniería por partes (Piecewise reengineering)*: Para sistemas de vida corta y cambios revolucionarios necesarios.
- *Traslación / reestructuración / modularización*: Migración de Plataforma.

Este tipo de mantenimiento presenta alternativas para la corrección de problemas de calidad:

1) Rediseño completo y reescritura.
2) Reestructuración completa o revisión del código existente.
3) Reestructuración parcial.
4) Retirar el sistema y rediseño completo.

1.4 Análisis de estándares y procedimientos para el mantenimiento del software

Existen diversos estándares que tienen una relación directa o indirecta con el mantenimiento del software:

- Para los procesos del ciclo de vida del software: IEEE 1074 e ISO 12207.
- Para la calidad del software y sus métricas: IEEE 1061 e ISO 9126.
- Para el mantenimiento del software: IEEE 1219 e ISO/IEC 14764.

Los estándares para los procesos del ciclo de vida del software permiten conectar y asociar el proceso de mantenimiento con los demás procesos existentes en el software. Los estándares de calidad del software tienen gran importancia para el mantenimiento del mismo, debido a que los factores de calidad (especialmente la complejidad y la mantenibilidad) inciden de forma directa sobre el esfuerzo de mantenimiento necesario.

A continuación se abordan cuatro estándares, dos dedicados al mantenimiento del software: IEEE 1219 e ISO/IEC 14764 y el resto relacionados con la calidad del software: IEEE 1061 e ISO 9126. Posteriormente se hace referencia a una metodología que también se enmarca en el ámbito del mantenimiento de software.

1.4.1 Estándar ISO 14764

Este es el estándar específico sobre mantenimiento del software publicado por la ISO en 1998. El estándar internacional ISO 14764 presenta los requerimientos para el proceso de mantenimiento del software, contiene las actividades y tareas del mantenedor, proporciona una guía que explica cómo llevar a cabo el proceso de mantenimiento y establece definiciones para los distintos tipos de mantenimiento. La guía es aplicable a la planificación, ejecución y control, mantenimiento, revisión y evaluación del proceso de mantenimiento.

La norma propone un plan que forma parte de la estrategia de mantenimiento, dicho plan es usado para guiar a los mantenedores de software, explica la necesidad de realizar mantenimiento, refiriéndose a quién efectúa ese trabajo y cómo se hace, contiene la documentación y responsabilidades de todos los involucrados. Además, debe incluir qué recursos hay disponibles para el mantenimiento, dónde se hace y cuándo comienza.

Una vez definido dicho plan, el estándar propone establecer una guía para desarrollar el mantenimiento. Esta guía debe contener:

- La descripción del sistema al que se le brinda soporte, aquí se especifican todos los detalles del sistema a mantener.
- Identificación del estado inicial del software, para saber cuáles son los cambios nuevos realizados.
- Descripción del soporte para facilitar el comienzo del desarrollo del mantenimiento del software.

- Identificación de la organización que debe hacer el soporte o mantenimiento para contemplar el objetivo del mantenimiento en el proceso de desarrollo del software.
- Descripción de cualquier acuerdo entre cliente y vendedor, se debe tener claro lo que quiere el cliente por escrito, de este modo el vendedor sabe lo que tiene que hacer para satisfacer al cliente.

Estos son los aspectos fundamentales en cuanto a la estrategia de mantenimiento que propone el estándar. Las actividades que comprende el proceso de mantenimiento son:

- Implementación del proceso.
- Análisis de modificaciones y problemas.
- Implementación de modificaciones.
- Revisión y aceptación del mantenimiento.
- Migración.
- Retiro.

Básicamente éste es el enfoque que brinda la norma ISO 14764 para realizar la actividad de mantenimiento de software. Esta norma identifica adecuadamente qué hacer en las actividades y tareas a desarrollar en el proceso de mantenimiento.

1.4.2 Estándar IEEE 1219

El IEEE 1219 *Standard for Software Maintenance*, hasta 1998 es el único estándar que íntegramente se ocupa del proceso de mantenimiento del software. Describe un proceso iterativo para la gestión y ejecución de las actividades del proceso. Aunque sólo menciona las fases de desarrollo y de producción de un producto de software, éstas cubren todo su ciclo de vida, cualquiera que sea su tamaño o complejidad. Esta norma define cambios en un producto de software a través de un proceso de mantenimiento dividido en fases, el proceso es iterativo y en cascada, con una gran semejanza al ciclo de vida del desarrollo clásico, como se menciona a continuación:

- Identificación del problema.
- Análisis.
- Diseño.
- Implementación.
- Pruebas del sistema.

- Pruebas de aceptación.
- Puesta en producción o liberación de versión.

Dentro de cada una de estas fases, el estándar define una serie de procedimientos que se han de llevar a cabo y con los que se identifica la documentación, las personas y productos de software que intervienen. Esta norma plantea un proceso de mantenimiento con gran nivel de detalle y documentación a llevar para su desarrollo, haciéndolo muy útil y necesario sobre todo en los lugares que se realiza mantenimiento del software, aquí es fundamental la traza que marca el estado y evolución de cada una de las fases pero pudiera resultar excesivo para pequeñas organizaciones que deseen aplicar dicho estándar en el mantenimiento de sus sistemas internos.

1.4.3 Estándar IEEE 1061

El IEEE 1061 *Standard for a Software Quality Metrics Methodology* provee una metodología para establecer requerimientos de calidad e identificar, implementar, analizar y validar métricas de calidad de productos y procesos software. La metodología es aplicable a todo el software, en todas las fases de cualquier estructura del ciclo de vida. En este estándar se establece la mantenibilidad como uno de los factores de la calidad del software.

1.4.4 Estándar ISO 9126

En 1992 se aprobó el estándar ISO/IEC llamado: "*ISO 9126: Software Product Evaluation: Quality Characteristics and Guidelines for their Use*". Este estándar define un modelo de calidad del software en el que la calidad se define como: "*La totalidad de características relacionadas con su habilidad para satisfacer necesidades establecidas o implicadas"*. Los atributos de calidad se clasifican según seis características, las cuales a su vez se subdividen en subcaracterísticas como se muestra en la figura 3.

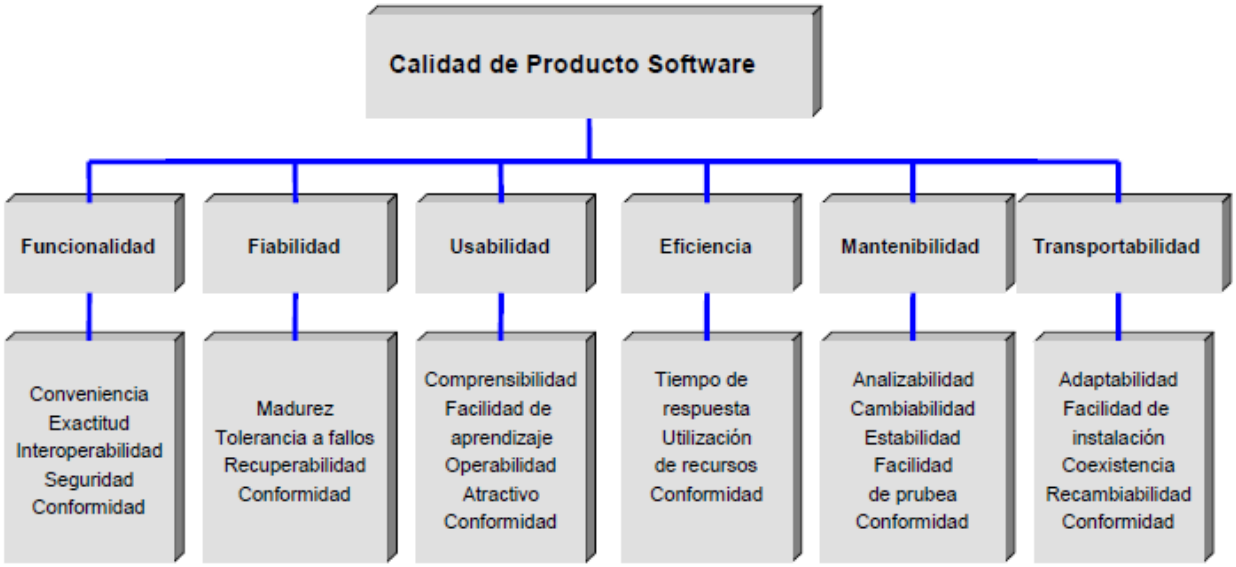


Figura3. Calidad de un producto de software (modelo ISO 9126).

Una de las características que denota la calidad del producto es la mantenibilidad. La ISO 9126 define mantenibilidad como*: "La capacidad de un producto software para ser modificado"* y la subdivide en cinco subcaracterísticas:

- *Analizabilidad*: Capacidad del producto software de diagnosticar sus deficiencias o causas de fallos, o de identificar las partes que deben ser modificadas.
- *Cambiabilidad*: Capacidad del producto software de permitir implementar una modificación especificada de forma previa. La implementación incluye los cambios en el diseño, el código y la documentación. Si el software es modificado por el usuario final, entonces, la cambiabilidad puede afectar a la operabilidad.
- *Estabilidad*: Capacidad del producto software de minimizar los efectos inesperados de las modificaciones.
- *Facilidad de prueba*: Capacidad del producto software de permitir evaluar las partes modificadas.
- *Conformidad*: Capacidad del producto software de satisfacer los estándares o convenciones relativas con la mantenibilidad.

En el avance tecnológico en que se encuentra el mundo, se exige cada vez más la aplicación de estándares internacionales que garanticen la calidad de los productos. Por esta razón, es necesario que todos los desarrolladores de software incluyan en sus procesos estándares de calidad que permitan certificarse en alguno de los modelos. El estándar ISO-9126 establece una guía para la evaluación de la calidad del software, teniendo en cuenta sus características y atributos.

1.4.5 Metodología MANTEMA

La metodología MANTEMA categoriza los diferentes tipos de mantenimiento en:

- **Mantenimiento no planificable (NP):**
  - ✓ *Correctivo Urgente (UC):* Localizar y eliminar los posibles defectos que bloquean el programa o los procesos de funcionamiento de la empresa.
- **Mantenimiento planificable (P):**
  - ✓ *Correctivo No Urgente (NUC):* Localizar y eliminar los posibles defectos de los programas que no son bloqueantes.
  - ✓ *Perfectivo (PER):* Añadir al software nuevas funcionalidades solicitadas por los usuarios.
  - ✓ *Adaptativo (A):* Modificar el software para adaptarlo a cambios en el entorno de trabajo (hardware o software).
  - ✓ *Preventivo (PRE):* Modificar el software para mejorar sus propiedades (calidad, mantenibilidad, entre otros). [1]

1.5 Bases de Datos (BD)

1.5.1 ¿Qué es una base de datos?

Damián Pérez Valdés[4] define una **base de datos** como: "*Una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular*". [9]

Beatriz Pereyra[5] define las **bases de datos** como: "*Un conjunto de datos relacionados entre sí y que tienen un significado implícito*". [10]

[4] Webmaster, Administrador de Sistemas con experiencia en desarrollo web y de aplicaciones.

[5] Pertenece a la Cátedra de Introducción a la Computación de Sistemas Informáticos Aplicados.

C.J.Date[6] brinda la siguiente definición de **base de datos**: "*Un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa dada*". [11]

En la investigación se considera por una **base de datos**: "*Un conjunto de datos relacionados entre sí, los cuales se pueden modificar y acceder rápidamente*".

1.5.2 Tipos de bases de datos

Las BD se pueden dividir en cuatro tipos básicos:

1. Bases de datos de fichero plano (o ficheros por bloques).
2. Bases de datos relacionales.
3. Bases de datos orientadas a objetos.
4. Bases de datos híbridas.

A continuación se presentan breves características de estos cuatro tipos de bases de datos anteriormente mencionados.

¿Qué es una base de datos de fichero plano o ficheros por bloques?

Las **bases de datos de fichero plano** consisten en ficheros de texto divididos en filas y columnas. Estas bases de datos son las más primitivas y quizás ni tan siquiera merezcan considerarse como tales. Pueden ser útiles para aplicaciones muy simples, no para aplicaciones medianas o complejas debido a sus grandes limitaciones.

¿Qué es una base de datos relacional?

Las **bases de datos relacionales** son las más populares actualmente. Su nombre proviene de su gran ventaja sobre las bases de datos de fichero plano: la posibilidad de relacionar varias tablas de datos entre sí, compartiendo información y evitando la duplicidad y los problemas que ello conlleva (*espacio de almacenamiento y redundancia*). Existen numerosas bases de datos relacionales para distintas plataformas (*Access, Paradox, Oracle, Sybase*) y son ampliamente utilizadas. Sin embargo, tienen un punto débil: la mayoría de ellas no admite la incorporación de objetos multimedia tales como sonidos, imágenes o animaciones.

[6] Autor del libro Introducción a los Sistemas de Bases de Datos.

¿Qué es una base de datos orientada a objetos?

Las **bases de datos orientadas a objetos** incorporan el paradigma de la Orientación a Objetos (OO), es decir, están constituidas por objetos que pueden ser de diversos tipos y sobre los cuales se encuentran definidas varias operaciones. Estas bases de datos pueden manejar información binaria (como objetos multimedia) de una forma eficiente. Su limitación suele residir en su especialización, pues suelen estar diseñadas para un tipo particular de objeto.

¿Qué es una base de datos híbrida?

Las **bases de datos híbridas** combinan características de las bases de datos relacionales y las bases de datos orientadas a objetos. Manejan datos textuales y datos binarios, a los cuales se extienden las posibilidades de consulta.

1.6 PostgreSQL

1.6.1 Resumen histórico de PostgreSQL

El Sistema Gestor de Bases de Datos Relacionales Orientado a Objetos conocido como PostgreSQL, está derivado del paquete Postgres escrito en Berkeley. Después de una década de desarrollo, PostgreSQL se ha convertido en el gestor de bases de datos de código abierto más avanzado de la actualidad, ofreciendo control de concurrencia multi-versión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, Perl, PHP y Python).

La implementación del Sistema Gestor de Base de Datos (DBMS) Postgres comienza en 1986, y desde entonces han transcurrido varias revisiones importantes. El primer sistema de pruebas operacional es en 1987 y se muestra en 1988. La versión 1.0 se libera en junio de 1989 a unos pocos usuarios y después la 2.0 en junio de 1990 debido a algunas críticas sobre el sistema de reglas que obliga a su reimplementación. La versión 3.0 aparece en el año 1991 y añade una implementación para múltiples gestores de almacenamiento, un ejecutor de consultas mejorado y un nuevo sistema de reescritura de reglas. En su mayor parte, las siguientes versiones hasta el lanzamiento de Postgre95 se centran en mejorar la portabilidad y la fiabilidad. El número de usuarios de la comunidad casi se duplica durante 1993. Pronto se hace obvio que el mantenimiento del código y las tareas de soporte están ocupando tiempo que debía dedicarse a la investigación. En un esfuerzo por reducir esta carga, el proyecto termina

oficialmente con la versión 4.2. En 1994, Andrew Yu y Jolly Chen añaden un intérprete de lenguaje SQL a este Gestor.

Postgres95 es publicado a continuación en la Web, escrito totalmente en C, su tamaño es reducido un 25% y se ejecuta entre un 30 y un 50% más rápido. En 1996 se hace evidente que el nombre "Postgre95" no resiste el paso del tiempo, los desarrolladores deciden cambiar el nombre y lo llaman PostgreSQL para reflejar la relación entre el Postgres original y las versiones más recientes con capacidades SQL. Al mismo tiempo, hacen que los números de versión partieran de la 6.0, volviendo a la secuencia seguida originalmente por el proyecto Postgres. [12]

1.6.2 Ventajas de PostgreSQL

PostgreSQL ofrece muchas ventajas respecto a otros sistemas de bases de datos, entre las cuales se pueden encontrar: [12]

- *Instalación ilimitada.* PostgreSQL no tiene costos asociados a la licencia del software, lo cual posibilita que se pueda instalar en varios servidores sin violar ningún acuerdo de licencia.
- *Soporte técnico.* Además de numerosas ofertas de soporte por parte de empresas de software libre, existen importantes comunidades de profesionales y entusiastas de PostgreSQL de las que se pueden obtener beneficios y contribución.
- *Ahorros considerables en costos de operación.* El software es diseñado y creado para tener presente un mantenimiento y ajuste mucho menor que los productos de los proveedores comerciales, conservando todas las características, estabilidad y rendimiento.
- *Estabilidad y confiabilidad.* En contraste a muchos sistemas de bases de datos comerciales es extremadamente común que las compañías reporten que PostgreSQL no presenta caídas en varios años de operación de alta actividad.
- *Extensible.* El código fuente está disponible sin costo para todo aquel que necesite extender o personalizar PostgreSQL de alguna manera.
- *Multiplataforma y variadas herramientas gráficas de diseño y administración.* Existen instaladores de los servicios y clientes PostgreSQL para los sistemas operativos Windows y las distribuciones de Linux. También se dispone de diversas herramientas gráficas de alta calidad para administrar las bases de datos y para hacer diseño: Toad, Data Architect, entre otras.

- *Diseñado para ambientes de alto volumen.* Al igual que los principales proveedores de sistemas de bases de datos comerciales, PostgreSQL usa una estrategia de almacenamiento de filas para conseguir una mejor respuesta en ambientes de grandes volúmenes.

1.6.3 Desventajas de PostgreSQL

Entre los posibles inconvenientes o desventajas que presenta PostgreSQL se pueden encontrar: [12]

- Consume gran cantidad de recursos.
- Tiene un límite de 8K por fila, aunque se puede aumentar a 32K, con una disminución considerable del rendimiento.
- Es de dos a tres veces más lento que MySQL.
- Posee menos funciones en PHP.

1.6.4 Diseño de PostgreSQL

PostgreSQL presenta las siguientes características en su diseño: [13]

- Características para la integridad de los datos: claves primarias, llaves foráneas con capacidad de actualizar en cascada o restringir la acción.
- Escritura adelantada de registros (WAL) para evitar pérdidas de datos en caso de fallos por energía, sistema operativo, hardware, entre otros.
- Disparadores (triggers).
- Vistas.
- PostgreSQL es muy Extensible.
  - ✓ Funciones.
  - ✓ Agregación.
  - ✓ Tipos de datos. Permite la creación de tipos de datos personalizados.
  - ✓ Índices compuestos, únicos, parciales, funcionales (sobre funciones) que pueden ser definidos como B-tree, R-tree, hash o GiST, y toda la infraestructura necesaria para extender estos tipos de índices.
  - ✓ Operadores.
- Funciones o procedimientos almacenados.

- ✓ Pueden escribirse en múltiples lenguajes: C, Java, Perl, Python, Ruby, PHP, así como su lenguaje nativo PL/PGSQL.
- ✓ Se usan de tres formas: las que retornan o no valores, las que se usan de triggers, las que retornan tablas.

- Diseñado para entornos de gran volumen de información y alta concurrencia. Utiliza la tecnología MVCC (Multi-Version Concurrency Control), que sirve para lograr un control de concurrencia tan eficiente que generalmente no se requiere de bloqueos.
- Es muy portable: Linux, Unix, BSD's, Mac OS X, Solaris, AIX, Irix, HP-UX, Windows.
- Replicación síncrona y asincrónica.
- Respaldos en línea e incrementales.

MicroOLAP es un diseñador de Base de datos para PostgreSQL, es una sencilla herramienta CASE[7] con interfaz gráfica intuitiva que permite construir una estructura de base de datos clara y eficaz representando todas las tablas, referencias entre ellas, vistas, procedimientos almacenados y otros objetos. Mediante esta herramienta se puede generar fácilmente una base de datos física en un servidor, modificar la misma conforme a cualquier cambio que usted haga al diagrama usando declaraciones ALTER[8].

1.6.5 Configuración de PostgreSQL

PostgreSQL se puede empezar a utilizar al terminar de instalarlo y después de inicializar el "cluster"[9], sin necesidad de configurar nada. Pero si se va a utilizar PostgreSQL para algo importante y con ciertos volúmenes de datos y usuarios es imprescindible que se configure para dicho trabajo. Hay que decir que cualquier base de datos que se esté usando activamente, no solo PostgreSQL, es un elemento dinámico y vivo en el que se están cambiando los datos constantemente y donde el tamaño de los datos

[7] Las herramientas CASE (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en tiempo y de dinero.

[8] Éste comando permite modificar la estructura de un objeto. Se pueden agregar o quitar campos a una tabla, modificar el tipo de un campo, agregar o quitar índices a una tabla, modificar un trigger.

[9] Se define cluster de base de datos, como una instancia de PostgreSQL, donde se lanza un proceso postmaster, que puede gestionar varias bases de datos. En un servidor, puede haber varios cluster, cada uno tendrá su postmaster, y cada postmaster escuchará por un puerto diferente.

almacenados suele ir creciendo con el tiempo. Esto significa que una configuración que funcione bien con ciertos valores hoy, puede que no funcione tan bien después de un año de uso y que necesite ajustarse para que funcione óptimamente.

El comportamiento de PostgreSQL en el sistema se puede controlar con tres ficheros de configuración que se encuentran en el directorio de datos donde inicializa el cluster. Estos tres ficheros son:

1. *pg_hba.conf*. Este fichero se utiliza para definir los diferentes tipos de accesos que un usuario tiene en el cluster.
2. *pg_ident.conf*. Este fichero se utiliza para definir la información necesaria en el caso de que se utilice un acceso del tipo ident en pg_hba.conf.
3. *postgresql.conf*. En este fichero se puede cambiar todos los parámetros de configuración que afectan al funcionamiento y al comportamiento de PostgreSQL en la máquina.

A continuación se explican los cambios más importantes que se pueden hacer en algunos de estos ficheros.

***pg_hba.conf*.** Este fichero se utiliza para definir cómo, dónde y desde qué sitio un usuario puede utilizar el cluster PostgreSQL. Todas las líneas que empiecen con el caracter # se interpretan como comentarios. El resto debe de tener el siguiente formato:
[Tipo de conexión][database][usuario][IP][Netmask][Tipo de autentificación][Opciones]

Dependiendo del tipo de conexión y del tipo de autentificación, [IP], [Netmask] y [opciones] pueden ser opcionales. El tipo de conexión puede tener los siguientes valores, local, host, hostssl y hostnossl. El tipo de método puede tener los siguientes valores, trust, reject, md5, crypt, password, krb5, ident, pam o ldap. Son muchas las posibilidades que brinda este fichero de configuración. Por supuesto que las bases de datos y usuarios usados en este fichero tienen que existir en el cluster para que todo funcione. Algunos de los parámetros solo se pueden usar si se ha compilado con las opciones pertinentes en el proceso de instalación (por ejemplo, hostssl, pam, krb5).

***postgresql.conf*.** Los cambios que se realicen en este fichero afectan a todas las bases de datos definidas en el cluster PostgreSQL. La mayoría de los cambios se pueden poner en producción con un simple 'reload' (*/usr/local/bin/pg_ctl -D /var/pgsql/data reload*), otros cambios necesitan que se arranque de nuevo el cluster (*/usr/local/bin/pg_ctl -D /var/pgsql/data restart*).

A continuación se muestran los parámetros más importantes que se deben cambiar para comenzar a usar PostgreSQL.

- ✓ *max_connections.* Número máximo de clientes conectados a la vez a la base de datos. Se debe incrementar este valor en proporción al número de clientes concurrentes en el cluster PostgreSQL. Un buen valor para empezar es 100.
- ✓ *shared_buffers*. Este parámetro es muy importante y define el tamaño del buffer de memoria utilizado por PostgreSQL. No por aumentar este valor se tiene una mejor respuesta. En un servidor se puede empezar con un 25% del total de la memoria, nunca más de 1/3 (33%) del total. Por ejemplo, en un servidor con 4 Gbytes de memoria se puede usar 1024MB como valor inicial.
- ✓ *work_mem.* Usada en operaciones que contengan ORDER BY, DISTINCT y JOINS. En un servidor se puede usar hasta un 4% del total de la memoria si hay solamente unas pocas sesiones (clientes) grandes. Como valor inicial se puede usar 8 Mbytes.
- ✓ *maintenance_work_mem*. Usada en operaciones del tipo VACUUM, ANALYZE, CREATE INDEX, ALTER TABLE, ADD FOREIGN KEY. Su valor depende mucho del tamaño de las bases de datos. Por ejemplo, en un servidor con 4 Gbytes de memoria se puede usar 256MB como valor inicial.
- ✓ *effective_cache_size.* Parámetro usado por 'query planner' del motor de bases de datos para optimizar la lectura de datos. En un servidor se puede empezar con un 50% del total de la memoria. Como máximo 2/3 (66%) del total. Por ejemplo, en un servidor con 4Gbytes de memoria, se puede usar 2048MB como valor inicial.

Es importante tener presente que al aumentar los valores por defecto de muchos de estos parámetros se tiene que aumentar los valores por defecto de algunos parámetros del Kernel[10] del sistema. [14]

1.6.6 Arquitectura de PostgreSQL

La arquitectura ANSI/SPARC[11] se divide en tres niveles (interno, conceptual y externo) respectivamente como se muestra en la figura 4.

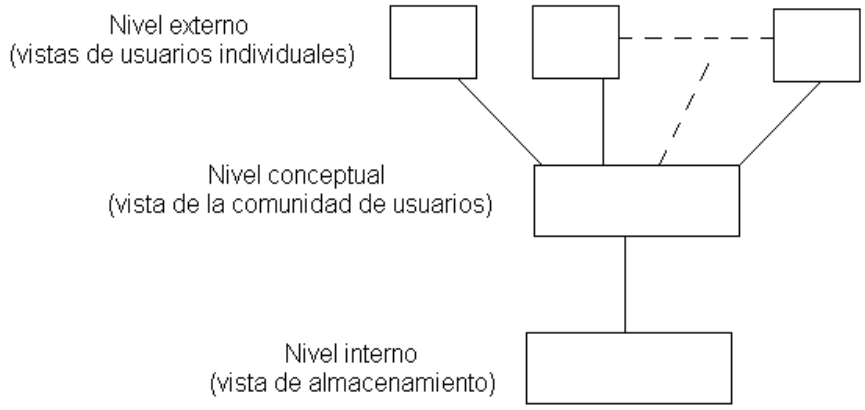


Figura4. Los tres niveles de la arquitectura.

A continuación se puede encontrar una breve explicación de los niveles anteriormente mencionados.

- El **nivel externo** (*también conocido como el nivel lógico de usuario*) es el más próximo a los usuarios; es decir, es el que tiene que ver con la forma en que los usuarios individuales ven los datos.
- El **nivel conceptual** (*también conocido como el nivel lógico de la comunidad, o en ocasiones sólo como el nivel lógico*) es un nivel de indirección entre los otros dos.
- El **nivel interno** (*también conocido como el nivel físico*) es el que está más cerca del almacenamiento físico; es decir, es el que está relacionado con la forma en que los datos están

[10] El Núcleo o Kernel, es un software que actúa de sistema operativo. Es el principal responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma más básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema.

[11] ANSI (Instituto Nacional Americano de Estándares), literalmente, ANSI/Comité de Planeación y Requerimientos de Sistemas; se usa para referirse a la arquitectura de sistemas de base de datos de tres niveles.

almacenados físicamente.

Desde otro punto de vista, un sistema de base de datos puede ser visto como un sistema que tiene una estructura muy sencilla de dos partes, las cuales consisten en un servidor (también denominado parte dorsal o servicios de fondo) y un conjunto de clientes (también llamados partes frontales, aplicaciones para el usuario o interfaces), como se muestra en la figura 5; desde luego, con la misma finalidad principal, apoyar el desarrollo y la ejecución de las aplicaciones de bases de datos. [15]

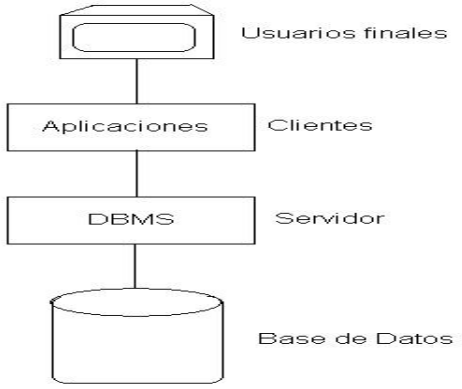


Figura5. Arquitectura Cliente_Servidor.

PostgreSQL funciona con la arquitectura Cliente/Servidor, un proceso servidor (*postmaster*) y una serie de aplicaciones cliente que realizan solicitudes de acciones contra la base de datos a su proceso servidor. Por cada una de estas aplicaciones cliente el proceso postmaster crea un proceso postgres, como se muestra en la figura 6.

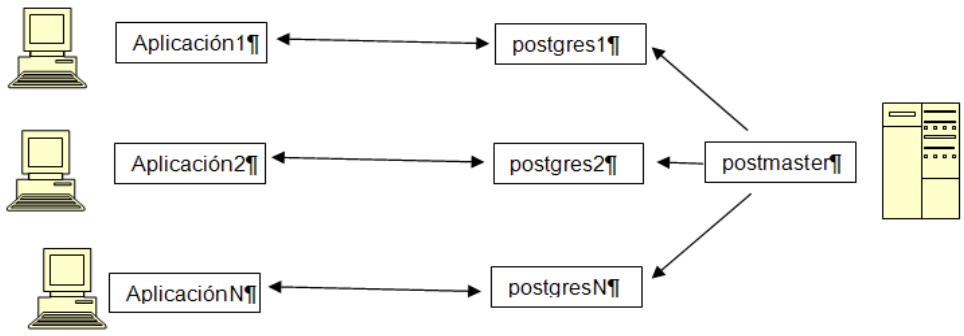


Figura6. Arquitectura de PostgreSQL.

Aquí se ejecutan una serie de aplicaciones cliente (*FrontEnd*) y una serie de procesos en el servidor (*BackEnd*), de modo que, por ejemplo, la interacción entre un programa que se ejecuta en un cliente (ejemplo, pgadmin3, una aplicación PHP, entre otros) y el servidor de base de datos sería como se muestra en la figura 7:

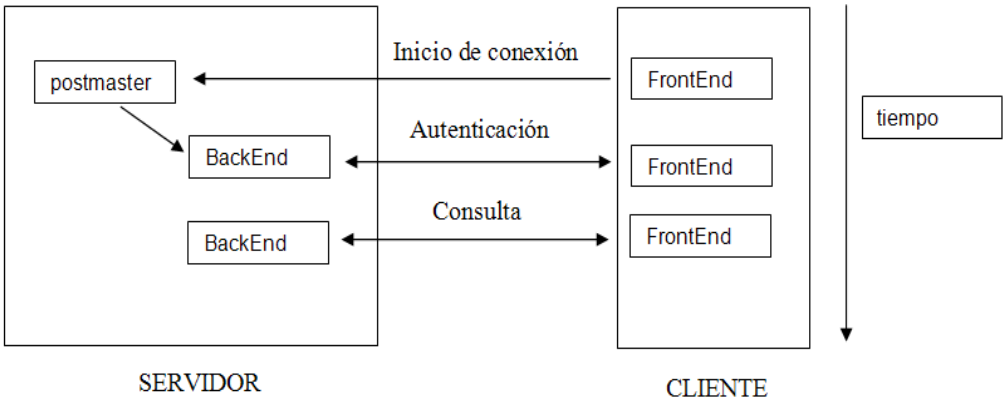


Figura7. Esquema de conexión entre procesos FrontEnd/BackEnd.

La comunicación entre BackEnd/FrontEnd se realiza mediante el protocolo TCP/IP, normalmente por el puerto 5432, creando un socket (/tmp/.s.PGSQL.5432). La arquitectura total de PostgreSQL se presenta en la figura 8.

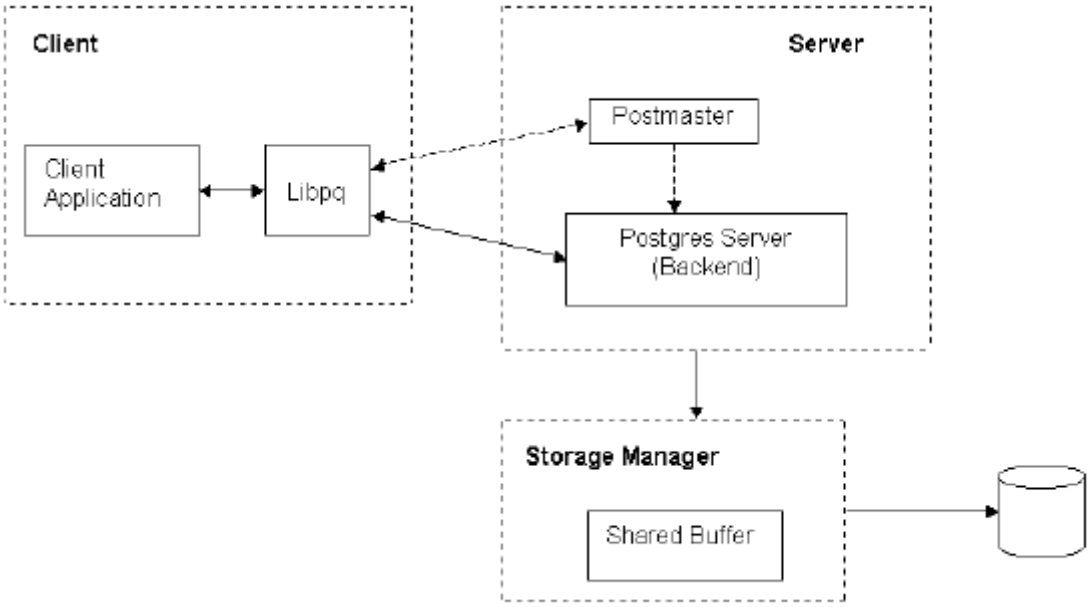


Figura8. Arquitectura total de PostgreSQL.

1) *Libpq* son responsables de manejar la comunicación con los procesos del cliente:

- Establecer la conexión al postmaster.
- Obtención del hilo de rosca del servidor del Postgre para la sesión operacional.

2) El *Servidor* se compone de dos subsistemas: el *postmaster* y el *servidor de Postgre*.

El *Postmaster* es el proceso inicial, el responsable de aceptar la petición de conexión entrante del cliente, de realizar control de la autentificación y de acceso en la petición del cliente y de establecer al cliente la comunicación con el servidor del Postgre, es decir, es quien gestiona los accesos multiusuario y multiconexión, quien levanta la memoria compartida, está al tanto de solicitudes de nuevas conexiones, lanza procesos de atención de demanda realizando las operaciones sobre la base de datos a solicitud de los clientes y realiza el enlazado con los archivos de datos, gestionando estos ficheros, donde los ficheros pueden pertenecer a varias bases de datos. El *servidor de Postgre* maneja todas las consultas y comandos del cliente.

3) El *Administrador de Almacenamiento* (Store Manager) es el responsable de la gestión de la memoria externa general y control de recurso en el back-end, incluyendo la gerencia de almacenador intermediario compartida, de la gerencia de archivo, del control de la consistencia y el encargado de la cerradura.

1.6.7 Codificación de PostgreSQL

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una sola empresa sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales. Los bloques de código que se ejecutan en el servidor, pueden ser escritos en varios lenguajes, con la potencia que cada uno de ellos brinda, desde las operaciones básicas de programación tales como bifurcaciones y bucles, hasta las complejidades de la programación orientada a objetos o la programación funcional. Algunos de los lenguajes que se pueden usar son los siguientes: [12]

- Un lenguaje propio llamado PL/PgSQL (similar al PL/SQL de Oracle)
- C
- C++
- Java PL/Java web
- PL/Perl
- plPHP
- PL/Python
- PL/Ruby
- PL/sh
- PL/Tcl
- PL/Scheme
- Lenguaje para aplicaciones estadísticas R por medio de PL/R

PostgreSQL soporta funciones que retornan “filas”, donde la salida puede tratarse como un conjunto de valores que pueden ser tratados igual a una fila retornada por una consulta (*query en inglés*). Está ampliamente considerado como el sistema de bases de datos de código abierto más avanzado del mundo.

1.6.8 Uso de PostgreSQL

Ejemplos de proyectos que hacen uso de PostgreSQL: [13]

- Agencias gubernamentales de Estados Unidos. US Army, Yahoo, Red Hat, Sun Microsystems, entre otros, como se muestra en la figura 9.

Figura9. Ejemplos agencias gubernamentales en EU que hacen uso de PostgreSQL.

- Ecuador. Universidad Politécnica Salesiana, Palo Santo Solutions Sistemas de información gerencial, la SENACYT Secretaría Nacional de Ciencia y Tecnología, como se muestra en la figura 10.

Figura10. Ejemplos centros en Ecuador que hacen uso de PostgreSQL.

- Perú. Asociación de empleados del BCP, Americatel, Cámara de Comercio de Lima, Prompyme atiende el portal de compras del estado, Grupo Carolina. Sistema de órdenes de trabajo, Gobierno Regional Lambayeque, como se muestra en la figura 11.

Figura11. Ejemplos centros en Perú que hacen uso de PostgreSQL.

- En el desarrollo de la economía cubana las Tecnologías de la Información y las Comunicaciones (TIC) en la actualidad constituyen un actor fundamental. Estas tecnologías son un desafío para la economía debido a que urge la necesidad de cambiar la concepción de las personas respecto a su uso. La mayoría de las aplicaciones que se utilizan en el país son propietarias, las cuales deben

ser sustituidas por herramientas libres, debido a la necesidad que existe de establecer cambios radicales en el sector. El cambio paulatino hacia las aplicaciones libres es eminente; en caso contrario la sociedad sigue dependiendo de las grandes multinacionales que controlan actualmente el mercado y continuará pagando grandes sumas de dinero que pueden emplearse en mejorar la infraestructura tecnológica del país. La Universidad de las Ciencias Informáticas nace con el propósito de que sus graduados fomenten el desarrollo de la Industria Cubana del Software contribuyendo a la informatización de las instituciones del país. El Centro de Desarrollo de Tecnologías de Datos adopta al gestor de bases de datos PostgreSQL como su prototipo de desarrollo, que es además la entidad por parte de la Universidad que dé el impulso necesario para que las empresas cubanas comiencen a utilizar tecnologías de bases de datos libres contribuyendo al necesario proceso de migración que debe llevar a cabo el país.

1.7 Mantenibilidad del software

1.7.1 Factores que afectan directamente la mantenibilidad del software

Existen factores que afectan directamente a la mantenibilidad, de forma que si alguno de ellos no se satisface adecuadamente, esta se resiente. Los tres más significativos son:

1. *Proceso de desarrollo*. La mantenibilidad debe formar parte integral del proceso de desarrollo del software. Las técnicas utilizadas deben ser lo menos intrusivas posible con el software existente.
2. *Documentación*. En múltiples ocasiones, ni la documentación, ni las especificaciones de diseño están disponibles, y por tanto, los costes del mantenimiento se incrementan debido al tiempo requerido para que un mantenedor entienda el diseño del software antes de poder modificarlo. Las decisiones sobre la documentación que debe desarrollarse son muy importantes cuando la responsabilidad del mantenimiento de un sistema se va a transferir a una organización nueva.
3. *Comprensión de Programas*. La causa básica de la mayor parte de los altos costes del MS es la presencia de obstáculos a la comprensión humana de los programas y sistemas existentes. Estos obstáculos surgen de tres fuentes principales:

    - La información disponible es incomprensible, incorrecta o insuficiente.
    - La complejidad del software, de la naturaleza de la aplicación o de ambos.
    - La confusión, mala interpretación u olvidos sobre el programa o sistema.

Se han identificado los factores específicos que influyen en la mantenibilidad según estudios realizados por Miguel Angel Sicilia[12]:

1. Falta de cuidado en las fases de diseño, codificación o prueba.
2. Pobre configuración del producto software.
3. Adecuada cualificación del equipo de desarrolladores del software.
4. Estructura del software fácil de comprender.
5. Facilidad de uso del sistema.
6. Empleo de lenguajes de programación y sistemas operativos estandarizados.
7. Estructura estandarizada de la documentación.
8. Documentación disponible de los casos de prueba.
9. Incorporación en el sistema de facilidades de depuración.
10. Disponibilidad del equipo (computador y periférico) adecuado para realizar el mantenimiento.
11. Disponibilidad de la persona o grupo que desarrolló originalmente el software.
12. Planificación del mantenimiento.[2]

La mantenibilidad se puede considerar como la combinación de dos factores de calidad diferentes: *Reparabilidad* y *Flexibilidad*.

- Un sistema software es **reparable** si permite la corrección de sus defectos con una cantidad de trabajo limitada y razonable. La reparabilidad se ve afectada por la cantidad y tamaño de los componentes o piezas. Un producto software que consiste en módulos bien diseñados es más fácil de analizar y reparar que uno monolítico, pero el incremento del número de módulos no implica un producto más reparable, pues también aumenta la complejidad de las interconexiones entre módulos.
- Un sistema software es **flexible** si permite cambios que satisfagan nuevos requerimientos, es decir, si puede evolucionar. Esta flexibilidad se ve disminuida con cada nueva versión de un producto software, pues cada versión complica la estructura del software y, por tanto, las futuras

[12] Desde el 2001 ha sido tutor virtual de la "Universidad Abierta de Cataluña", donde obtuvo un postgrado en Bioinformática. Obtuvo un doctorado en Informática por la "Universidad Carlos III" en el 2003. Actualmente es profesor del Departamento de Ciencias de la Computación de la "Universidad de Alcalá de Henares" (Madrid).

modificaciones son más difíciles. Este impacto en la flexibilidad puede disminuirse con la aplicación de técnicas y metodologías apropiadas.

1.7.2 Índice de Mantenibilidad (IM)

Existen maneras de medir la mantenibilidad para todos los elementos software que pueden estar sometidos a mantenimiento (código, documentos de usuario, documentos de análisis o diseño, entre otros). Existen modelos de regresión polinomial[13] que predicen la mantenibilidad del software. Para ello, se utiliza una combinación de variables de predicción en una ecuación polinomial para definir un Índice de Mantenibilidad (IM).

El IM mide la facilidad de mantenimiento del producto considerado. Toda acción de mantenimiento puede dividirse en 3 tareas:

- Comprensión de los cambios que deben hacerse.
- Realización de las modificaciones necesarias.
- Pruebas de los cambios realizados.

El IM expresa una fuerte correlación entre la métrica Halstead, la complejidad ciclomática, las líneas de código, y el número de observaciones al sistema que es objeto del mantenimiento de software.

El IM de un conjunto de programas es un polinomio de la siguiente forma:

IM = 171 – 5.2* ln (aveV) – 0.23 *aveV (g') -16.2 * ln (aveLOC) + 50 *sin (sqrt (2.4 * perCM))

Donde:
aveV: es la media del volumen por módulo según Halstead.
aveV(g'): es la media de la complejidad ciclomática por módulo que mide el número de caminos linealmente independientes a través de un módulo de programa.
(M = Número de condiciones + 1)
aveLOC: es la media del número de líneas de código por módulo.
perCM: es la media porcentual de líneas de código comentadas.

[13] La regresión polinomial examina la relación entre una variable continua de la respuesta (Y) y una variable del predictor (X). Es diferente de la regresión simple, sin embargo, un modelo polinomial puede incluir los términos para los exponentes de X**.**

Para realizar un análisis de si se ha elevado o no la mantenibilidad de un sistema tributando a su calidad se debe comparar el indicador IM, cuanto mayor sea el IM mayor mantenibilidad tiene el sistema. Los valores superiores a 85 indican que el software es muy mantenible, valores entre 85 y 65 sugieren moderado de mantenimiento, y los valores por debajo de 65 indican que el sistema es difícil de mantener.[16]

1.8 Conclusiones parciales

Del estudio realizado en la fundamentación teórica de la mantenibilidad en bases de datos realizadas en PostgreSQL se puede concluir que el mantenimiento de software es la primera abstracción a realizar en el proceso de evolución de un sistema. El mismo puede ser de tipo correctivo, preventivo, adaptativo y el que por lo general predomina es el perfectivo. Además, el proceso de mantenimiento se compone de actividades que pautan su desarrollo, estas van desde el análisis y la comprensión del sistema hasta su migración y retirada, por lo que la mantenibilidad hay que tenerla presente durante todas estas actividades. Una buena práctica para medir la mantenibilidad de los sistemas como característica de calidad de software es utilizar el Índice de Mantenibilidad.

2.1 Introducción

En este capítulo se describe la propuesta de la guía para obtener mantenibilidad en las bases de datos realizadas en PostgreSQL. Esta guía está basada en prácticas establecidas por los desarrolladores y en la referencia de estándares y normas estudiadas en el capítulo anterior. Se sustenta teniendo en cuenta las principales dificultades que se pueden encontrar al realizar mantenimiento de software y que imposibilitan la mantenibilidad del mismo.

2.1.1 Información general sobre la guía de buenas prácticas para obtener mantenibilidad de las bases de datos desarrolladas en PostgreSQL

La guía de buenas prácticas para obtener mantenibilidad en las bases de datos desarrolladas en PostgreSQL está estructurada en epígrafes: Diseño, Configuración, Arquitectura y Codificación respectivamente, que son los elementos a tener en cuenta en la investigación para obtener mantenibilidad en estas bases de datos. En los epígrafes anteriormente mencionados se exponen criterios sobre cómo se refleja la mantenibilidad en el proceso tratado, se enuncian un conjunto de buenas prácticas y ejemplos con la finalidad de elevar el factor de calidad tratado. También se realizan preguntas de control para comprobar los conocimientos adquiridos y una lista de chequeo para verificar hasta qué punto ponen en práctica los desarrolladores la guía propuesta. La siguiente figura muestra la estructura antes mencionada.

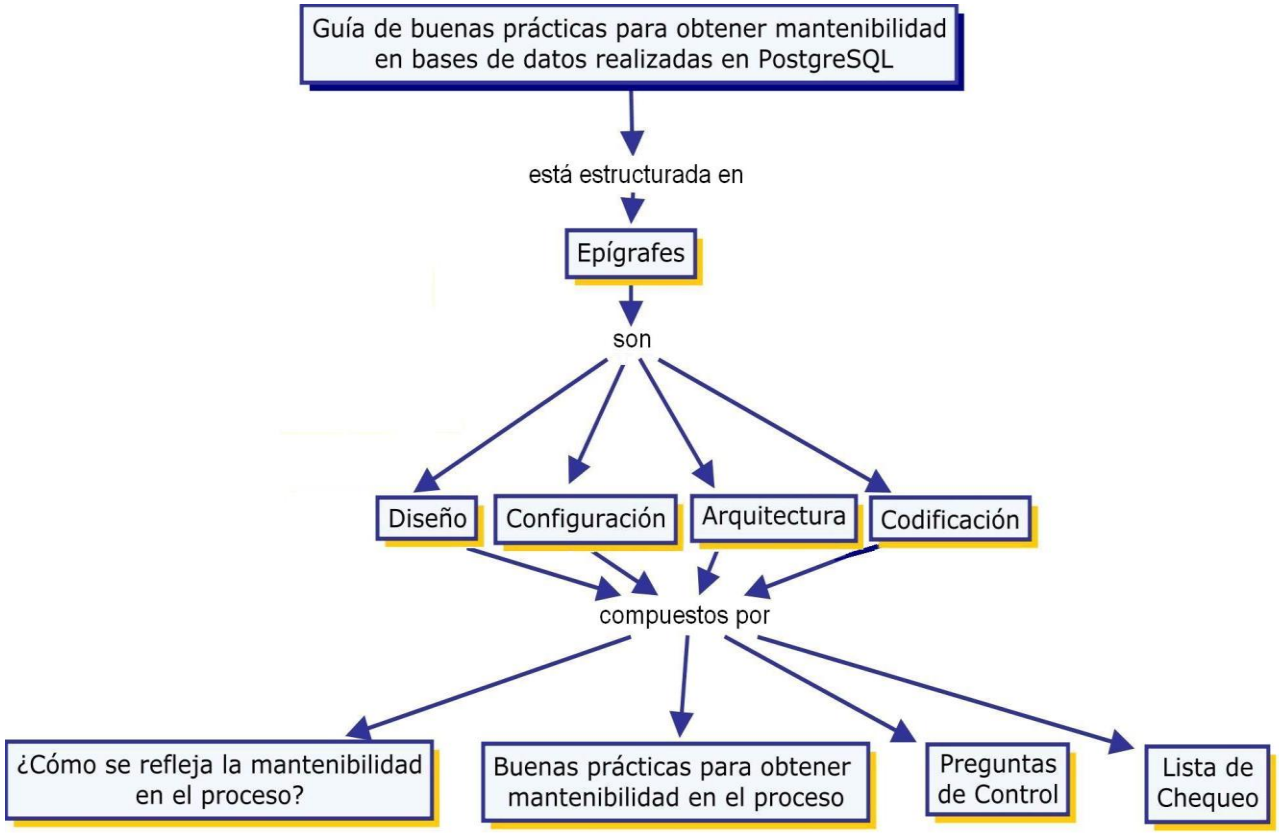


Figura12. Estructura de la Guía.

2.2 Diseño

2.2.1¿Cómo se refleja la mantenibilidad en el diseño?

El concepto de mantenibilidad implica que un programa debe ser capaz de adaptarse a un medio siempre cambiante. La tarea más importante consiste en realizar una copia de seguridad de los datos de forma rutinaria para que se pueda restaurar el sistema en caso de daños o pérdida de datos. También es importante garantizar que el tamaño de la base de datos no supere el espacio máximo de disco y que el espacio disponible en el disco sea suficiente para las necesidades. A pesar de estos requisitos básicos de mantenimiento, la base de datos sufre daños y el proceso de corregir la estructura de los datos es extremadamente difícil.

2.2.2 Buenas prácticas para lograr mantenibilidad en el diseño

2.2.2.1 Almacenar sólo la información necesaria

A menudo, se diseña y se almacena en una base de datos todos los datos necesarios o no. Es muy importante conocer las necesidades y decidir qué información es realmente necesaria. Frecuentemente se puede generar algunos datos sin tener que almacenarlos en una tabla de la base de datos, esto trae consigo que las bases de datos no se sobrecarguen con datos innecesarios y así se facilite su mantenimiento.

Ejemplo:

Una tabla de productos para un catálogo en línea puede contener nombres, descripciones, tamaños, pesos y precios de varios productos. Además del precio, se puede guardar los impuestos y los gastos de envío asociados con cada producto. Realmente no hay ninguna necesidad de hacer esto, por los siguientes motivos:

1. Los impuestos como los gastos de envío se pueden calcular (por la misma aplicación).
2. Si se cambian los impuestos o los gastos de envío, se tiene que escribir las búsquedas necesarias para actualizar los impuestos y los gastos de envío en cada registro del producto.

2.2.2.2 Normalizar/Desnormalizar las estructuras de tablas

Normalización

Una de las formas más fáciles de entender qué es la normalización de datos es pensar en las tablas como hojas de cálculo. Por ejemplo, si se quiere seguir la pista de una colección de CDs en una hoja de cálculo, se puede diseñar algo parecido a lo que se muestra en la siguiente tabla.

Álbum	Tema 1	Tema 2	……..	Tema 10
Calidad del Software	Principios de Calidad	Pruebas de Calidad	……	Auditorías de Calidad

Tabla #1.Colección de CDs.

Esto parece ventajoso, sin embargo, la dificultad es que el número de pistas que tiene un CD es bastante variable, lo que significa que con este método es necesario tener una hoja de cálculo realmente grande para guardar todos los datos, que no es muy favorable. Uno de los objetivos de una estructura de tabla

normalizada es minimizar el número de "celdas vacías". En el caso de que las listas de campos se expandan "hacia la derecha", es decir, que se incrementen como en este ejemplo de los CDs, se recomienda que se dividan los datos en dos o más tablas a las que luego se pueda acceder de forma conjunta para obtener los datos que se necesitan.

El objetivo principal del diseño de bases de datos es generar tablas que modelan registros en los que se permita guardar información. Es importante que esta información se almacene sin redundancia, para tener una recuperación rápida y eficiente de los datos. A través de la normalización se evitan ciertos defectos que conducen a un mal diseño y que llevan a un procesamiento menos eficaz de los datos.

Los objetivos principales de la normalización son los siguientes:

- Controlar la redundancia de la información.
- Evitar pérdidas de información.
- Capacidad para representar toda la información.
- Mantener la consistencia de los datos.

Desnormalización

Con el proceso de normalización se consigue evitar al máximo la redundancia de datos y permite realizar modificaciones de un modo cómodo. Para ello se ha indicado que se desglose la base de datos en tantas tablas como sea necesario.

La lógica y la experiencia del administrador son reglas que están por encima de cualquier otra, éstas no son obligatorias, son aconsejables y en muchos casos son de gran ayuda. El programador indica que siguiendo la normalización de la base de datos se puede conseguir desglosar la estructura en diferentes tablas con sus relaciones concernientes. Esto puede provocar que la búsqueda de un registro tenga que llevarse a cabo a través de varias tablas y relaciones, con un rendimiento indeseable. Para solucionar esto, el desarrollador lleva a cabo el proceso de desnormalización, que tiene consecuencias de redundancia de datos pero que es necesario para la mejora del rendimiento.

Para conseguir alcanzar el término medio entre el proceso de normalización y desnormalización, el mejor medio es la experiencia. Se expone la base de datos en explotación como prueba piloto y se analiza la actividad que se realiza sobre ella, se estudia si los resultados se adaptan a las necesidades y cumplen

con el rendimiento esperado, sino es así, gracias a estos estudios se puede observar que es necesario realizar modificaciones para mejorar el diseño.

2.2.2.3 Utilizar índices apropiados (Reindexación)

Los índices son un sistema especial que utilizan las bases de datos para mejorar su rendimiento global. Los índices hacen que las consultas se ejecuten más rápido y esto hace que se desee indexar todas las columnas de las tablas. Sin embargo, lo que se debe conocer es que utilizar los índices provoca dificultades. Cada vez que se hace un INSERT, UPDATE, REPLACE, o DELETE sobre una tabla, hay que actualizar cualquier índice en la tabla para reflejar los cambios en los datos.

En algunas ocasiones puede haber dificultades con los índices, ya sean causados por problemas de hardware o software, o puede que estos sean incorrectos. En estos casos es necesario realizar la reindexación. Aquí se elimina los datos muertos (aquellas que no son usados) y se vuelve a reasignar los índices de manera que estos optimicen la búsqueda de los datos y mejoren el rendimiento de los servidores. El espacio en disco requerido para almacenar el índice es típicamente menor que el espacio de almacenamiento de la tabla, pues los índices generalmente solo contienen los campos clave de acuerdo con los que la tabla es ordenada y excluyen el resto de los detalles de la tabla.

Ventajas

- Cuando existe un índice se evita un “escaneo completo de la tabla” lo que hace que cuando haya grandes cantidades de datos en las tablas la mejora puede ser muy importante.
- El escaneo completo de la tabla evita los siguientes problemas: sobrecarga de CPU, sobrecarga de disco y concurrencia.
- Con los índices se evita realizar lecturas secuenciales.

Desventajas

A pesar de sus ventajas, no se debe abusar de los índices puesto que en determinadas situaciones no admiten una mejora:

- Son una desventaja en aquellas tablas que se utilizan frecuentemente operaciones de escritura (INSERT, UPDATE, o DELETE), pues los índices se actualizan cada vez que se modifica una columna.

- Tienen una desventaja en tablas demasiado pequeñas puesto que no se necesita ganar tiempo en las consultas.
- No son muy aconsejables cuando se pretende que la tabla sobre la que se aplica devuelva una gran cantidad de datos en cada consulta.
- Ocupan espacio y en determinadas ocasiones más que los propios datos.

2.2.2.4 Utilizar patrones de diseño

A) Patrón Visitor

Este patrón representa una operación sobre elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera. Brinda una forma sencilla y mantenible de realizar acciones sobre una familia de clases.

Ejemplo:

Un compilador interpreta código fuente y lo representa como un árbol de sintaxis abstracta (Abstract Syntax Tree, AST), el cual cuenta con diversos tipos de nodos (asignaciones, expresiones condicionales, entre otros). Sobre este árbol se desea ejecutar algunas operaciones (revisar que todas las variables fueron declaradas, chequeos de tipos de datos y generación de código). Una forma de realizar estas operaciones es mediante la implementación del patrón Visitor, el cual recorre toda la estructura del árbol. Cuando un nodo acepte al Visitor, éste invoca al método de visita definido en el Visitor que toma por parámetro al nodo siendo visitado.

B) Patrón Interpreter

En un contexto donde se repita una determinada clase de problemas y el dominio es bien conocido, se puede caracterizar estos problemas como un lenguaje y, a su vez, estos problemas pueden ser tratados por un "motor" de interpretación. Este patrón busca definir un intérprete para dicho lenguaje, el cual define una gramática y un intérprete de la misma y así resolver los problemas.

Ejemplo:

Distintos motores de bases de datos (Oracle, SQL Server, Sybase, DB2 y PostgreSQL) utilizan distintos códigos de error para indicar fallas (errores de clave duplicada, violación de restricciones de integridad referencial y longitud de datos). La utilización de este patrón permite definir un intérprete de errores para cada motor de base de datos con el cual se determina la falla y se toman las acciones pertinentes en

función de la misma. El sistema se debe configurar para utilizar el intérprete adecuado según el motor de base de datos.

2.2.3 Preguntas de control

Todas las respuestas de estas preguntas se pueden encontrar en el Anexo 1.

1. Son muchas las consideraciones a tener en cuenta en el momento para realizar el diseño de la base de datos, entre estas se pueden encontrar:
   - La velocidad de acceso.
   - El tipo de la información.
   - Facilidad para extraer la información requerida.

a) ¿Considera usted que estas sean las únicas?

b) ¿Cree que se deba agregar otros ejemplos?

2. Marque Verdadero o Falso (V o F).

Un buen diseño de base de datos es aquel que:

_____ Divide la información en tablas basadas en temas para reducir los datos redundantes.

_____ Proporciona la información necesaria para reunir la misma de las tablas cuando así se precise.

_____ Ayuda a garantizar la exactitud e integridad de la información.

_____ Satisface las necesidades de procesamiento de los datos y de generación de informes.

3. ¿Ignorar la Normalización es fundamental para obtener un buen rendimiento y una sencilla programación en el desarrollo de una base de datos? Argumente su respuesta.

4. ¿Puede afirmar que los aspectos citados a continuación forman parte de los objetivos principales de la Normalización?
   - Controlar la redundancia de la información.
   - Evitar pérdidas de información.
   - Capacidad para representar toda la información.
   - Mantener la consistencia de los datos.

5. ¿Cree que es importante utilizar índice en el manejo o trabajo con la base de datos? Argumente su respuesta.

2.2.4 Lista de chequeo

En la siguiente tabla se puede observar la lista de chequeo que se propone en la investigación.

Nivel	Criterio de evaluación	Evaluación	N.P.	Observaciones
	Diseño			
	➔ Utilizar normalización/desnormalización.			
	1. ¿Utiliza la normalización/desnormalización en su base de datos?			
	➔ Almacenar solo la información necesaria.			
	1. ¿En su base de datos almacena solo la información necesaria?			
	2. ¿Los datos que pueden ser calculados por la aplicación los tiene incluido en la base de datos?			
	➔ Utilizar índices apropiados.			
	1. ¿Utiliza los índices de forma apropiada?			
	2. ¿Tiene pocos índices definidos en una tabla en la que se realizan muchas actualizaciones?			
	3. ¿El rendimiento del trabajo con las tablas de la base de datos y acorde a lo deseado?			
	4. ¿Los índices son utilizados donde realmente se explote su uso?			
	➔ Utilizar patrones de diseños.			
	1. ¿Utiliza patrones de diseño para la confección de su base de datos?			
	5. ¿Utiliza el patrón Visitor?			

	6. ¿ Utiliza el patrón Interprete?			

Tabla #2.Lista de Chequeo para el Diseño.

Donde:

- En el **Nivel** solo se define cuando existen aspecto de mayor importancia que otros, para ello se usarán signos de exclamación (!).
- En el **Criterio de evaluación** se ubica el aspecto a evaluar, siempre con una redacción nítida.
- **Evaluación** es para el caso de la aplicación de la Lista de Chequeo. Dicha evaluación se encontrará en el rango de 0-5.
- **N.P.** significa No Procede y es para el caso del aspecto que no sea factible cuando se valore en la aplicación.
- **Observaciones** son para cualquier elemento que quiera incluir la persona que aplica dicha lista.

2.3 Configuración

2.3.1 ¿Cómo se refleja la mantenibilidad en la configuración?

Configuración consiste en adaptar una aplicación de software o un elemento hardware al resto de los elementos del entorno y a las necesidades específicas del usuario, de esta manera, se pone en práctica el mantenimiento adaptativo y la necesidad de facilitar su realización. Esta es una tarea esencial antes de trabajar con cualquier nuevo elemento. La tendencia actual es reducir las necesidades de configuración mediante sistemas que permiten al nuevo elemento detectar en qué entorno se instala, configurándose automáticamente sin requerir la participación del usuario. Cuando esta es necesaria, se intenta facilitar al máximo el proceso de configuración.

PostgreSQL desde su versión 8.0 cuenta con una configuración básica mejorada para una gran compatibilidad en vez de un gran rendimiento. Muchas de las personas que utilizan PostgreSQL, ya sea a nivel particular o en proyectos de explotación, desconocen que este formidable servidor parte inicialmente de una configuración mínima. Esto hace que el servidor arranque con cualquier configuración hardware, aunque no es válida para gestionar las bases de datos de manera eficiente.

Se puede mejorar el rendimiento de PostgreSQL desde dos elementos:

Hardware. Configurar y optimizar los recursos hardware con que va a disponer el servidor, por ejemplo la memoria RAM.
Software. Utilizar técnicas basadas en software como el uso de índices, optimización de consultas y vacuuming.

Es muy importante comenzar por ajustar el hardware para una vez optimizado, solo tener en cuenta la afinación del software. Es necesario recordar el papel que juega la memoria RAM en un ordenador, así como los efectos indeseados de la paginación.

La RAM se puede observar como un recurso limitado dividido en rodajas o segmentos. Los segmentos de un tamaño fijo se agrupan formando páginas de memoria. En la memoria se guarda todo lo que el CPU necesita para hacer su trabajo, esto incluye programas, datos requeridos por los programas, el Kernel y las zonas de trabajo de PostgreSQL. Para optimizar el uso del espacio disponible en la memoria las páginas que hace algún tiempo no se utilizan son expulsadas por el Sistema Operativo al disco a una zona denominada "swap[14]". Esta actividad se denomina "swap pageout" y no presenta inconveniente, pues se produce en períodos de inactividad del CPU. El inconveniente existe cuando hay que recuperar una página desde la swap (que recientemente ha sido expulsada de la memoria), pues el programa que la requiere tiene que esperar hasta que se encuentre de nuevo allí. Este efecto adverso crece a medida que hay más páginas que se necesitan traer de la swap.

Lo difícil consiste en optimizar el uso de memoria para PostgreSQL minimizando en lo posible el número de intercambios con la swap, en algunos casos hay que tener en cuenta que esto depende de las peticiones que reciba el servidor, si es un servidor de mapas o un servidor de Data WareHouse[15] el paginado siempre se utiliza, porque estos manipulan hasta Tb (Tera Bytes) de información en consultas simples. El mejor ajuste de los parámetros de configuración es aquel que obtiene la máxima disponibilidad en memoria para la BD, sin perjudicar al resto de los elementos que también deben permanecer en memoria.

[14] Se conoce con el nombre swap pagein o paginación.

[15] Es el lugar donde están todos los datos de la empresa, a los que se puede acceder de forma fácil, eficiente y sin esfuerzo.

2.3.2 Buenas prácticas para lograr mantenibilidad en la configuración

2.3.2.1 Realizar configuración del Shared_Buffers

El número de "*shared_buffers"* es el parámetro que más afecta al rendimiento de PostgreSQL. Este valor de tipo entero indica el número de bloques de memoria o buffers de 8KB (8192 bytes) que PostgreSQL reserva como zona de trabajo en el momento del arranque para procesar las consultas. De forma predeterminada (en *postgresql.conf*), su valor es de 1000. Un número claramente insuficiente para conseguir un rendimiento mínimo aceptable. Estos buffers se ubican dentro de los denominados segmentos de memoria compartida. Es importante saber que el espacio ocupado por el número de buffers que se pretenda asignar nunca puede exceder al tamaño máximo que tienen los segmentos de memoria, en caso contrario, PostgreSQL se niega a arrancar avisando con un error que no puede reservar el espacio solicitado.

Es necesario tener en cuenta que si existe un sistema con 1GB o más de RAM, un valor inicial razonable es un 1/4 de dicha memoria. Si tiene menos se debe calcular cuidadosamente este valor de acuerdo con el sistema operativo, cercano al 15% en los casos más comunes. En Windows y versiones antiguas de PostgreSQL (anteriores a la 8.1), altos valores de "*shared_buffers"* no son efectivos, es un buen resultado mantenerlo relativamente bajo (alrededor de 50.000) y utilizar mejor el caché del sistema operativo. [17]

2.3.2.2 Realizar configuración del Work_Mem

Este parámetro configura el espacio de memoria que PostgreSQL utiliza para realizar ordenaciones de tablas o de resultados parciales de consultas, sobre todo en cláusulas como ORDER BY, CREATE INDEX o MERGE JOIN. Este valor es más difícil de configurar porque depende de lo grande que sean las tablas o resultados que hay que ordenar y del número de peticiones simultáneas para esa misma consulta (para cada una se emplea la misma cantidad de memoria).

Un buen comienzo es asignar entre un 2% y un 4% del total de la memoria si se provee pocos accesos simultáneos a grandes sesiones de orden y mucho menor si se espera varios accesos simultáneos a sesiones de orden pequeño. Lo mejor es ir probando distintos valores y ver en qué pueden afectar a la paginación adversa. Se recomienda que el valor se debe expresar en KB. [17]

2.3.2.3 Realizar configuración del Autovacuum

El proceso de VACUUM lo que realiza es una limpieza de tuplas muertas que han sido marcadas como borradas o modificadas, pues el motor de la base de datos no las borra inmediatamente de la parte física para no sobrecargar las operaciones normales. El proceso automático de Vacuum (autovacuum) realiza una serie de operaciones de mantenimiento en la base que se necesite. En la versión 8.3 está activo por defecto y se debe dejar así, sin embargo, en la 8.1 y 8.2 se debe activar. Es necesario tener en cuenta que en estas versiones antiguas se debe configurar sus variables para que tenga un efecto más rotundo, puede que no haga el trabajo suficiente por defecto si se tiene una base de datos muy grande o realice muchas actualizaciones en los datos. [18]

2.3.2.4 Poseer un manual de configuración

Al no contar con una documentación disponible correcta los costes de mantenimiento de software se incrementan debido al tiempo que necesita el mantenedor para comprender el diseño. Para facilitar el mantenimiento del software se hace necesario contar con un correcto manual de configuración. En las bases de datos desarrolladas en PostgreSQL de DATEC se puede aumentar la mantenibilidad utilizando dicho manual. Este debe contar con las políticas para realizar la configuración, es decir, se debe especificar cómo se configura el servidor según el tipo de aplicación que se requiera configurar. Se recomienda que se realice un estándar para cada manual de configuración teniendo en cuenta el servidor según el tipo de aplicación que se desee configurar.

2.3.3 Preguntas de control

Las respuestas a estas preguntas se pueden encontrar en el Anexo 2.

1. El número de "*shared_buffers"* es el parámetro que más afecta al rendimiento de PostgreSQL. Este valor indica el número de bloques de memoria o buffers que PostgreSQL reserva como zona de trabajo en el momento del arranque para procesar las consultas. Sabe usted:
   a) ¿Cuántos "shared_buffers" se pueden asignar?
   b) ¿Cómo sabe cuál es el tamaño de un segmento?
   c) ¿Qué puede hacer si se supera el tamaño máximo del segmento?

2. ¿Cuál cree usted que sería el papel del proceso automático de Vacuum (autovacuum)? ¿Lo considera importante? Argumente su respuesta.

2.3.4 Lista de chequeo

En la siguiente tabla se puede observar la lista de chequeo que se propone en la investigación.

Nivel	Criterio de evaluación	Evaluación	N.P.	Observaciones
	Configuración			
	➔ Configurar el Autovacuum.			
	1. ¿ Utiliza una configuración de las variables para que tenga un efecto más rotundo el proceso automatico del Vacum?			
	➔ Configurar el “Shared_ Buffers”.			
	1. ¿Para un sistema de 1GB o más de RAM configura el “Shared_ Buffers” dándole un valor inicial de ¼ de dicha memoria?			
	➔ Configurar el “Work_Mem”.			
	1. ¿Utiliza una configuración del “Work_Mem” entre un 2 y un 4% del total de su memoria?			
	➔ Utilizar un manual de configuración.			
	1. ¿Utiliza un manual de configuración?			

Tabla #3.Lista de Chequeo para la Configuración.

Donde:

- En el **Nivel** solo se define cuando existen aspecto de mayor importancia que otros, para ello se usarán signos de exclamación (!).
- En el **Criterio de evaluación** se ubica el aspecto a evaluar, siempre con una redacción nítida.
- **Evaluación** es para el caso de la aplicación de la Lista de Chequeo. Dicha evaluación se encontrará en el rango de 0-5.
- **N.P.** significa No Procede y es para el caso del aspecto que no sea factible cuando se valore en

la aplicación.

- **Observaciones** son para cualquier elemento que quiera incluir la persona que aplica dicha lista.

2.4 Arquitectura

2.4.1 ¿Cómo se refleja la mantenibilidad en la arquitectura?

La mantenibilidad es una característica de calidad interna, por lo que tiene relación directa con la arquitectura del software, pues la misma presenta una organización de sus relaciones y componentes de manera que se favorezcan o se castiguen diferentes atributos. Esta organización de componentes proporciona la facilidad del software para ser cambiado, para ser probado y para adaptarse de manera estable a los cambios. Dentro de las decisiones arquitectónicas que pueden influir en la calidad del software se encuentra el uso de diferentes mecanismos, tales como los patrones y estilos arquitectónicos, al igual que los patrones de diseño. Los patrones plantean soluciones probadas a problemas recurrentes y los estilos arquitectónicos proponen una forma de agrupar los componentes de la arquitectura, de todos estos se han identificado sus ventajas en cuanto a propiciar la característica de calidad mencionada.

2.4.2 Buenas prácticas para lograr mantenibilidad en la arquitectura

2.4.2.1 Utilizar el patrón Modelo Vista Controlador (MVC o Model-View-Controller)

El patrón arquitectónico Modelo-Vista-Controlador (MVC) fue introducido inicialmente en la comunidad de desarrolladores de Smalltalk[16]-80. Es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. A continuación se presenta una breve explicación de cada uno de ellos.

- *Modelo (Model).* Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y comportamiento de entrada. Es decir, representa la información con la que trabaja la aplicación, en otras palabras, su lógica de negocio.
- *Vista (View).* Muestra la información al usuario. Obtiene los datos del modelo. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador, es decir, transforma el modelo en una página web que permite al usuario interactuar con ella.

[16] Smalltalk es un lenguaje de programación que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales.

- *Controlador (Controller).* Recibe las entradas, usualmente como eventos que codifican los movimientos o pulsación de botones del mause, pulsaciones de teclas, entre otros. Los eventos son traducidos a solicitudes de servicio para el modelo o la vista. El usuario interactúa con el sistema a través de los controladores. Es decir, se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista. La figura 13 ilustra el funcionamiento de patrón MVC.

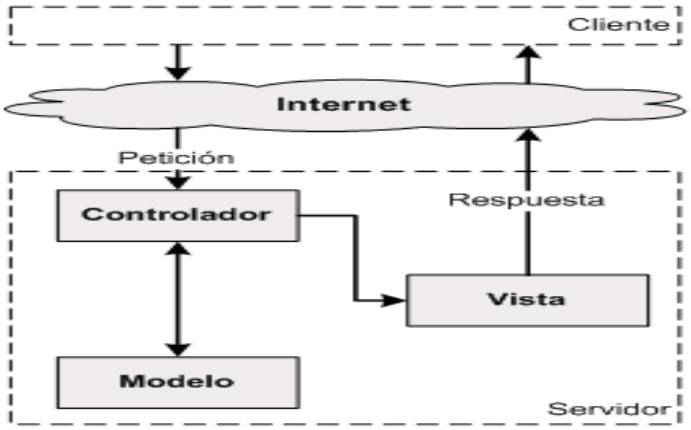


Figura13. Funcionamiento del patrón MVC.

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista), por lo que se consigue un mantenimiento más fácil y sencillo de las aplicaciones. Si por ejemplo una misma aplicación debe ejecutarse tanto en un navegador estándar como en un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo manteniendo el controlador y el modelo original. El controlador se encarga de aislar al modelo y la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, entre otros). El modelo se encarga de la abstracción de la lógica relacionada con los datos, hace que la vista y las acciones sean independientes, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación.

Este patrón es muy popular y ha sido aplicado a gran cantidad de entornos y frameworks, como ASP y Symfony. Las herramientas de programación visual como Visual Basic, Visual Studio y .Net siguen también alguna variante de este esquema. El MVC es un patrón ampliamente utilizado en múltiples plataformas y lenguajes. Algunos de sus principales beneficios son:

- Menor acoplamiento.
  - ✓ Desacopla las vistas de los modelos.
  - ✓ Desacopla los modelos de la forma en que se muestran e ingresan los datos.
- Mayor cohesión.
  - ✓ Cada elemento del patrón está altamente especializado en su tarea (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio).
- Las vistas proveen mayor flexibilidad y agilidad.
  - ✓ Se pueden crear múltiples vistas de un modelo.
  - ✓ Se pueden crear, añadir, modificar y eliminar nuevas vistas dinámicamente.
  - ✓ Las vistas pueden anidarse.
  - ✓ Se pueden cambiar el modo en que una vista responde al usuario sin cambiar su representación visual.
  - ✓ Se puede sincronizar las vistas.
  - ✓ Las vistas pueden concentrarse en diferentes aspectos del modelo.
- Mayor facilidad para el desarrollo de clientes ricos en múltiples dispositivos y canales.
  - ✓ Una vista para cada dispositivo que puede variar según sus capacidades.
  - ✓ Una vista para la Web y otra para aplicaciones de escritorio.
- Más claridad de diseño.
- Facilita el mantenimiento.
- Mayor escalabilidad.

Ejemplo:

Para poder entender las ventajas de utilizar el patrón MVC, en el siguiente ejemplo se puede observar cómo una aplicación simple realizada con PHP se transforma en una aplicación que sigue la arquitectura MVC. Un buen ejemplo para ilustrar esta explicación es el de mostrar una lista con las últimas entradas o artículos de un blog.

La figura 14 muestra el script necesario para mostrar los artículos almacenados en una base de datos utilizando solamente PHP. [19]

```
<?php

// Conectar con la base de datos y seleccionarla
$conexion = mysql_connect('localhost', 'miusuario', 'micontrasena');
mysql_select_db('blog_db', $conexion);

// Ejecutar la consulta SQL
$resultado = mysql_query('SELECT fecha, titulo FROM articulo', $conexion);

?>

<html>
  <head>
    <title>Listado de Artículos</title>
  </head>
  <body>
   <h1>Listado de Artículos</h1>
   <table>
     <tr><th>Fecha</th><th>Titulo</th></tr>
<?php
// Mostrar los resultados con HTML
while ($fila = mysql_fetch_array($resultado, MYSQL_ASSOC))
{
echo "\t<tr>\n";
printf("\t\t<td> %s </td>\n", $fila['fecha']);
printf("\t\t<td> %s </td>\n", $fila['titulo']);
echo "\t</tr>\n";
}
?>
    </table>
  </body>
</html>

<?php

// Cerrar la conexion
mysql_close($conexion);

?>
```

Figura14. Un script simple.

El script que se muestra en la figura anterior es fácil de escribir y rápido de ejecutar, pero muy difícil de mantener y actualizar. Las llamadas a *"echo"* y *"printf"* dificultan la lectura del código, de hecho, modificar el código HTML del script anterior para mejorar la presentación es muy difícil debido a como está programado, por tanto es necesario que el código se divida en dos partes. En primer lugar, el código PHP puro con toda la lógica de negocio se incluye en el script del controlador, como se muestra en la figura 15.

```php
<?php

// Conectar con la base de datos y seleccionarla
$conexion = mysql_connect('localhost', 'miusuario', 'micontrasena');
mysql_select_db('blog_db', $conexion);

// Ejecutar la consulta SQL
$resultado = mysql_query('SELECT fecha, titulo FROM articulo', $conexion);

// Crear el array de elementos para la capa de la vista
$articulos = array();
while ($fila = mysql_fetch_array($resultado, MYSQL_ASSOC))
{
  $articulos[] = $fila;
}

// Cerrar la conexión
mysql_close($conexion);

// Incluir la lógica de la vista
require('vista.php');

?>
```

Figura15. La parte del controlador.

El código HTML que contiene PHP se almacena en el script de la vista, como se muestra la figura 16.

```
<html>
  <head>
    <title>Listado de Artículos</title>
  </head>
  <body>
    <h1>Listado de Artículos</h1>
    <table>
      <tr><th>Fecha</th><th>Título</th></tr>
    <?php foreach ($articulos as $articulo): ?>
      <tr>
        <td><?php echo $articulo['fecha'] ?></td>
        <td><?php echo $articulo['titulo'] ?></td>
      </tr>
    <?php endforeach; ?>
    </table>
  </body>
</html>
```

Figura16. La parte de la vista.

Una buena regla general para determinar si la parte de la vista está suficientemente limpia de código es que esta debe contener una cantidad mínima de código PHP, la suficiente como para que un diseñador de HTML sin conocimientos en PHP pueda entenderla. Las instrucciones más comunes en la parte de la vista suelen ser "*echo"*," *if/endif"* y "*foreach/endforeach"*. Además, no se deben incluir instrucciones PHP que generen etiquetas HTML. Toda la lógica está centralizada en el script del controlador que solamente contiene código PHP y ningún tipo de HTML.

La mayor parte del script del controlador se encarga de la manipulación de los datos. Pero, ¿Y si se quieren centralizar todas las consultas a la base de datos en un único sitio para evitar duplicidades? ¿Qué ocurre si cambia el modelo de datos y la tabla "*articulo"* pasa a llamarse "*articulo_blog"*? Para poder hacer todo esto, es imprescindible eliminar del controlador todo el código que se encarga de la manipulación de los datos y ponerlo en otro script llamado modelo, tal y como se muestra en la figura 17.

```php
<?php

function getTodosLosArticulos()
{
  // Conectar con la base de datos y seleccionarla
  $conexion = mysql_connect('localhost', 'miusuario', 'micontrasena');
  mysql_select_db('blog_db', $conexion);

  // Ejecutar la consulta SQL
  $resultado = mysql_query('SELECT fecha, titulo FROM articulo', $conexion);

  // Crear el array de elementos para la capa de la vista
  $articulos = array();
  while ($fila = mysql_fetch_array($resultado, MYSQL_ASSOC))
  {
    $articulos[] = $fila;
  }

  // Cerrar la conexión
  mysql_close($conexion);

  return $articulos;
}

?>
```

Figura 17. La parte del modelo.

El controlador modificado se puede observar en la figura 18.

```php
<?php

// Incluir la lógica del modelo
require_once('modelo.php');

// Obtener la lista de artículos
$articulos = getTodosLosArticulos();

// Incluir la lógica de la vista
require('vista.php');

?>
```

Figura18. La parte del controlador modificada.

Ahora el controlador es mucho más fácil de leer, su única tarea es obtener los datos del modelo y pasárselos a la vista. En las aplicaciones más complejas el controlador se encarga de procesar las peticiones, las sesiones de los usuarios y la autenticación. El uso de nombres apropiados para las funciones del modelo hace que sea innecesario añadir comentarios al código del controlador.

El script del modelo solamente se encarga del acceso a los datos y puede ser reorganizado. Todos los parámetros que no dependen de la capa de datos (como por ejemplo los parámetros de la petición del usuario) se deben obtener a través del controlador y por tanto, no se puede acceder a ellos directamente desde el modelo. Las funciones del modelo se pueden reutilizar fácilmente en otros controladores.

El principio más importante de la arquitectura MVC es la separación del código del programa en tres capas, dependiendo de su naturaleza. La lógica relacionada con los datos se incluye en el modelo, el código de la presentación en la vista y la lógica de la aplicación en el controlador. De esta forma, las capas del modelo, la vista y el controlador se pueden subdividir en más capas.

2.4.2.2 Utilizar una arquitectura en capas

Aunque PostgreSQL en su nivel superior presenta una arquitectura cliente/servidor, la arquitectura de su servidor puede poseer un estilo en capas. De esta manera, se separan las responsabilidades y se alcanza una mayor independencia de lo datos almacenados. Garlan y Shaw[17] definen el estilo en capas como *"una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior"*.

En un estilo en capas los conectores se definen mediante los protocolos que determinan las formas de la interacción. Las restricciones topológicas del estilo incluyen una limitación rigurosa, que exige a cada capa operar sólo con capas adyacentes y a los elementos de una capa entenderse sólo con otros elementos de la misma. Si esta exigencia se debilita el estilo deja de ser puro, también se pierde la posibilidad de reemplazar totalmente una capa sin afectar a las restantes, disminuye la flexibilidad del conjunto y se complica su mantenimiento. En la figura 19 se muestran las diferentes capas.

1. Capa de datos.

[17] Autores del libro An introduction to Software Architecture.

2. Capa de negocios.
3. Capa de presentación.

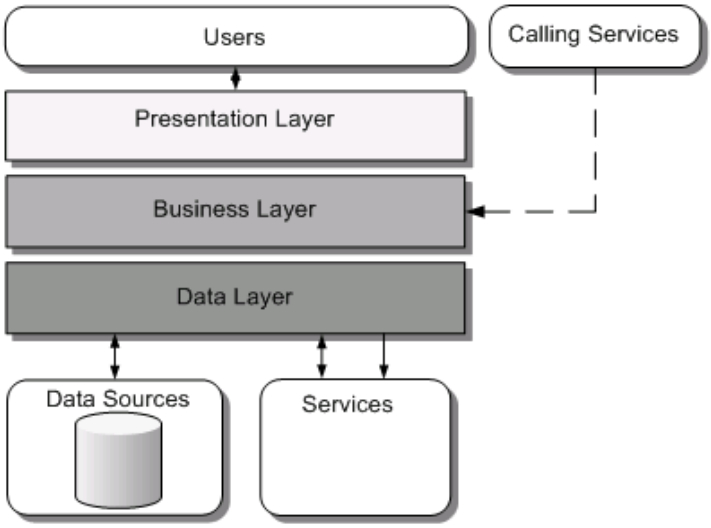


Figura19. Diferentes capas de la arquitectura.

A continuación se presenta una breve explicación de las capas anteriormente mencionadas.

1. *Capa de datos.* En esta capa se encuentra todos aquellos componentes cuya funcionalidad está centrada en recuperar o almacenar los datos con los que trabaja la aplicación. Los diferentes componentes que se pueden encontrar en ella son:

   - Base de datos.
   - Tablas.
   - Procedimientos almacenados.
   - Componentes de datos.

2. *Capa de negocio.* Es la que soporta toda la lógica de negocio. En esta capa se encuentra todas aquellas funciones que hacen algún tipo de tratamiento de los datos y se aplican las reglas de negocio. Los diferentes componentes que puede presentar esta capa son:

   - Reglas del negocio.

- Validaciones.
- Cálculos.
- Flujos y procesos.

3. *Capa de presentación.* Está orientada a soportar la interactividad de los usuarios con las funcionalidades brindadas por la capa de negocio. En esta capa se encuentran:

   - Formularios.
   - Informes.
   - Respuestas al usuario.

Ventajas de la arquitectura en capas

Las ventajas que puede proporcionar esta arquitectura son las siguientes:

- Modularidad.
- Soporta un diseño basado en niveles de abstracción crecientes lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- Proporciona amplia reutilización.
- Soporta fácilmente la evolución del sistema y los cambios sólo afectan a las capas vecinas.
- Se pueden cambiar las implementaciones respetando las interfaces con las capas adyacentes.

Ejemplo:

Al conectarse a internet se navega en capas, como se muestra en la figura 20.

- Al abrir un formulario web de inscripción (*capa de presentación*).
- Después de enviar la información esta es verificada (*capa de negocios*).
- Finalmente, la información es grabada en una base de datos (*capa de datos*).

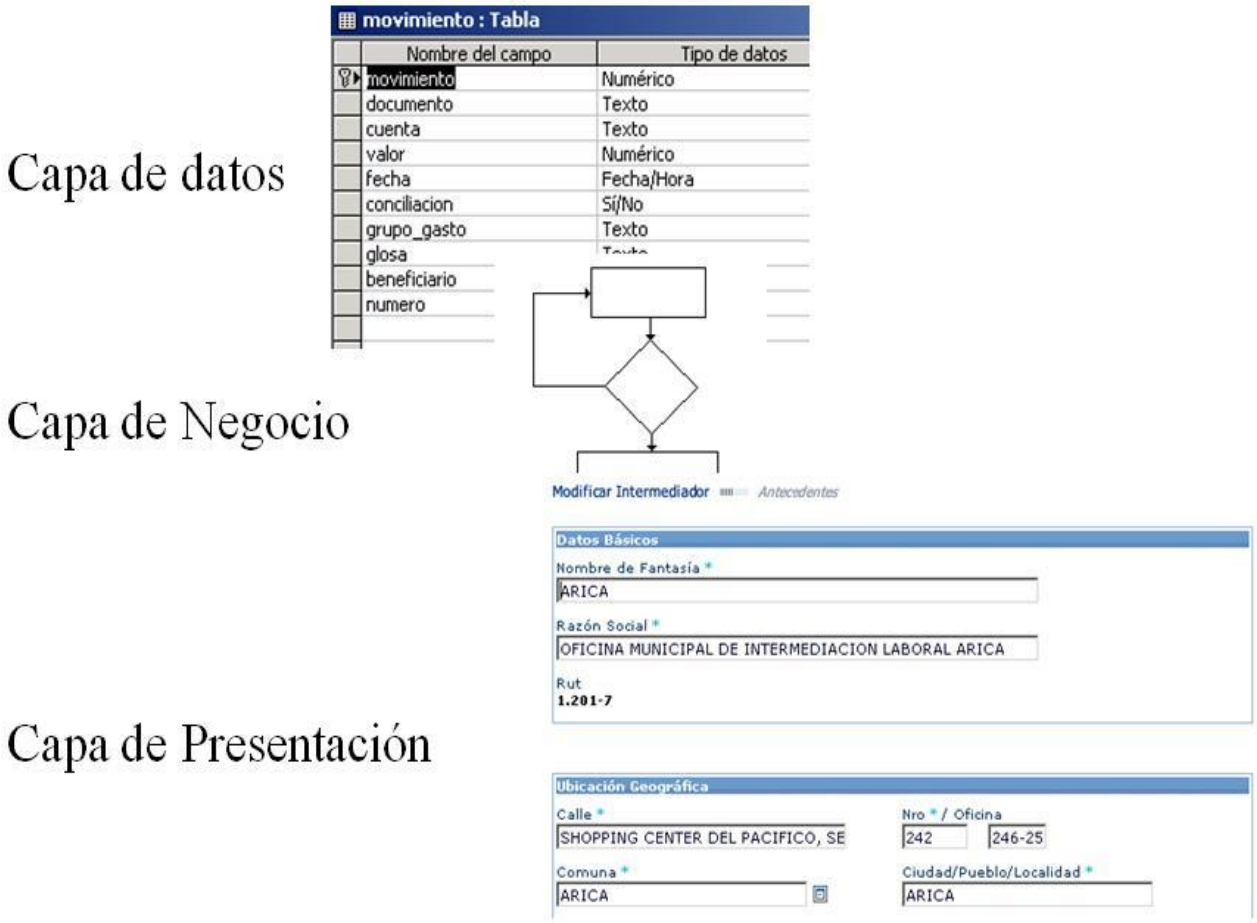


Figura20. Ejemplo de la arquitectura en tres capas.

2.4.2.3 Realizar el trabajo por módulos

El trabajo por módulo posee varias ventajas, entre las cuales la más importante es la facilidad con que se puede realizar el mantenimiento el software. Esta facilidad se establece cuando ocurre alguna falla, pues la misma es fácil de encontrar debido al comportamiento de los módulos. Es decir, el trabajo modular permite descomponer un problema en varios subproblemas independientes entre sí, más sencillos de resolver y que pueden ser tratados de forma separada unos de otros. Gracias a la modularidad se pueden probar los subprogramas o módulos de manera independiente, depurándose sus errores antes de su inclusión en el programa principal y almacenarse para su posterior utilización cuantas veces se precise.

Un programa principal llama a estos módulos a medida que se necesitan. Un módulo es un segmento, rutina, subrutina, subalgoritmo o procedimiento, que puede definirse dentro de un algoritmo con el fin de ejecutar una tarea específica y puede ser llamado o invocado desde el proceso principal cuando sea necesario. Los módulos son independientes en el sentido de que ninguno puede tener acceso directo a cualquier otro, con excepción del módulo al que llama y sus propios submódulos. Sin embargo, los resultados producidos pueden ser utilizados por cualquier otro cuando se transfiera a ellos el control. Se pueden tomar decisiones dentro de un módulo que tenga repercusión en todo el flujo, pero el salto debe ser únicamente hacia el programa principal. Al descomponer un programa en módulos independientes más simples se conoce también como el método de "Divide y Vencerás".

Ventajas del trabajo modular

Como los módulos son independientes, el desarrollo de un programa se puede efectuar con mayor facilidad, pues cada uno de ellos se puede crear aislado y varios desarrolladores pueden trabajar simultáneamente en la confección de un algoritmo repartiéndose las distintas partes del mismo, además se puede modificar un módulo sin afectar a los demás. El uso de módulos facilita la proyección y la comprensión de la lógica subyacente para el programador y el usuario. Aumenta la facilidad de depuración y búsqueda de errores en un programa, lo que permite realizar modificaciones y mantenimiento de una forma más sencilla.

2.4.3 Preguntas de control

Las respuestas a estas preguntas se pueden encontrar en el Anexo 3.

1. Responda Verdadero o Falso (V o F).

   ____ El uso de patrones y estilos arquitectónicos afectan negativamente la mantenibilidad del software.

   ____ Utilizar el Patrón Modelo Vista Controlador (MVC) aumenta la mantenibilidad.

   ____ El uso de la Arquitectura en Capas propicia la facilidad del software para ser cambiado, para ser probado y para adaptarse de manera estable a diferentes cambios.

   ____ El uso del trabajo por Módulo propicia la facilidad del software para ser cambiado, para ser probado y para adaptarse de manera estable a diferentes cambios.

2. En un proyecto productivo de la Universidad se cuenta con sistemas que están sujetos a una dinámica de mantenimiento constante, adaptándose para satisfacer nuevas necesidades y soportar nuevas tecnologías. Todo esto provoca que más de la mitad del esfuerzo de desarrollo de la organización se destine a este mantenimiento. Un desarrollador propone a sus compañeros realizar un trabajo por módulo para facilitar el mantenimiento que tiene que realizar.
   ¿Cree usted que la recomendación de este desarrollador puede ser satisfactoria? Argumente su respuesta.

2.4.4 Lista de chequeo

En la siguiente tabla se puede observar la lista de chequeo que se propone en la investigación.

Nivel	Criterio de evaluación	Evaluación	N.P.	Observaciones
	Arquitectura			
	➔ Debe utilizar patrones y estilos arquitectónicos.			
	1 ¿Separa usted los datos de una aplicación, la interfaz de usuario y la lógica de control en tres o más componentes distintos?			
	2 ¿El modelo se encarga de la abstracción lógica de los datos?			
	3 ¿Divide el trabajo por módulos?			
	4 ¿Utiliza una arquitectura en capas?			

Tabla #4.Lista de Chequeo para la Arquitectura.

Donde:

- En el **Nivel** solo se define cuando existen aspecto de mayor importancia que otros, para ello se usarán signos de exclamación (!).
- En el **Criterio de evaluación** se ubica el aspecto a evaluar, siempre con una redacción nítida.
- **Evaluación** es para el caso de la aplicación de la Lista de Chequeo. Dicha evaluación se encontrará en el rango de 0-5.
- **N.P.** significa No Procede y es para el caso del aspecto que no sea factible cuando se valore en

la aplicación.

- **Observaciones** son para cualquier elemento que quiera incluir la persona que aplica dicha lista.

2.5 Codificación

2.5.1 ¿Cómo se refleja la mantenibilidad en la codificación?

La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores y mejorar el rendimiento. Aunque la misma es el resultado de muchos factores, un elemento del desarrollo de software en el que todos los programadores influyen especialmente es en la técnica de codificación. Muchas veces se olvida que el código va a tener que ser interpretado, leído y mantenido por otras personas o equipos de desarrollo, incluso por el mismo programador en un futuro.

Mantener estructuras similares para programar, un estilo de programación, un orden determinado y un código claro, hacen que la mantenibilidad de las aplicaciones se eleve rápidamente. El mejor método para asegurarse de que un equipo de programadores mantenga un código con calidad, elevando la mantenibilidad del mismo, es establecer un estándar de codificación. Si se aplica de forma continua dicho estándar bien definido y posteriormente, se efectúan revisiones del código, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

2.5.2 Buenas prácticas para lograr mantenibilidad en la codificación

El conjunto de buenas prácticas propuestas en la investigación hace referencia a la variante "*lowerCamelCase, camelCase o dromedaryCase*" de la Notación "CamelCasing[18]" y están basadas en experiencias de los desarrolladores de bases de datos realizadas en PostgreSQL y en técnicas para

[18] La notación Camel consiste en escribir los identificadores con la primera letra de cada palabra en mayúsculas y el resto en minúscula. Se llama notación "Camel" porque los identificadores recuerdan las jorobas de un camello. Existen dos variantes:

- ✓ UpperCamelCase, CamelCase o PascalCase: en esta variante la primera letra también es mayúscula.
- ✓ lowerCamelCase, camelCase o dromedaryCase: la primera letra es minúscula.

escribir un código mantenible realizado por diferentes especialistas, como son: Josh Berkus, Leo Hsu, Regina Obe y Hubert Lubaczewski.

Las técnicas de codificación están divididas en tres secciones:

1. Nombrado.
2. Documentación Interna (*Comentarios*).
3. Formato.

A continuación se especifican un conjunto de buenas prácticas para elevar la mantenibilidad en la codificación desde estas tres secciones.

2.5.2.1 Estándar para el nombrado

El nombre es una de las ayudas más importantes para entender el flujo lógico de una aplicación. Se recomienda seguir la siguiente guía para realizar nombrados:

1. Un nombre debe expresar el "qué" y no el "cómo". La implementación está sujeta a cambios, por eso se debe utilizar un nombre que no se refiera a la implementación para que conserve la abstracción de la estructura. Lo más importante para que otra persona entienda el código es saber qué hace el método, la función, etc. y no cómo lo hace.

```
Ejemplo: Es más claro:
         seleccionarExpediente()

         Que:
         realizarConsultaSelect()
```

2. Los nombres deben ser largos para que se entiendan, pero a la vez cortos para que no brinden información irrelevante. Se recomienda que sean menores de 32 caracteres y para nombres relativamente largos utilizar abreviaturas para mantener la longitud de los nombres dentro de un límite razonable, éstas deben ser a lo largo de toda codificación y no usadas aleatoriamente.

```
Ejemplo: Es mejor:
         seleccionarSexo()

         Que:
         seleccionarElSexoDelMenu()
```

3. Para nombrar los procedimientos se debe utilizar la técnica verbo_sustantivo.

   Ejemplo: seleccionarSexo()

4. Todas las primeras letras de los nombres de variables, tablas, funciones, parámetros, etc. deben comenzar con minúsculas, para el caso de las variables o tablas que cuyo nombre conste de más de una palabra la primera letra del segundo nombre será en mayúscula (*Notación CamelCasing*)

   Ejemplo: nombre
   edadEstudiante

5. Las variables o funciones booleanas deben contener un prefijo que proporcione la idea que va a devolver un booleano y siempre debe referir al estado verdadero.

   Ejemplo: esProfesor()
   tieneHijos()

6. Los nombres de las tablas deben ser en singular.

   Ejemplo: Utilice:
   estudiente

   En lugar de :
   estudiates

7. Al poner el nombre de una columna de una tabla evitar repetir el nombre de la tabla.

   Ejemplo: Evite:
   estudianteApellido

8. Evitar poner en el nombre de una columna el tipo de dato que se almacena.
9. Minimizar el uso de las abreviaturas y en caso de que se utilicen, se debe emplear coherentemente las que ya hayan sido definidas. Cada abreviatura debe tener un solo significado y viceversa.

   Ejemplo: Si utiliza "min" para abreviar mínimo,
   no se utiliza también para abreviar minuto

10. Cuando se nombre una función o procedimiento que retorne un valor, se debe hacer mencionando la entidad que será devuelta.

    Ejemplo: consultarSaldo()
    obtenerMayor()

11. Evitar el uso de los caracteres como %, $, #, @, etc.
12. Las variables deben contener nombres descriptivos del dato que se va a almacenar.

13. El nombre de las llaves foráneas debe contener el prefijo Fk, y en su nombre debe haber una referencia a la tabla que pertenece.

2.5.2.2 Estándar para la documentación interna (Comentarios)

Existen dos tipos de documentación de software: externa e interna. La documentación externa, se debe mantener fuera del código fuente, como por ejemplo las especificaciones, los archivos de ayuda, los documentos de diseño, etc. La documentación interna está formada por los comentarios que los programadores escriben dentro del código.

Uno de los problemas de la documentación interna del software, es garantizar que se mantengan y actualicen los comentarios al mismo tiempo que el código fuente. Unos buenos comentarios en el código fuente resultan valiosísimos para quien tenga que mantener un software. Se recomienda seguir la siguiente guía para realizar comentarios:

1. Los comentarios deben ser escritos en español y con mayúscula, para hacer notar la diferencia entre el código y el comentario.
2. Evite añadir comentarios al final de una línea de código, porque lo hacen más difícil de leer. Sin embargo, los comentarios de final de línea sí son apropiados al anotar declaraciones de variables, en este caso, alinee todos los comentarios de final de línea en la misma posición de tabulación.
3. Evitar los comentarios recargados o extensos.
4. Si se necesita realizar comentarios para explicar una sección de código compleja, examine el código para decidir si debería volver a escribirlo. Siempre que sea posible, no documente un código malo, vuelva a escribirlo. Aunque, por regla general, no debe sacrificarse el rendimiento para hacer un código más simple para el usuario, es indispensable un equilibrio entre rendimiento y mantenibilidad.
5. Usar frases completas cuando se escriba comentarios. Los comentarios deben aclarar el código, no añadirle ambigüedad.
6. Se debe ir comentando al mismo tiempo que se programa, porque probablemente no haya tiempo de hacerlo más tarde. Por otro lado, aunque exista la oportunidad de revisar el código que se ha escrito, lo que parece obvio hoy es posible que seis semanas después no lo sea.

7. Evitar los comentarios innecesarios o inapropiados, es decir, se debe evitar lo comentarios que expliquen cosas obvias, en la mayoría de las cosas el código debe ser autoexplicativo. Un buen código con buenas prácticas de nombrado no debe contener muchos comentarios.
8. Usar los comentarios para explicar el propósito del código, no como si fueran traducciones literales.
9. Para evitar problemas recurrentes, se debe hacer siempre comentarios al depurar errores y solucionar problemas de codificación, especialmente cuando trabaje en equipo.
10. Se debe hacer comentarios en el código que esté formado por ciclos, en este caso son áreas clave que ayudarán a los lectores del código fuente.
11. Realizar los comentarios en un estilo uniforme, respetando una puntuación y estructura coherente a lo largo de toda la aplicación.
12. Cada declaración de variable, debe incluir un comentario en la misma línea que describa el uso, especifique su objetivo y clasificación de la variable que se declara.
13. Una escritura incorrecta demuestra un desarrollo negligente, por tanto, todos los comentarios deben estar ortográficamente bien escritos.
14. Se debe disponer de un comentario que contenga la labor que va a realizar la función, qué papel juega cada uno de sus parámetros y qué valores se esperan retornar. También un comentario que muestre el inicio y fin de la misma.

```
Ejemplo: //.....INICIO DE FUNCION...../

         //...INFORMACION DE LA FUNCION.../
         //...FIN DE LA INFORMACION.../

                  FUNCION

         //.....FIN DE LA FUNCION...../
```

2.5.2.3 Estándar para el formato

El formato hace que la organización lógica del código sea más clara. Si se realiza un código fuente con un formato coherente y lógico, resulta de gran utilidad a los programadores que tengan que descifrarlo. Se recomienda seguir la siguiente guía para establecer el formato:

1. Establecer un tamaño estándar de sangría y utilizarlo siempre, se debe alinear las secciones de código mediante la sangría predeterminada.
2. Cuando se publiquen versiones impresas del código fuente, se debe utilizar un único tipo de letra.
3. Se debe declarar una sola variable por línea.
4. Uso de foreach en lugar de for. Usando foreach se aumenta la legibilidad y la elegancia del código.
5. Se recomienda hacer uso del "*alias"*, esto permite no repetir el nombre de la tabla que se le realice una consulta y para hacer cambios es muy fácil.
6. Para hacer referencia al nombre de tablas, esquemas, secuencias etc. se recomienda que se utilice "quote_ident", de esta manera se identifica que es un objeto.
7. Para hacer un resguardo, una salva etc. se recomienda el formato "*Plano*", en lugar del "*Compress*" o el "*Tar"*, debido a que estos dos últimos hacen el resguardo en un formato que no es compatible para todos los clientes y esto dificulta el mantenimiento.
8. Se recomienda el comando "*Insert*" y no el "*Copy*", este último solo es necesario para grandes volúmenes de información.
9. Incluir llaves en los bloques condicionales, así sea de una sola sentencia. Realmente funcionan igual, pero si a las sentencias for, foreach, if y else se le añaden llaves de apertura y cierre aunque incluyan una única sentencia, se lee mejor.
10. Las llaves de apertura y cierre deben estar alineadas verticalmente.
11. Utilizar espacios antes y después de los operadores.
12. Usar espacios en blanco para organizar el código en secciones. De tal manera que se comprenda la segmentación del código y en caso de ser muy extenso se puede utilizar regiones. Se recomienda que las instrucciones de las funciones deben contener una identación de tres espacios.
13. Siempre que sea posible, no coloque más de una instrucción por línea.
14. Cuando se utilice más de un atributo, se debe colocar todos separados por comas.
15. Cada cláusula (SELECT, FROM, WHERE, GROUP, ORDER) debe ir en su propia línea, o sea, no manejar más de una instrucción por línea, de esta manera, son más fáciles de leer.

Ejemplo:
```
SELECT nombre, apellido
FROM estudiante
WHERE fecha = '1'
```

16. Las cláusulas de longitud mayor de 80 caracteres, deben ser divididas en las divisiones naturales (como comas entre los identificadores) y en ocasiones en espacios[19] hacia adentro, para contener una nueva línea.
17. Las subconsultas deben comenzar en su propia línea, y ser espacios hacia adentro para que coincida con el inicio de la subconsulta, cada cláusula de subconsulta debe iniciar su propia línea.
18. Las expresiones complejas también deben obtener su propia línea, con espacios en ellas.

```
Ejempplo: SELECT table1.field1,table2.field2,
             expression1,
             expression2
          FROM table1 JOIN table2 ON condition
             JOIN table3 ON condition
          WHERE filter1
             AND filter2
             AND table1.field5 IN
                (SELECT table1.fieldA
                FROM subtableA
                WHERE filterA)
          GROUP BY field1,field2
          ORDER BY field1,field2
```

19. Palabras de la gramática de SQL y órdenes internas como (SELECT, UPDATE, WHERE, INSERT, AND, OR, FROM, etc.) deben estar en todas las etapas con mayúsculas, y los identificadores y las funciones habituales, deben estar en minúsculas.
20. En las cláusulas FROM y WHERE, tratar de poner las tablas y campos en el mismo orden en que se encuentran en la selección al igual que las cláusulas GROUP BY.
21. Nunca utilice el método abreviado de referencia GRUPO identificadores de la posición por ejemplo, (ORDER BY 2,3) aunque el estándar SQL lo permite.
22. Evite comparar las variables booleanas contra false o true, en vez de eso utilice el operador de negación.

2.5.2.4 Realizar refactorización

La utilización de herencia, encapsulamiento y polimorfismo debe combinarse con técnicas de mayor nivel. Estas técnicas de mayor nivel son los patrones de diseño para obtener software flexible, los antipatrones

[19] Los espacios en estas buenas prácticas son sinónimos de viñetas o pestañas.

para evitar cometer errores recurrentes y la refactorización para asegurar reorganizaciones ordenadas de código fuente para mejorar la mantenibilidad, o salir de algún antipatrón.

"Refactorizar un software es modificar su estructura interna con el objetivo de que sea más fácil de entender y de modificar en el futuro, tal que el comportamiento observable del software al ejecutarse no se vea afectado."

Una refactorización es una transformación controlada del código fuente de un sistema que no altera su comportamiento observable, cuyo fin es hacerlo más comprensible y de más fácil mantenimiento. Es una forma disciplinada de limpiar el código minimizando las probabilidades de introducir defectos. La refactorización es la parte del mantenimiento del código donde no se arregla errores ni se añade funcionalidad. El objetivo, por el contrario, es mejorar la facilidad de comprensión del código o cambiar su estructura y diseño, así como eliminar código muerto para facilitar el mantenimiento en el futuro.

Este proceso permite tomar diseños defectuosos con código mal escrito, por ejemplo (duplicidad y complejidad innecesaria) y adaptarlo a uno más organizado. También muestra que el diseño no se da sólo al inicio, sino también a lo largo del ciclo de desarrollo, durante la codificación, de manera tal que el diseño original no decaiga.
La figura 21 muestra cómo luego de haber modificado la estructura interna (al inicio, gris, con formas irregulares, y al final blanco, con figuras regulares) ante la misma entrada (un óvalo negro), se obtiene la misma salida (un pentágono gris). Esto ilustra la modificación de la estructura interna sin haber modificado el comportamiento observable.

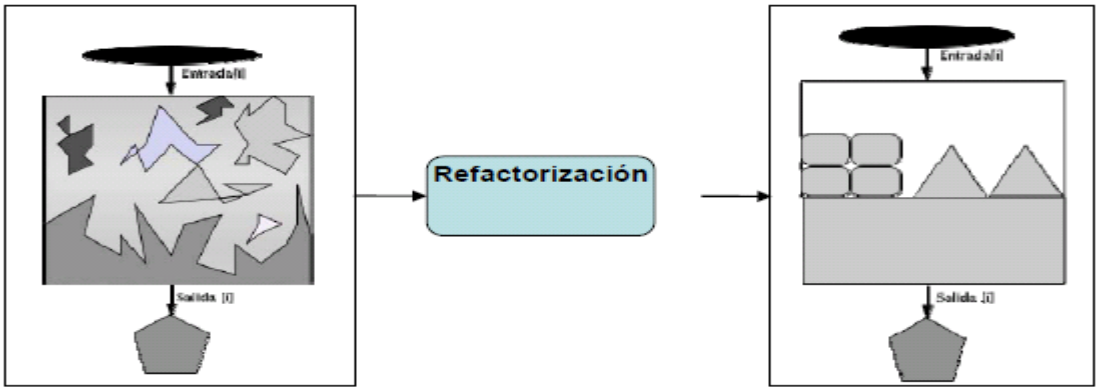


Figura21. Refactorización: antes y después.

Para poder refactorizar de forma satisfactoria es indispensable contar con un conjunto de casos de prueba que validen el correcto funcionamiento del sistema. Los casos de prueba permiten verificar repetida e incrementalmente si los cambios introducidos han alterado el comportamiento observable del programa. El primer paso del proceso de refactorización es ejecutar las pruebas antes de haber efectuado cualquier cambio, de este modo se puede conocer el comportamiento actual del sistema. Los resultados deben ser los mismos que se obtengan luego de la refactorización. El segundo paso consiste en analizar los cambios a realizar, y el tercero es, finalmente, la aplicación del cambio. Luego se vuelven a ejecutar las pruebas para comprobar los resultados antes y luego de efectuada la refactorización, los cuales deben ser iguales. Al ser iguales los resultados de las pruebas se verifica que no se ha modificado el comportamiento observable del sistema y por consiguiente, no se han introducido fallas. Se debe aclarar que una optimización de código no es una refactorización, ambas tienen en común la modificación del código fuente sin alterar el comportamiento observable del software, la diferencia radica en la forma de impactar el código fuente, ya que la optimización suele agregarle complejidad. [20]

Aspectos favorables

El proceso de refactorización presenta algunas ventajas, entre las que se encuentran el mantenimiento del diseño del sistema, incremento de facilidad de lectura y comprensión del código fuente, detección temprana de fallos y aumento en la velocidad en la que se programa. Una forma importante de mejorar el código es eliminando código duplicado, lo que tiene un impacto directo en las modificaciones futuras del código fuente: mientras más código hay, más complicado es modificarlo correctamente ya que hay más código para analizar. Tanto (Fowler et al, 1999) como (Piattini y García, 2003), coinciden en lo siguiente:

- La refactorización facilita la comprensión del código fuente, principalmente para los desarrolladores que no estuvieron involucrados desde el comienzo del desarrollo. El hecho de que el código fuente sea complejo de leer reduce mucho la productividad ya que se necesita demasiado tiempo para analizarlo y comprenderlo. Invirtiendo algo de tiempo en refactorizar de manera tal que exprese de forma más clara cuáles son sus funciones, en otras palabras, que sea lo más auto-documentable posible, facilita su comprensión y mejora la productividad.
- La refactorización permite detectar errores. Cuando el código fuente es más fácil de comprender permite detectar condiciones propensas a fallos, o analizar supuestos desde los que se partió al inicio del desarrollo, que pueden no ser correctos. Mejora la robustez del código escrito.

- La refactorización permite programar más rápido, lo que eleva la productividad de los desarrolladores. Un punto importante a tener en cuanta al desarrollar el código es la rapidez con que este se pueda efectuar, de hecho, un factor clave para permitir el desarrollo rápido del mismo es contar con un buen diseño.
- La velocidad en la programación se obtiene al reducir los tiempos que lleva la aplicación de cambios: si el código fuente no es fácilmente comprensible, entonces los cambios llevaran más tiempo. Refactorizar mejora el diseño, la lectura y comprensión del código fuente y reduce la cantidad de posibles fallas, lo que lleva a mejorar la calidad del software entregado, como así también aumenta la velocidad de desarrollo.

Aspectos desfavorables

Tanto Fowler como Piattini y García, coinciden en que las áreas conflictivas de la refactorización son las bases de datos, y los cambios de interfaces.

El cambio de base de datos tiene dos problemas: los sistemas están fuertemente acoplados a los esquemas de las bases de datos y el segundo de ellos radica en la migración tanto estructural como de datos. Se deben aplicar los cambios necesarios y luego migrar los datos existentes en la base de datos, lo que es muy costoso.

Ejemplo:

Descomponer un condicional

En la figura 22 se muestra el cambio de un condicional por una llamada a una función, es decir, la refactorización se muestra en tres llamadas a un procedimiento.

Observe el siguiente ejemplo:

```
if (date.before(SUMMER_START) || date.after(SUMMER_END))
    charge = quantity * _winterRate + _winterService Charge;
else
    charge = quantity * _summerRate;
```

y su refactorización:

```
if (summer(date))
    charge = summerCharge(quantity);
else
    charge = winterCharge(quanity);
```

Figura22. Descomponer un condicional.

2.5.3 Preguntas de control

Las respuestas a esta pregunta se pueden encontrar en el Anexo 4.

1. Responda verdadero o Falso (V o F).

____ Utilizando estructuras diferentes para programar se logra que la mantenibilidad del código se eleve rápidamente.

____ Un orden determinado en el código facilita el mantenimiento del mismo.

____ Con un estándar de codificación se puede asegurar un código con calidad pero no se eleva la mantenibilidad del mismo.

____ El mejor método para asegurarse de que un equipo de desarrolladores mantenga un código con calidad elevando la mantenibilidad del mismo es estableciendo un estándar de codificación.

____ Un estándar de codificación tiene que presentar técnicas para el nombrado, la documentación externa y el formato.

2.5.4 Lista de chequeo

En la siguiente tabla se puede observar la lista de chequeo que se propone en la investigación.

Nivel	Criterio de evaluación	Evaluación	N.P.	Observaciones
	Codificación			
	➔ Debe utilizar o regirse por			

	estándares de codificación.			
	1 ¿Utiliza estándares de codificación?			
	2 ¿Se identifican diferentes secciones en los estándares de codificación por los que se rige?			
	➔ Debe utilizar una sección para el nombrado en el estándar de codificación.			
	1 ¿Tiene en cuenta técnicas para el nombrado en el estándar de codificación utilizado?			
	2 ¿Los nombres de sus procedimientos expresan el objetivo de los mismos?			
	3 ¿utiliza en las declaraciones de sus variables y procedimientos técnicas de la notación CamelCasing?			
	4 ¿Evita el uso de caracteres extraños tales como %, #, $, @ al realizar nombrados?			
	5 ¿En el nombre de las llaves foráneas utiliza el prefijo FK y hace referencia a la tabla que pertenece?			
	➔ Debe utilizar una sección para los comentarios en el estándar de codificación.			
	1 ¿Tiene en cuenta técnicas para los			

		comentarios en el estándar de codificación utilizado?			
	2	¿Actualiza los comentarios de su código al mismo tiempo que lo mantiene?			
	3	¿Hace notar la diferencia entre el código y los comentarios?			
	4	¿Evita comentarios extensos y confusos?			
	5	¿Realiza comentarios al depurar errores?			
	6	¿Tiene en cuenta reglas ortográficas para realizar los comentarios?			
	➔	Debe utilizar una sección para el formato en el estándar de codificación.			
	1	¿Tiene en cuenta técnicas para el formato en el estándar de codificación utilizado?			
	2	¿Declara una sola variable por línea?			
	3	¿Utiliza el "*alias"*?			
	4	¿Utiliza "*quote_ident*" para hacer referencia al nombre de alguna tabla?			
	5	¿Utiliza el formato "*Plano*" para hacer una copia de seguridad?			
	6	¿Tiene incluida llaves en los			

	bloques condicionales?			
	7 ¿Las llaves de apertura y cierre están alineadas verticalmente?			
	8 ¿Utiliza espacios antes y después de los operadores?			
	9 ¿Las palabras de la gramática SQL y las órdenes internas las tiene escrita con mayúscula?			
	10 ¿Utiliza para cada cláusula una línea?			

Tabla #5.Lista de Chequeo para la Codificación.

Donde:

- En el **Nivel** solo se define cuando existen aspecto de mayor importancia que otros, para ello se usarán signos de exclamación (!), entre más tenga más importante es.
- En el **Criterio de evaluación** se ubica el aspecto a evaluar, siempre con una redacción nítida.
- **Evaluación** es para el caso de la aplicación de la Lista de Chequeo. Dicha evaluación se encontrará en el rango de 0-5.
- **N.P.** significa No Procede y es para el caso de la aplicación que ese aspecto no sea factible su valoración.
- **Observaciones** son para cualquier elemento que quiera incluir la persona que aplica dicha lista.

2.6 Conclusiones parciales

Partiendo de la necesidad de obtener mantenibilidad en las bases de datos realizadas en PostgreSQL como fundamento para la realización de la guía de buenas prácticas se analizaron del mismo gestor aspectos tales como el Diseño, Configuración, Arquitectura y Codificación, para cada uno de estos aspectos analizados se propuso un conjunto de buenas prácticas, así como una lista de chequeo y preguntas de control con el objetivo de que los desarrolladores corroboren los conocimientos obtenidos. La guía propuesta está basada en experiencias de desarrolladores de bases de datos y en varias bibliografías consultadas.

3.1 Introducción

El objetivo de la investigación es realizar una guía de buenas prácticas para obtener mantenibilidad en las bases de datos desarrolladas en PostgreSQL; para dar cumplimiento a dicho objetivo se hace necesario someter la propuesta a la valoración de un conjunto de especialistas para que brinden sus criterios y sugerencias. Con este fin se utilizan técnicas propuestas por el método Delphi basado en el criterio de especialistas.

3.2 Método Delphi

El método Delphi, cuyo nombre se inspira en el antiguo oráculo de Delphos, fue ideado originalmente a comienzos de los años 50 en el seno del Centro de Investigación estadounidense RAND Corporation por Olaf Helmer y Theodore J. Gordon, como un instrumento para realizar predicciones sobre un caso de catástrofe nuclear. Desde entonces, ha sido utilizado frecuentemente como sistema para obtener información sobre el futuro.

Linston y Turoff definen la técnica Delphi como: "*Un método de estructuración de un proceso de comunicación grupal que es efectivo a la hora de permitir a un grupo de individuos, como un todo, tratar un problema complejo"*.

Este método consiste en la selección de un grupo de especialistas a los que se les encuesta su opinión sobre cuestiones referidas a acontecimientos del futuro. La capacidad de predicción del método se basa en la utilización sistemática del juicio intuitivo emitido por el grupo seleccionado. Es decir, el método Delphi procede por medio de la interrogación a especialistas con la ayuda de cuestionarios sucesivos, a fin de poner de manifiesto convergencias de opiniones y deducir eventuales consensos. La encuesta es realizada de forma anónima por lo que la calidad de los resultados depende del cuidado que se ponga en la confección del cuestionario y en la elección de los especialistas a consultar. [21]

Dentro de las características que presenta se puede encontrar: [22]

- *Anonimato.* Durante el Delphi ningún experto conoce la identidad de los otros que componen el grupo de debate.

- *Iteración y realimentación controlada.* La iteración se consigue al presentar varias veces el mismo cuestionario, lo que permite disminuir el espacio intercuartil, ya que se consigue que los especialistas vayan conociendo los diferentes puntos y puedan ir modificando su opinión.
- *Respuesta del grupo en forma estadística:* La información que se presenta no es solo el punto de vista de la mayoría sino que se presentan todas las opiniones indicando el grado de acuerdo que se ha obtenido.
- *Heterogeneidad:* Pueden participar especialistas de determinadas ramas de actividad sobre las mismas bases.

3.3 Validación por el método Delphi

En la investigación se utiliza el método Delphi para la validación de la propuesta, pues mediante éste los especialistas deben predecir los resultados a alcanzar con la propuesta elaborada, lo que es muy exacto para obtener información sobre el futuro. El procesamiento estadístico de la información es la característica más importante del método que lo diferencia del resto de los métodos pues la decisión final que se toma es un criterio fuertemente avalado por la experiencia y conocimiento del colectivo de especialistas consultado.

A continuación se detallan los pasos necesarios para la aplicación del método al problema en cuestión, pero antes se debe mencionar los 4 aspectos que éste presenta:

1. Selección de los especialistas.
2. Elaboración del cuestionario para la validación de la propuesta.
3. Cálculo de concordancia entre los especialistas.
4. Desarrollo práctico y explotación de los resultados.

3.3.1 Selección de los especialistas

En la investigación se considera que un experto se refiere a: "*Especialistas que son capaces de brindar criterios terminantes sobre el proceso de obtener mantenibilidad en las bases de datos desarrolladas en PostgreSQL"*.

Teniendo en cuenta lo anteriormente planteado, se realiza la selección de especialistas bajo las siguientes condiciones:

- Debe ser graduado de nivel superior.
- Debe estar vinculado a la investigación y al desarrollo de bases de datos desarrolladas en PostgreSQL.
- Debe contar con un año de experiencia como mínimo en el tema.
- Debe poseer conocimiento en el proceso de creación de bases de datos desarrolladas en PostgreSQL.
- Debe contar con conocimiento y habilidades en actividades de calidad del software.

La cantidad de especialistas seleccionada es once y a estos se les aplica la encuesta de autovaloración que se observa en el Anexo 5 para determinar el coeficiente de competencia.

Una metodología completa y sencilla para la determinación de la competencia de los especialistas la constituye la aprobada en febrero de 1971 por el comité estatal para Ciencia y la técnica de Rusia para elaboración de pronósticos científicos técnico. En esta metodología la competencia de los especialistas se termina por el coeficiente (**k**), el cual se calcula de acuerdo con la opinión del experto sobre su nivel de conocimiento acerca del problema que se está resolviendo y con las fuentes que le permiten argumentar sus criterios.

El **coeficiente de competencia** se calcula por la siguiente fórmula:

$$K = ½ (kc + ka)$$

Donde:

kc- es el coeficiente de conocimiento o información que tiene el experto acerca del problema, calculado sobre la valoración del propio experto en una escala del 0 al 10 y multiplicado por 0,1; de esta forma, la evaluación "0" indica que el experto no tiene absolutamente ningún conocimiento de la problemática correspondiente, mientras que la evaluación "10" significa que el experto tiene pleno conocimiento de la problemática tratada. El experto debe marcar con una cruz (X) en la casilla que estime pertinente. La siguiente tabla muestra el nivel de conocimiento de los especialistas seleccionados.

No. Experto	0	1	2	3	4	5	6	7	8	9	10
1								X			
2							X				

3							X				
4				X							
5								X			
6									X		
7								X			
8								X			
9									X		
10									X		
11										X	

Tabla #6.Coeficiente de conocimiento de los especialistas seleccionados.

ka- es el coeficiente de argumentación o fundamentación de los criterios del experto.

Para calcular este coeficiente el experto debe marcar según su consideración, cuáles fueron sus fuentes para la obtención del conocimiento que le permite argumentar su evaluación del nivel de conocimiento que especifica anteriormente, en el Anexo 5 se puede observar dicha tabla. En la siguiente tabla se muestra los valores obtenidos por cada uno de los especialistas.

FUENTES DE ARGUMENTACIÓN	**Grado de influencia de cada una de las fuentes en sus criterios.**		
	A (alto)	**M (medio)**	**B(bajo)**
Análisis teóricos realizados por usted.	1,2,3,4,6,7,10,11	8,9	5
Su experiencia obtenida.	1,6,9,10,11	2,3,4,5,7,8	
Trabajos de autores nacionales.		2,3,5,9,11	1,4,6,7,8,10
Trabajos de autores extranjeros.	1,2,3,5,6,7,8,10	4,9,11	
Su propio conocimiento del estado del problema en el extranjero.		1,6,10,11	2,3,4,5,7,8,9
Su intuición.		2,3,9	1,4,5,6,7,8,10,11

Tabla #7. Tabla para calcular el coeficiente de argumentación.

Para calcular el coeficiente de argumentación las respuestas de los especialistas se traducen a puntos según lo que muestra la siguiente tabla patrón, este se calcula sumando los valores de la tabla patrón en concordancia con las respuestas dadas por los especialistas.

FUENTES DE ARGUMENTACIÓN	Grado de influencia de cada una de las fuentes en sus criterios.		
	A (alto)	M (medio)	B (bajo)
Análisis teóricos realizados por usted.	0.3	0.2	0.1
Su experiencia obtenida.	0.5	0.4	0.2
Trabajos de autores nacionales.	0.05	0.05	0.05
Trabajos de autores extranjeros.	0.05	0.05	0.05
Su propio conocimiento del estado del problema en el extranjero.	0.05	0.05	0.05
Su intuición.	0.05	0.05	0.05

Tabla #8. Tabla patrón para determinar el coeficiente de argumentación.

El código de interpretación de tales coeficientes de competencias es:

- Si 0,8 < k < 1,0 el coeficiente de competencia es Alto.
- Si 0,5 < k < 0,8 el coeficiente de competencia es Medio.
- Si k < 0,5 el coeficiente de competencia es Bajo.

En la siguiente tabla se muestran los resultados obtenidos en la encuesta de autovaloración realizada a los especialistas seleccionados.

No. Experto	Kc Coeficiente de Conocimiento	Ka Coeficiente de Argumentación	K Coeficiente de Competencia	Grado
1	0.7	1	0.85	Alto
2	0.6	0.9	0.75	Medio
3	0.6	0.9	0.75	Medio
4	0.3	0.9	0.6	Medio

5	0.7	0.7	0.7	Medio
6	0.8	1	0.9	Alto
7	0.7	0.9	0.8	Alto
8	0.7	0.8	0.75	Medio
9	0.8	0.9	0.85	Alto
10	0.8	1	0.9	Alto
11	0.9	1	0.95	Alto

Tabla #9. Resultados obtenidos en la Encuesta de Autovaloración.

Teniendo en cuenta que los once especialistas están de acuerdo en responder las preguntas del cuestionario y que todos tienen un coeficiente entre medio y alto, se decide que el número de especialistas del Comité de Especialistas en la investigación es once.

3.3.2 Elaboración del cuestionario para la evaluación de la propuesta

Para la evaluación de la guía de buenas prácticas para obtener mantenibilidad en las bases de datos desarrolladas en PostgreSQL se utiliza la encuesta que se observa el en Anexo 6, entre sus objetivos se puede mencionar:

1. Establecer la importancia de la propuesta para dar solución a la problemática planteada en la investigación.
2. Establecer el nivel de recomendación de la guía.
3. Determinar la complejidad de la guía.
4. Determinar el alcance de la guía.
5. Determinar la necesidad de la guía.
6. Determinar la adaptabilidad de la guía a otros proyectos productivos que hacen uso de PostgreSQL.
7. Determinar la efectividad de las actividades del proceso propuesto.
8. Determinar el cumplimiento del objetivo de la investigación *(elevar la mantenibilidad de las bases de datos desarrolladas en PostgreSQL*).
9. Identificar aspectos erróneos y recomendaciones que permitan mejorar la propuesta. Para ello, en cada pregunta formulada los especialistas podían realizar sus observaciones.

En la siguiente tabla se puede observar el objetivo que se satisface en cada una de las preguntas en específico.

OBJETIVOS	PREGUNTAS					
	1	2	3	4	5	6
Importancia de la Guía.	X					
Recomendación de la Guía.		X				
Complejidad de la Guía.			X			
Alcance de la Guía.				X		
Necesidad de la Guía.					a)	
Adaptabilidad de la Guía a otros proyectos productivos.					b)	
Efectividad de la Guía.					c)	
Cumplimiento del Objetivo de la Guía.						X

Tabla #10. Objetivos por preguntas específicas.

3.3.3 Cálculo de concordancia entre los especialistas

Definido ya el grupo de especialistas y enviado a estos la encuesta, se buscaron sus criterios sobre la guía de buenas prácticas para obtener mantenibilidad en las bases de datos desarrolladas en PostgreSQL. Antes de realizar la explotación de los resultados es necesario calcular la concordancia entre los especialistas para medir en qué magnitud estos están de acuerdo en sus respuestas. Este coeficiente fue posible calcularlo utilizando el software "*SPSS 13.0 for Windows*". El resultado obtenido fue 0.84, siendo este un valor muy cercano a 1.00, lo que evidencia que los especialistas tienen un alto grado de concordancia en todas las respuestas. En la siguiente figura se puede observar el coeficiente de Kendall obtenido.

Test Statistics

N			11
Kendall's W(a)			.843
Chi-Square			64.895
df			7
Asymp. Sig.			.000
Monte Carlo Sig.	**Sig.**		.000(b)
	95% Confidence Interval	**Lower Bound**	.000
		Upper Bound	.000
a Kendall's Coefficient of Concordance			
b Based on 10000 sampled tables with starting seed 2000000.			

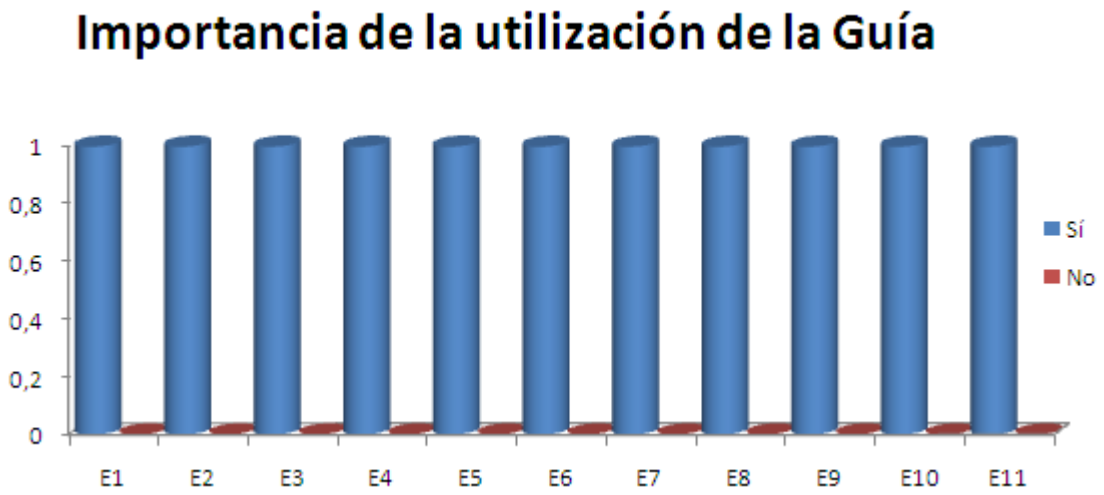
Figura23. Coeficiente de Kendall obtenido utilizando "SPSS 13.0 for Windows".

3.3.4 Desarrollo práctico y explotación de los resultados

Para ir almacenando los resultados aportados por los especialistas se confeccionan tablas utilizando el programa "*Excel 2007*". A continuación se muestra los gráficos y porcentajes obtenidos por cada uno de los objetivos establecidos.

3.3.4.1 Establecer la importancia de la propuesta para dar solución a la problemática planteada en la investigación

En la pregunta uno de la encuesta se le da cumplimiento a este objetivo, donde los especialistas responden negativa o positivamente en cuanto a su consideración si es de gran importancia la creación y utilización de la guía de buenas prácticas para ayudar a los desarrolladores de bases de datos a elevar la mantenibilidad en las mismas. El siguiente gráfico muestra el resultado obtenido:

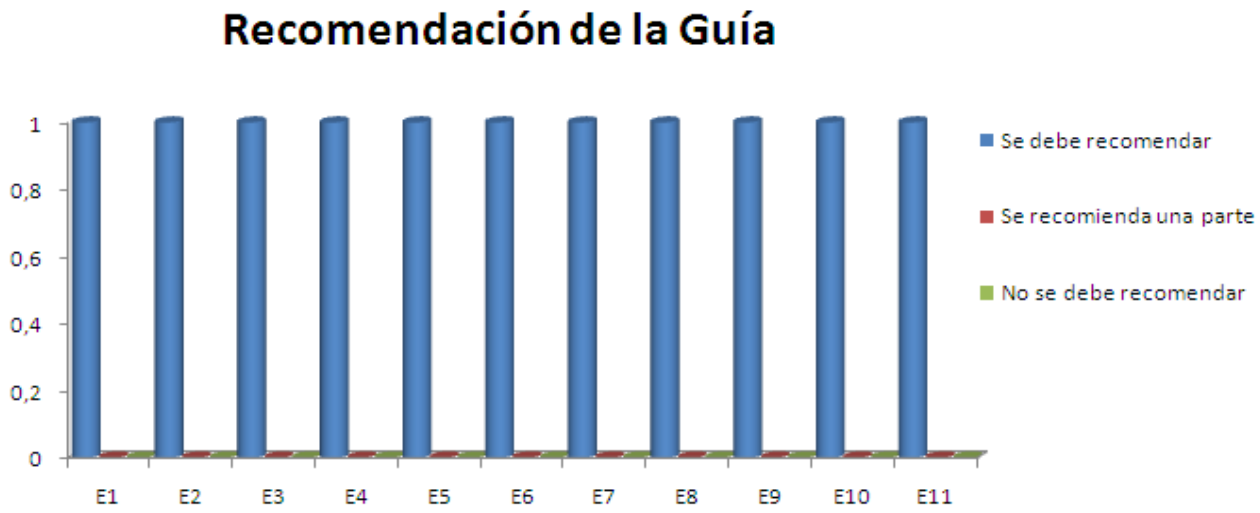


Como se puede observar en el gráfico, todos los especialistas coinciden en la importancia de la creación y utilización da la guía de buenas prácticas para ayudar a los desarrolladores de bases de datos elevar la mantenibilidad de las mismas, por lo que se obtiene un 100% de aceptación, opinando lo siguiente:

- La misma proporciona elementos al desarrollador ayudándolo a obtener conocimientos para obtener mantenibilidad en las bases de datos.
- La explotación de grandes sistemas de almacenamiento contiene implícito el problema de la mantenibilidad como elemento necesario, en ocasiones imprescindibles para la estabilidad y prestaciones de las mismas.
- Ayuda a los especialistas con poca experiencia en el tema.
- Específicamente en el GBD PostgreSQL sería muy útil pues contribuye a utilizar dicho gestor, ya que éste es muy potente y no se le otorga la importancia que requiere.
- Se hace necesario normalizar y estandarizar las prácticas para lograr un mejor funcionamiento de los gestores de bases de datos y en especial PostgreSQL.
- Es importante porque así se haría más fácil mantener las bases de datos de los proyectos, éstas se parecerían más entre ellas y no sería tan complicado comprenderlas y efectuar cambios o tareas de mantenimiento.
- Para garantizar un mejor funcionamiento del SGBD.

3.3.4.2 Establecer el nivel de recomendación de la guía

A este objetivo se le da cumplimiento en la pregunta dos de la encuesta, donde los especialistas tienen que responder: “se debe recomendar”, “se debe recomendar una parte” o “no se debe recomendar” a los desarrolladores de bases de datos el uso de la guía de buenas prácticas para obtener mantenibilidad en las mismas. Los resultados alcanzados se muestran en el siguiente gráfico:

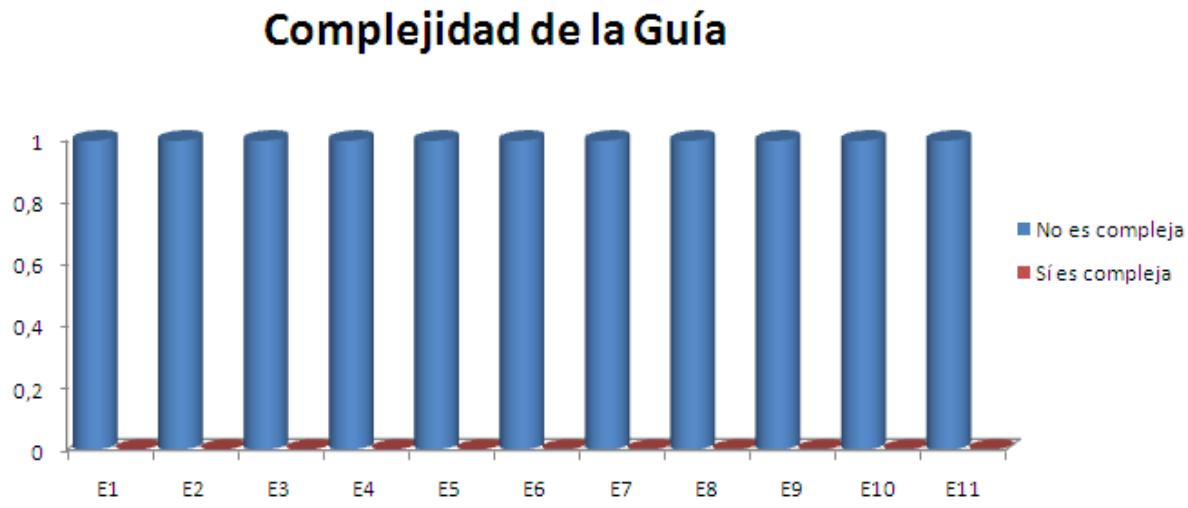


Todos los especialistas concuerdan en que se debe recomendar la guía de buenas prácticas para obtener mantenibilidad en las bases de datos, lo que constituye un 100% de aceptación y opinan lo siguiente:

- Se debe recomendar por el hecho de que estos aspectos hay que tenerlos en cuenta desde el mismo diseño y concepción del sistema.
- En esta guía se plantean prácticas muy eficientes para el trabajo con las bases de datos.
- Muchos proyectos no llevan a cabo un plan de mantenimiento de bases de datos ni se realizan las adecuadas normas de administración.
- Garantiza que el servicio esté bien configurado.
- Se debe recomendar porque los desarrolladores son los máximos responsables del diseño de la base de dato, así como de garantizar el óptimo funcionamiento de la misma.
- Se debe recomendar para que los desarrolladores realicen mantenimiento de una manera uniforme y se eviten errores posteriores.

3.3.4.3 Determinar la complejidad de la guía

La pregunta tres responde a este objetivo, donde los especialistas responden positiva o negativamente según sus conocimientos si existe alguna dificultad o complejidad en el proceso de aplicación de la guía por los desarrolladores de bases de datos, los resultados alcanzados se muestran en el siguiente gráfico:

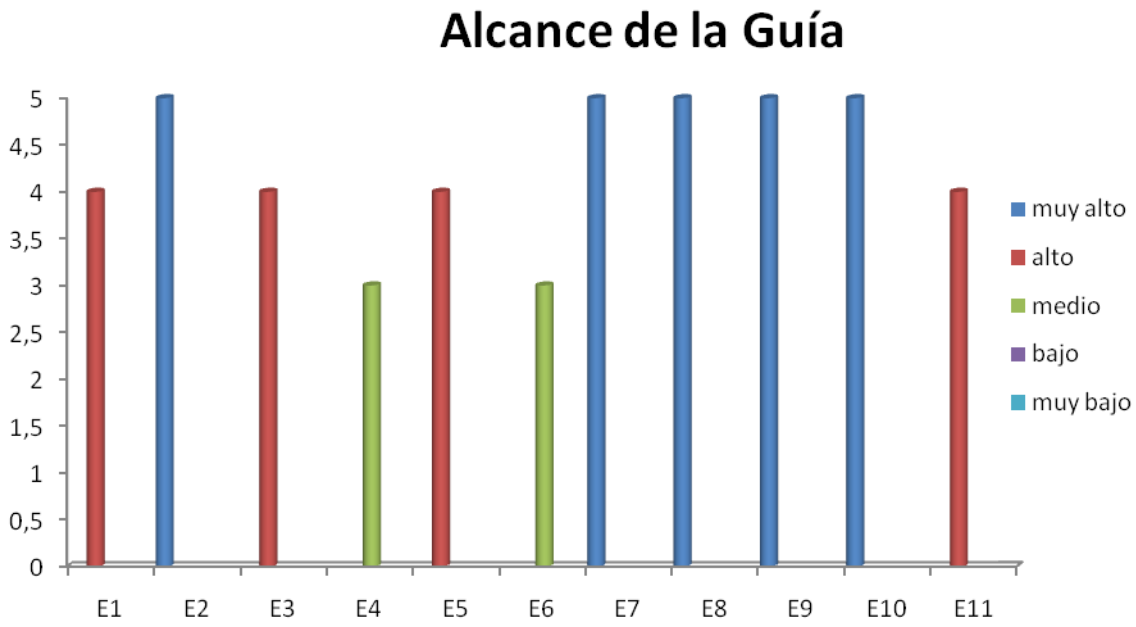


En esta pregunta todos los especialistas concuerdan que la guía de buenas prácticas para obtener mantenibilidad en las bases de datos no es compleja ni presenta dificultad para su aplicación, lo que constituye un 100% y opinando lo siguiente:

- Es una guía sencilla y entendible por cualquier usuario.
- El primer paso debe constituirlo la capacitación y adiestramiento de los administradores de bases de datos en el uso y comprensión de dichos conceptos.
- Está fácil de realizar.
- Es una guía fácil de seguir.

3.3.4.4 Determinar el alcance de la guía

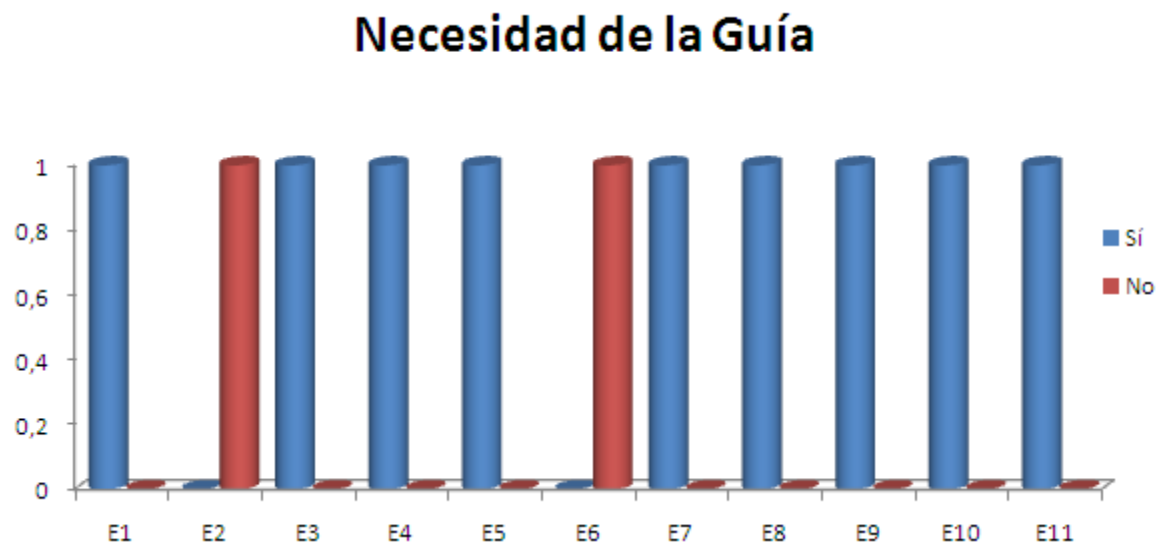
La pregunta cuatro de la encuesta brinda respuesta a este objetivo, aquí los especialistas tienen que seleccionar según sus conocimientos si la guía de buenas prácticas es abarcadora, sus respuestas pueden ser: “muy alto”, “alto”, “medio”, “bajo” o “muy bajo”. Los resultados obtenidos se muestran en el siguiente gráfico:



Como se puede observar cinco especialistas (E2, E7, E8, E9, E10) coinciden en que el alcance de la guía de buenas prácticas para obtener mantenibilidad en las bases de datos es muy alto, cuatro (E1, E3, E5, E11) coinciden en que es alto el alcance y dos (E4, E6) en que es medio, lo que constituye un 81.81% de aceptación de que la guía presenta un alcance alto.

3.3.4.5 Determinar la necesidad de la guía

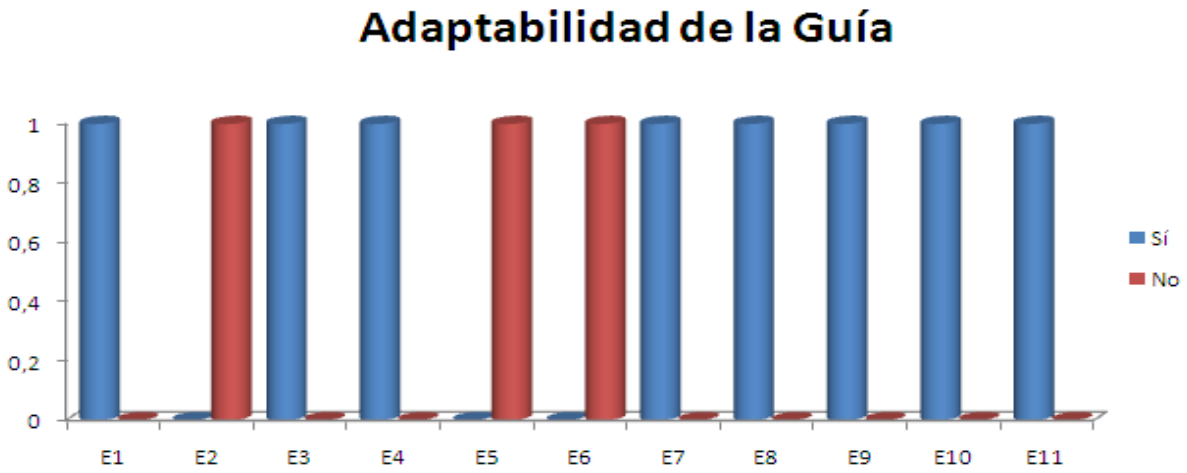
El inciso a de la pregunta cinco responde a este objetivo, aquí los especialistas deben seleccionar si existe la necesidad de emplear la guía de buenas prácticas para obtener mantenibilidad en las bases de datos, en el gráfico que se presenta a continuación se muestran los resultados logrados:



Como se puede observar solo dos especialistas (E2, E6) coinciden en que no existe una necesidad de empleo de la propuesta, por lo que la necesidad de utilización de la guía de buenas prácticas para obtener mantenibilidad en las bases de datos representa un 81.81% de coincidencia.

3.3.4.6 Determinar la adaptabilidad de la guía

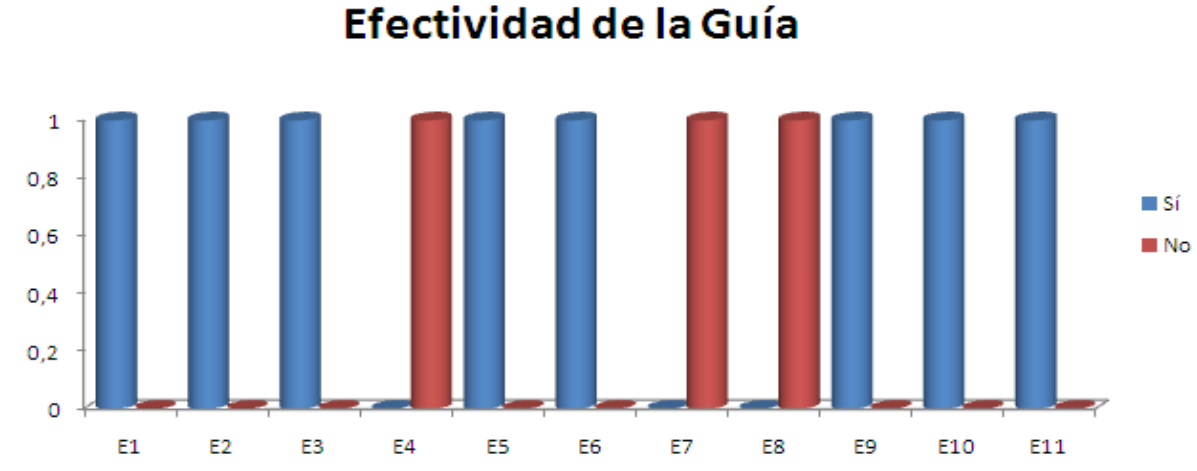
El inciso b de la pregunta cinco comprueba la adaptabilidad de la guía a los proyectos productivos, en esta pregunta los especialistas deben seleccionar si existe una adaptabilidad de la guía a proyectos productivos, los resultados alcanzados se muestran en el siguiente gráfico:



Como se puede apreciar, tres especialistas (E2, E5, E6) coinciden en que la guía de buenas prácticas no presenta adaptabilidad a proyectos productivos, es decir, ocho de los especialistas si concuerdan en que sí existe una adaptabilidad de la guía a proyectos productivos, lo que representa un 72.72% .

3.3.4.7 Determinar la efectividad de las actividades del proceso propuesto

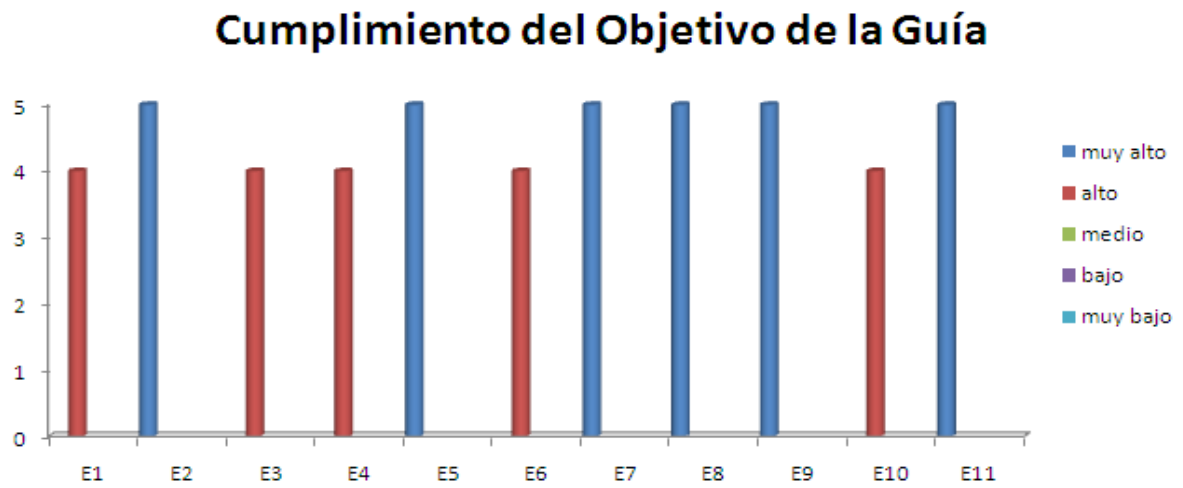
En el inciso c de la pregunta cinco se comprueba la efectividad de la utilización de la guía de buenas prácticas para obtener mantenibilidad en las bases de datos, los especialistas deben seleccionar su respuesta lo que permitirá afirmar o negar el cumplimiento del objetivo, en el siguiente gráfico se puede observar los resultados logrados:



Se puede observar que tres especialistas (E4, E7, E8) coinciden en que la guía de buenas prácticas no presenta una efectividad en su utilización, lo que representa que un 72.72% sí concuerdan en la efectividad de la misma.

3.3.4.8 Determinar el cumplimiento del objetivo de la investigación

La pregunta seis da cumplimiento a este objetivo, aquí los especialistas deben seleccionar si la guía de buenas prácticas propuesta ayuda a elevar la mantenibilidad de las bases de datos, sus respuestas pueden ser: “muy alto”, “alto”, “medio”, “bajo” o “muy bajo”. Los resultados alcanzados se pueden observar en el siguiente gráfico:



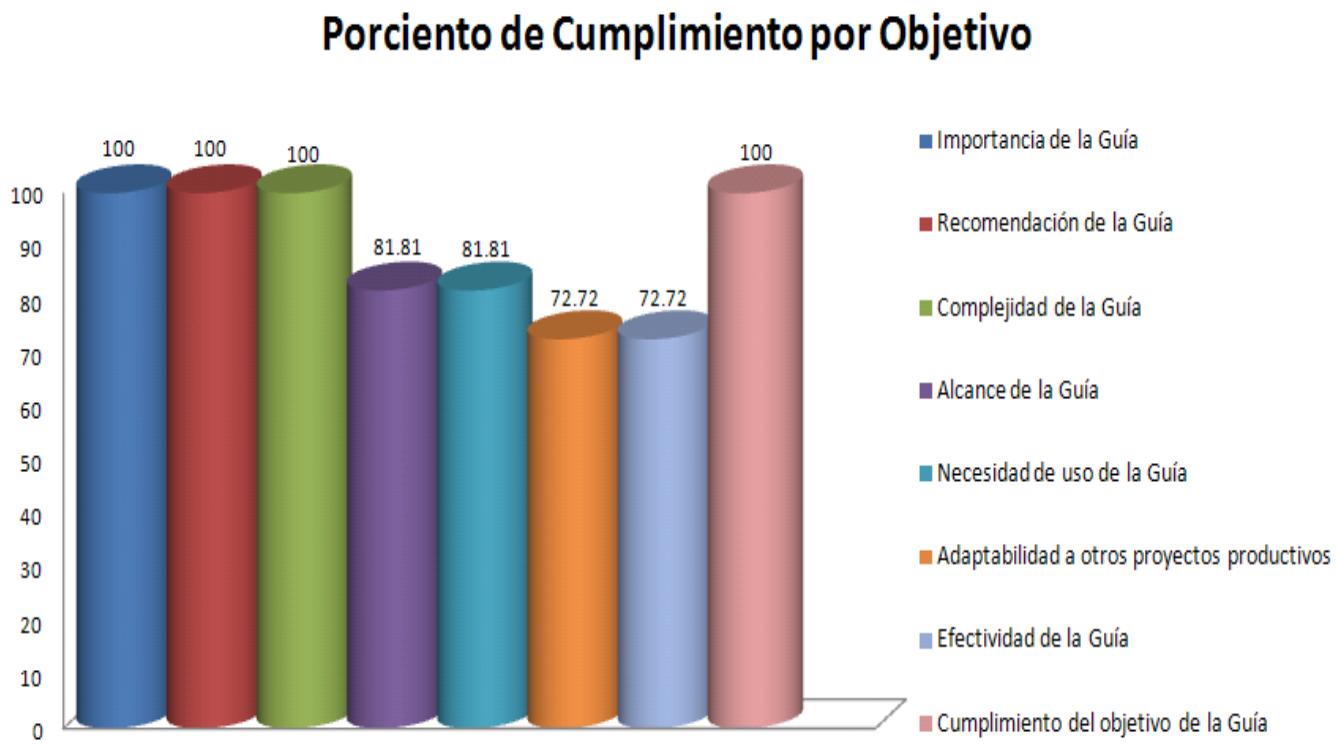
Como se puede observar todos los especialistas coinciden en que el cumplimiento del objetivo de la guía es muy alto o alto, lo que representa un 100% de concordancia.

3.3.4.9 Resultado final

El porcentaje de respuestas de los especialistas a cada uno de los objetivos propuestos fue positivo, los resultados generales se pueden observar en el próximo gráfico, pero antes es necesario tener en cuenta los siguientes resultados:

- El objetivo ” alcance de la guía” es uno de los que presenta un bajo porcentaje debido a que dos de los especialistas percibieron un alcance medio de la misma, sin embargo, en la pregunta referida al cumplimiento de los objetivos todos plantean que estos están cumplidos , por lo que el porcentaje faltante no es significativo en la validación.
- En el objetivo “necesidad de empleo de la guía” dos de los especialistas plantean que no existe la necesidad de empleo, sin embargo, en el objetivo de la Importancia y la Recomendación de la misma, todos plantean que se debe recomendar y que presenta una gran importancia, por lo que el porcentaje faltante no es significativo en la validación.
- La “adaptabilidad de la guía”es un objetivo que tampoco contradice la validación, puesto que con el objetivo tres se señala que la guía no presenta dificultad o complejidad para aplicarse, por tanto esta guía se puede aplicar a las bases de datos que se desarrollan en PostgreSQL.
- El motivo de que el objetivo “efectividad de la guía”presente un bajo porcentaje radica en que tres de los especialistas señalan que la guía no presenta efectividad, sin embargo, en el objetivo de

cumplimiento de los objetivos todos plantean que estos estaban cumplidos, por lo que el porcentaje faltante no es significativo en la validación.



3.4 Conclusiones parciales

Después de haber analizado los resultados del método Delphi, se puede afirmar que la guía propuesta para obtener mantenibilidad en las bases de datos desarrolladas en PostgreSQL es útil, correcta y efectiva. el proceso propuesto es calificado por los especialistas como un procedimiento muy adecuado que tiene en cuenta aspectos estudiados en importantes fuentes bibliográficas y que proporciona un cierto orden a todo lo que se había visto en las mismas.

Después de desarrollada toda la investigación descrita en la tesis, se ha llegado a las siguientes conclusiones:

1. Se realizó un estudio sobre la mantenibilidad de software demostrando que la misma debe ser un parámetro a tener en cuenta durante todo el desarrollo de la base de datos, pues influye en su posterior puesta en práctica.
2. Se obtuvo una guía que permite lograr una alta mantenibilidad en las bases de datos desarrolladas en PostgreSQL, basada en estudios internacionales y en las experiencias de administradores de bases de datos.
3. Se realizó la evaluación de la guía utilizando el método Delphi, en el que se consultaron once especialistas que la valoraron de útil, correcta y efectiva para lograr una alta mantenibilidad en las bases de datos desarrolladas en PostgreSQL.

Se recomienda lo siguiente:

- Calcular, antes y después de realizar el mantenimiento, el Índice de Mantenibilidad, el cual serviría como una forma de evaluar los cambios realizados.

- Utilizar herramientas que favorezcan más el trabajo con la mantenibilidad, com son “pg_top” y “pgFouine” las cuales optimizarán el trabajo a realizar.

- Aplicar la guía propuesta a todos los proyectos productivos que hacen uso de bases de datos desarrolladas en PostgreSQL.

- Profundizar más en el estudio de nuevas prácticas a aplicar en cada uno de los aspectos tratados en la guía.

[1]. **Macario Polo , Francisco Ruiz.** Mantenimiento del Software. *Mantenimiento del Software.* [Online] 2000/2001. [Cited: noviembre 10, 2009.] http://alarcos.inf-cr.uclm.es/per/fruiz/cur/mso/trans/s1.pdf.

[2]. **Miguel Angel Sicilia.** Connexions. *Connexions.* [Online] noviembre 27, 2008. [Cited: noviembre 10, 2009.] http://cnx.org/content/m17452/latest/.

[3]. **Verónica De la Morena.** Conexxions. *Conexxions.* [Online] septiembre 4, 2008. [Cited: noviembre 17, 2009.] http://cnx.org/content/m17399/latest/.

[4]. **Juan Cesar Martínez, Marina E Ramírez Andonegui.** Norma ISO 14764 Mantenimiento de Software. *Norma ISO 14764 Mantenimiento de Software.* [Online] [Cited: noviembre 17, 2009.] http://www.ci.ulsa.mx/~elinos/docencia/ctrldesa/ISO14764.pdf.

[5]. **Universidad Cooperativa de Colombia Sede Villamaría.** *Análisis y Diseño.*

[6]. **Miguel Angel Sicilia.** Connexions. *Connexions.* [Online] enero 7, 2009. [Cited: diciembre 2, 2009.] http://cnx.org/content/m17461/latest/.

[7]. **Carlos D. González.** usabilidadweb.com.ar. *usabilidadweb.com.ar.* [Online] octubre 13, 2009. [Cited: diciembre 2, 2009.] http://www.usabilidadweb.com.ar/metodos_eval_calidad_web.php.

[8]. **Juan Antonio López Quesada.** *Mantenimiento del Softeare.*

[9]. **Damián Pérez Valdés .** maestros del web. *maestros del web.* [Online] [Cited: enero 12, 2010.] http://www.maestrosdelweb.com/principiantes/%C2%BFque-son-las-bases-de-datos/.

[10]. **Beatriz Pereyra.** *Bases de Datos.* 2005.

[11]. **C.J.Date.** ¿Qué es una Base de Datos? *Introducción a los Sistemas de Bases de Datos.* La Habana : Félix Varela, 2003.

[12]. **Reisel González Pérez.** *Introducción al Sistema de Gestión de Base de Datos PostgreSQL.* Ciudad de La Habana: Universidad de las Ciencias Informáticas : s.n., 2008.

[13]. **Luis A. Guevara Alcalde.** *PostgreSQL.* Gobierno Regional de Lambayeque : s.n.

[14]. **Rafael Martínez.** PostgreSQL-es.org. *PostgreSQL-es.org.* [Online] marzo 29, 2009. [Cited: febrero 1, 2010.] http://www.postgresql-es.org/node/219.

[15]. **C.J.Date.** Arquitectura de los sistemas de bases de datos. *Introducción a los Sistemas de Bases de Datos.* La Habana : Félix Varela, 2003.

[16]. **Miguel Angel Sicilia.** Connexions. *Connexions.* [Online] enero 7, 2009. [Cited: febrero 10, 2010.] http://cnx.org/content/m17473/latest/.

[17]. **Juan Carlos Moral.** juancarlosmoral.es. *juancarlosmoral.es.* [Online] abril 27, 2007. [Cited: marzo 10, 2010.] http://www.juancarlosmoral.es/postgresql-hardware-tunning/.

[18]. **Greg Smith, Robert Treat, Christopher Browne.** PostgreSQL Wiki. *PostgreSQL Wiki.* [Online] julio 22, 2009. [Cited: marzo 11, 2010.] http://wiki.postgresql.org/wiki/Mejorando_el_rendimiento_de_tu_Postgresql_Server.

[19]. **Javier Eguíluz.** libroweb-.es. *libroweb-.es.* [Online] [Cited: marzo 31, 2010.] http://www.librosweb.es/symfony_1_0/capitulo2/el_patron_mvc.html.

[20]. **Prof. Alejandro Teruel.** Refactorización. *Refactorización.* [Online] octubre 13, 2000. [Cited: marzo 31, 2010.] http://www.ldc.usb.ve/~teruel/ci4712/clases2000/refactorizacion.html.

[21]. **Eneko Astigarraga.** El Método Delphi. *El Método Delphi.* [Online] [Cited: mayo 5, 2010.] http://www.unalmed.edu.co/~poboyca/documentos/documentos1/documentos-Juan%20Diego/Plnaifi_Cuencas_Pregrado/Sept_29/Metodo_delphi.pdf.

[22]. **Cecilia Paula Izquierdo Moreno, Beatriz Pascual Plaza, Alberto Romero Blanco, Virginia Gómez Millan.** *El Método Delphi.*

1. **Carlos D. González.** usabilidadweb.com.ar. *usabilidadweb.com.ar.* [Online] octubre 13, 2009. [Cited: diciembre 2, 2009.] http://www.usabilidadweb.com.ar/metodos_eval_calidad_web.php.

2. **Macario Polo , Francisco Ruiz.** Mantenimiento del Software. *Mantenimiento del Software.* [Online] 2000/2001. [Cited: noviembre 10, 2009.] http://alarcos.inf-cr.uclm.es/per/fruiz/cur/mso/trans/s1.pdf.

3. **Miguel Angel Sicilia.** Connexions. *Connexions.* [Online] noviembre 27, 2008. [Cited: noviembre 10, 2009.] http://cnx.org/content/m17452/latest/.

4. **Verónica De la Morena.** Conexxions. *Conexxions.* [Online] septiembre 4, 2008. [Cited: noviembre 17, 2009.] http://cnx.org/content/m17399/latest/.

5. **Juan Cesar Martínez, Marina E Ramírez Andonegui.** Norma ISO 14764 Mantenimiento de Software. *Norma ISO 14764 Mantenimiento de Software.* [Online] [Cited: noviembre 17, 2009.] http://www.ci.ulsa.mx/~elinos/docencia/ctrldesa/ISO14764.pdf.

6. **Miguel Angel Sicilia.** Connexions. *Connexions.* [En línea] 7 de enero de 2009. [Citado el: 10 de febrero de 2010.] http://cnx.org/content/m17473/latest/.

Janhil Aurora Trejo Martinez. monografías.com. *monografías.com.* [Online] [Cited: enero 12, 2010.] http://www.monografias.com/trabajos11/basda/basda.shtml#bas.

7. **Damián Pérez Valdés .** maestros del web. *maestros del web.* [Online] [Cited: enero 12, 2010.] http://www.maestrosdelweb.com/principiantes/%C2%BFque-son-las-bases-de-datos/.

8. **Rafael Martínez.** PostgreSQL-es.org. *PostgreSQL-es.org.* [Online] marzo 29, 2009. [Cited: febrero 1, 2010.] http://www.postgresql-es.org/node/219.

9. **Miguel Angel Sicilia.** Connexions. *Connexions.* [Online] enero 7, 2009. [Cited: febrero 10, 2010.] http://cnx.org/content/m17473/latest/.

10. **David Garlan, Mary Shaw.** An Introduction to Software Architecture. *An Introduction to Software Architecture.* [Online] enero 1994. [Cited: marzo 1, 2010.] http://www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf.

11. **Juan Carlos Moral.** juancarlosmoral.es. *juancarlosmoral.es.* [Online] abril 27, 2007. [Cited: marzo 10, 2010.] http://www.juancarlosmoral.es/postgresql-hardware-tunning/.

12. **León Welicki.** Patrones y Antipatrones: una Introducción - Parte II. *Patrones y Antipatrones: una Introducción - Parte II.* [Online] [Cited: marzo 10, 2010.] http://msdn.microsoft.com/es-es/library/bb972251.aspx.

13. **León Welicki.** Patrones y Antipatrones: una Introducción - Parte II. *Patrones y Antipatrones: una Introducción - Parte II.* [Online] [Cited: marzo 11, 2010.] http://msdn.microsoft.com/es-es/library/bb972251.aspx.

14. **Greg Smith, Robert Treat, Christopher Browne.** PostgreSQL Wiki. *PostgreSQL Wiki.* [Online] julio 22, 2009. [Cited: marzo 11, 2010.] http://wiki.postgresql.org/wiki/Mejorando_el_rendimiento_de_tu_Postgresql_Server.

15. **Josh Berkus.** Toolbox for IT. *Toolbox for IT.* [Online] diciembre 30, 2008. [Cited: marzo 26, 2010.] http://it.toolbox.com/blogs/database-soup/writing-maintainable-queries-part-i-29052.

16. **Leo Hsu, Regina Obe.** SQL Coding Standards To Each His Own . *SQL Coding Standards To Each His Own .* [Online] diciembre 31, 2008. [Cited: marzo 26, 2010.] http://www.postgresonline.com/journal/index.php?/archives/94-SQL-Coding-Standards-To-Each-His-Own.html.

17. **Regina Obe, Leo Hsu.** SQL Coding Standards To Each His Own Part II. *SQL Coding Standards To Each His Own Part II.* [Online] enero 6, 2009. [Cited: marzo 26, 2010.] http://www.postgresonline.com/journal/index.php?/archives/97-SQL-Coding-Standards-To-Each-His-Own-Part-II.html..

18. **Hubert Lubaczewski.** DSHL. *DSHL.* [Online] enero 4, 2009. [Cited: marzo 26, 2010.] http://www.depesz.com/index.php/2009/01/04/maintainable-queries-my-point-of-view/.

19. **Prof. Alejandro Teruel.** Refactorización. *Refactorización.* [Online] octubre 13, 2000. [Cited: marzo 31, 2010.] http://www.ldc.usb.ve/~teruel/ci4712/clases2000/refactorizacion.html.

20. **Josh Berkus.** Toolbox for IT. *Toolbox for IT.* [Online] mayo 1, 2009. [Cited: marzo 26, 2010.] 26. http://it.toolbox.com/blogs/database-soup/writing-maintainable-queries-part-ii-29120.

21. **thierry alemany.** Programación en castellano. *Programación en castellano.* [Online] agosto 20, 2003 . [Cited: marzo 31, 2010.] http://www.programacion.com/articulo/refactorizacion:_camino_hacia_la_calidad_221.

22. **Javier Eguíluz.** libroweb-.es. *libroweb-.es.* [Online] [Cited: marzo 31, 2010.] http://www.librosweb.es/symfony_1_0/capitulo2/el_patron_mvc.html.

23. Verificación y Validación. *Verificación y Validación.* [Online] [Cited: mayo 5, 2010.] http://wwwdi.ujaen.es/asignaturas/computacionestadistica/pdfs/tema6.pdf.

24. **Eneko Astigarraga.** El Método Delphi. *El Método Delphi.* [Online] [Cited: mayo 5, 2010.] http://www.unalmed.edu.co/~poboyca/documentos/documentos1/documentos-Juan%20Diego/Plnaifi_Cuencas_Pregrado/Sept_29/Metodo_delphi.pdf.

25. **Lic. Alina Maceo Sánchez, Ms. C. María del Carmen Fernández Coira.** Fuentes de Información para la Inteligencia Competitiva en I+D. *Fuentes de Información para la Inteligencia Competitiva en I+D.* [Online] 2004. [Cited: mayo 10, 2010.] http://www.bibliociencias.cu/gsdl/collect/eventos/index/assoc/HASH3ce4.dir/doc.pdf.

26. **C.J.Date.** ¿Qué es una Base de Datos? *Introducción a los Sistemas de Bases de Datos.* La Habana : Félix Varela, 2003.

27. **Universidad Cooperativa de Colombia Sede Villamaría.** *Análisis y Diseño.*

28. **C.J.Date.** Arquitectura de los sistemas de bases de datos. *Introducción a los Sistemas de Bases de Datos.* La Habana : Félix Varela, 2003.

29. **Beatriz Pereyra.** *Bases de Datos.* 2005.

30. **Cecilia Paula Izquierdo Moreno, Beatriz Pascual Plaza, Alberto Romero Blanco, Virginia Gómez Millan.** *El Método Delphi.*

31. **Juan Antonio López Quesada.** *Mantenimiento del Softeare.*

32. **Reisel González Pérez.** *Introducción al Sistema de Gestión de Base de Datos PostgreSQL.* Ciudad de La Habana: Universidad de las Ciencias Informáticas : s.n., 2008.

33. **Ing. Roberto Hugo Vázquez.** *Introducción a la Calidad del Software.* s.l. : marzo, 2006.

34. **Luis A. Guevara Alcalde.** *PostgreSQL.* Gobierno Regional de Lambayeque : s.n.

Anexo 1. Respuestas de las Preguntas de Diseño

Respuesta a la pregunta 1:

No son las únicas consideraciones a tener en cuenta, se pueden agregar las siguientes:

- El tamaño de la información.
- Facilidad de acceso a la información.
- El comportamiento del manejador de bases de datos con cada tipo de información.

Respuesta a la pregunta 2:

Un buen diseño de base de datos es aquél que:

__V__ Divide la información en tablas basadas en temas para reducir los datos redundantes.

__V__ Proporciona la información necesaria para reunir la misma de las tablas cuando así se precise.

__V__ Ayuda a garantizar la exactitud e integridad de la información.

__V__ Satisface las necesidades de procesamiento de los datos y de generación de informes.

Respuesta a la pregunta 3:

No es fundamental ignorar la Normalización para obtener un buen rendimiento y una sencilla programación en el desarrollo una base de datos La Normalización es un conjunto de reglas que sirven para ayudar a los diseñadores a desarrollar un esquema que minimice los problemas de lógica. Una de las ventajas de ella es el consumo de espacio, es decir, hay menos repetición de datos lo que tiene como consecuencia un menor uso de espacio en disco. La normalización ayuda a clarificar la base de datos y a organizarla en partes más pequeñas y más fáciles de entender.

Respuesta a la pregunta 4:

Si se puede afirmar que los aspectos citados forman parte de los objetivos principales de la Normalización.

Respuesta a la pregunta 5:

Es importante porque con la presencia o existencia de los índices se evita un escaneo completo de la tabla, lo que hace que cuando haya grandes cantidades de datos en las tablas este elemento sea muy necesario. Se evita también problemas de sobrecarga de CPU, sobrecarga de disco y concurrencia.

Anexo 2. Respuestas de las Preguntas de Configuración

Respuesta a la pregunta 1:

Para responder las inquietudes planteadas anteriormente se debe empezar por:

- Considerar un número superior al actual de "*shared_buffers"*.
- Modificar el tamaño del segmento si no cabe el número de buffers.
- Comprobar el rendimiento y paginación.
- En función del resultado obtenido, aumentar o disminuir el porcentaje de memoria y empezar de nuevo.

Una buena recomendación es empezar asignando un 10% del total de la memoria RAM para *shared_buffers* y a partir de ahí, ir aumentando o disminuyendo dicho porcentaje en función del rendimiento y la paginación. (1048576 KB/10) = 104857 KB, shared_buffers: (104857 KB/8 KB) = 13107.

Nota: *Para comprobar el rendimiento se debe aplicar EXPLAIN a las consultas. Para ver la paginación del servidor se puede usar herramientas como "vmstat" o "ipcs" (consulta sus páginas man).*

Una vez realizado esto se puede observar que el primer intento ha fallado mostrando:
FATAL: no se pudo crear el segmento de memoria compartida: Argumento inválido.
DETALLE: La llamada a sistema fallida fue shmget (key=5432001, size=112009216,03600).
HINT: Este error significa que una petición de PostgreSQL para obtener un segmento de memoria compartida excedió el parámetro "SHMMAX" del Kernel.

Se puede reducir el tamaño de la petición o reconfigurar el Kernel con un "SHMMAX" superior. Para reducir el tamaño de la petición (actualmente 112009216 bytes), reduzca el parámetro de PostgreSQL shared_buffers (actualmente 13107) y/o el parámetro max_connections (actualmente 100). Si el tamaño de la petición ya es pequeño, es posible que sea inferior al parámetro SHMMIN del Kernel, en cuyo caso se requiere alzar el tamaño de la petición o disminuir SHMMIN.

Es evidente el tamaño ocupado por los 13107 buffers que se ha pedido reservar, simplemente no caben en el tamaño actual de segmento (SHMMAX). La solución está en modificar el tamaño máximo del segmento de memoria compartida. Esto lo hacemos asignando un nuevo valor al parámetro del Kernel SHMMAX. ¿Qué valor? Si volvemos atrás, al mensaje de error, avisa exactamente de cuál es el tamaño

mínimo que PostgreSQL necesita para arrancar. El tamaño de SHMMAX debe ser, como mínimo, ese valor.

Respuesta a la pregunta 2:

Se considera importante este proceso de autovacuum ya que realiza una limpieza de tuplas muertas que han sido marcadas como borradas o modificadas, lleva a cabo una serie de operaciones de mantenimiento en la base de datos que Ud. necesita, además se recomienda usarlo más seguido.

Anexo 3. Respuestas de las Preguntas de Arquitectura

Respuesta a la pregunta 1:

__F__ El uso de patrones y estilos arquitectónicos afectan negativamente la mantenibilidad del software.

__V__ Utilizar el Patrón Modelo Vista Controlador (MVC) aumenta la mantenibilidad.

__V__ El uso de la Arquitectura en Capas propicia la facilidad del software para ser cambiado, para ser probado y para adaptarse de manera estable a diferentes cambios.

__V__ El uso del trabajo por Módulo propicia la facilidad del software para ser cambiado, para ser probado y para adaptarse de manera estable a diferentes cambios.

Respuesta a la pregunta 2:

El trabajo por módulo facilita el mantenimiento ya que mediante este se puede descomponer el trabajo en un conjunto de submódulos independientes entre sí, permitiendo que estos sean más sencillos y más fáciles de encontrar errores en caso de que ocurran, lo que permite que estos errores no lleguen a su programa principal. Por esta razón se puede decir que la recomendación del desarrollador provocará éxito en el trabajo.

Anexo 4. Respuesta de la Pregunta de Codificación

Respuesta a la pregunta 1:

__F__ Utilizando estructuras diferentes para programar se logra que la mantenibilidad del código se eleve rápidamente.

__V__ Un orden determinado en el código facilita el mantenimiento del mismo.

__F__ Con un estándar de codificación se puede asegurar un código con calidad pero no se eleva la mantenibilidad del mismo.

__V__ El mejor método para asegurarse de que un equipo de desarrolladores mantenga un código con calidad elevando la mantenibilidad del mismo es estableciendo un estándar de codificación.

__F__ Un estándar de codificación tiene que presentar técnicas para el nombrado, la documentación externa y el formato.

Anexo 5. Encuesta de Autovaloración

La presente encuesta tiene como finalidad la evaluación de la Guía de buenas prácticas para obtener mantenibilidad en las bases de datos desarrolladas en PostgreSQL, es necesario que responda todas las preguntas; la encuesta presenta un carácter anónimo.

1. Valore su conocimiento en el campo de la mantenibilidad en bases de datos, marque con una X en una escala del 0 al 10.

0	1	2	3	4	5	6	7	8	9	10

2. Seleccione el grado que usted crea que han influenciado las siguientes fuentes de conocimiento.

FUENTES DE ARGUMENTACIÓN	Grado de influencia de cada una de las fuentes en sus criterios.		
	A (alto)	M(medio)	B (bajo)
Análisis teóricos realizados por usted.			
Su experiencia obtenida.			
Trabajos de autores nacionales.			
Trabajos de autores extranjeros.			
Su propio conocimiento del estado del problema en el extranjero.			
Su intuición.			

Anexo 6. Encuesta a Especialistas

1**-** Cree usted que es de gran importancia la creación y utilización de la guía de buenas prácticas para ayudar a los desarrolladores de bases de datos a elevar la mantenibilidad en las mismas.

____Si ____No ____No sé

¿Por qué?

_____

_____

_____

2- Mencione usted si existen razones o cuestiones en el trabajo de los desarrolladores de bases de datos por las cuales el uso de la guía de mantenibilidad.

___Se debe recomendar ____No se debe recomendar ____Se recomienda una parte

¿Por qué?

_____

_____

_____

3- Ve usted dificultades o alguna complejidad en el proceso de aplicación de la guía para el desarrollador de bases de datos.

____Si ____No

¿Por qué?

_____

_____

_____

4- Cree usted que es abarcadora la guía de buenas prácticas. En una escala del 1 al 5 indique que tan abarcadora es. En caso de su respuesta ser diferente de 5, diga las razones, por las que la guía está incompleta.

1	2	3	4	5

Nota: Tenga en cuenta que 1 quiete decir que la guía no es abarcadora y 5 que lo abarca todo.

_____

_____

_____

5- Diga si la guía de buenas prácticas está facultada para cumplir las siguientes expectativas, para ello marque con una X las que usted cree que pueda cumplir. Escriba el por qué de su selección:

a) ___ Necesidad del empleo de la propuesta.

b) ___ Adaptabilidad a proyectos productivos.

c) ___ Efectividad de su utilización.

_____

_____

_____

6- Cree que la utilización de la guía cumple con el objetivo para la cual fue creada (ayudar a elevar la mantenibilidad en bases de datos). Indíquelo en una escala del 1 al 5. En caso de su respuesta ser diferente de 5, diga las razones, por las que la guía no cumple el objetivo.

1	2	3	4	5

Nota: Tenga en cuenta que 1 quiete decir que la guía no cumple con el objetivo y 5 que lo cumple.

_____

_____

_____

ANSI/SPARC: Instituto Nacional Americano de Estándares.

ASP: Active Server Pages.

BSD: Licencia de software otorgada para los sistemas Berkeley Software Distribution.

CASE: Computer Aided Software Engineering.

DBMS: Database Management System.

ISO: International Organization for Standardization.

MS: Mantenimiento del Software.

MVC: Modelo Vista Controlador.