ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

"ESTUDIO DEL LENGUAJE INTEGRADO DE CONSULTAS (LINQ), APLICADO AL DESARROLLO DEL SISTEMA DE INVENTARIO Y FACTURACIÓN DE LA LIBRERÍA POLITÉCNICA"

TESIS DE GRADO

Previa a la obtención del título de INGENIERO EN SISTEMAS INFORMÁTICOS

Presentado por:

René Alfonso Barragán Torres

Paúl Xavier Paguay Soxo

Riobamba – Ecuador

2008

AGRADECIMIENTO

Agradezco a mis padres por guiar mi camino, ser el pilar fundamental en mi vida, a mis hermanos, a mis primos Fernando y Shirley quienes me han apoyado en todas las decisiones que he tomado y me han brindado todo su cariño y a toda mi familia por su apoyo incondicional.

René Barragán Torres

"*La gratitud es la más bella joya del corazón*"

En todo este caminar para lograr este objetivo, existen muchas personas a las cuales deseo agradecer pero sobre todo a mi padre que siempre confió en mí, a mi madre que constantemente me apoyó, a mi hermana, a todos mis amigos sinceros y a la persona que ha sido mi inspiración, Dorian.

Paúl Xavier Paguay Soxo

DEDICATORIA

Dedicamos este trabajo a nuestros padres que sin su apoyo no hubiese sido posible y a los buenos profesores que ponen el corazón en su profesión, ayudando a sus estudiantes en su preparación profesional, haciéndoles ver sus errores para corregirlos y alentándolos a seguir mejorando cuando logren grandes metas en su camino universitario.

FIRMAS DE RESPONSABILIDAD Y NOTA DEL TRIBUNAL

"Nosotros, René Alfonso Barragán Torres y Paúl Xavier Paguay Soxo, somos los responsables de las ideas, doctrinas y resultados expuestos en esta: Tesis, y el patrimonio intelectual de la misma pertenecen a la Escuela Superior Politécnica de Chimborazo"

René Alfonso Barragán Torres

Paúl Xavier Paguay Soxo

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

En esta tesis se presentan los resultados de la investigación realizada para determinar si la nueva tecnología para el acceso a datos (LINQ), mejorará la productividad al momento de trabajar con datos habilitados.

El objetivo principal de esta tesis es realizar un estudio sobre la nueva tecnología LINQ (Language Integrated Query), para el desarrollo del sistema de inventario y facturación de la librería politécnica.

En adelante se mostrará el resultado del estudio de LINQ (Language Integrated Query), además del análisis de las nuevas características del framework 3.5 el cual permite utilizar LINQ (Language Integrated Query)

Al analizar las métricas a utilizar se mostrará de manera detallada los parámetros a utilizar para lograr la demostración de nuestra hipótesis planteada.

En el desarrollo del sistema de inventario y facturación se utilizaron seis módulos (Módulo Productos, Módulo Clientes, Módulo Facturación, Módulo Proveedores, Módulo Autenticación, Módulo de Consultas), los cuales se detallarán a continuación. La metodología empleada para el desarrollo del sistema de inventario y facturación de la librería politécnica es Craig Larman. La metodología se encuentra detallada en la documentación entregada en anexos.

El primer capítulo de esta tesis trata sobre el marco referencial en el cual se encuentra de manera general la justificación del proyecto de tesis además de los objetivos a alcanzar con el desarrollo de la tesis.

En el segundo capítulo se describe a LINQ (Language Integrated Query), su significado como trabajar con LINQ, su sintaxis y los operadores que utiliza.

Para el tercer capítulo se ha previsto estudiar LINQ en visual Basic 9.0 ya que es el lenguaje de programación elegido para desarrollar el sistema de inventario y facturación de la librería politécnica.

El desarrollo de sistema de inventario y facturación de la librería politécnica se encuentra en el capítulo cuarto, sus diagramas de casos de uso, diagramas de secuencias, diagramas de estado, diagramas de calles y las posibles pantallas a utilizar así como la arquitectura física y lógica de la aplicación.

El estudio de resultados se encuentra en el capítulo quinto. En este capítulo se detallara los parámetros utilizados para la comprobación de la hipótesis planteada.

CAPÍTULO I

MARCO REFERENCIAL

1.1 Antecedentes

En la actualidad el acceso a datos desde una aplicación consta de una serie de pasos entre estos esta el crear conexiones y sentencias TRANSACT-SQL, para lograr extraer la información desde un repositorio de datos, lo cual implica que un programador debe aprender tanto el lenguaje para construir la aplicación, como el de extracción de datos, esto se complica más si se toma en cuenta que la información puede provenir de diferentes orígenes de datos, como pueden ser los archivos xml u objetos creados, con lo cual se necesitaría especialistas en cada uno de estos campos para poder terminar un proyecto de software que cumpla con los requerimientos del usuario y que a su vez el tiempo de desarrollo de la aplicación no se amplíe más de lo planificado.

Por otro lado, una parte importante, en cuanto a la extracción de datos de una manera desconectada es el uso de los datasets, los cuales nos permiten mantener

los datos temporalmente en memoria cache, evitando realizar varias peticiones al servidor, pero así mismo este objeto no ofrece facilidades para generar consultas sobre estos datos recuperados.

LINQ es una tecnología que simplifica el acceso a la información situada en un repositorio de datos, ya que trabaja con un estándar similar al TRANSACT-SQL para la extracción de datos, lo que permite realizar consultas en tiempo de ejecución, ahorrando el tiempo de un programador. LINQ integra el lenguaje de consultas para la extracción de datos (TRANSACT-SQL), y el lenguaje para la construcción de una determinada aplicación (Visual Basic).

LINQ to DataSet facilita y acelera las consultas en datos almacenados en caché en un objeto DataSet. Esas consultas se expresan en el lenguaje de programación mismo, en lugar de cadenas incrustadas en el código de la aplicación. Esto significa que los desarrolladores no tienen que aprender un lenguaje de consultas diferente.

LINQ (Language Integrated Query) es un nuevo conjunto de herramientas diseñado para reducir la complejidad del acceso a Base de Datos, a través de extensiones para C# y Visual Basic así como para Microsoft .NET Framework 3.0. Permite filtrar, enumerar, y crear proyecciones de muchos tipos y colecciones de datos utilizando la misma sintaxis, prescindiendo del uso de lenguajes especializados como SQL o XPath.

En lo concerniente a los antecedentes de la Librería ESPOCH, cabe recalcar que

el departamento ha contado con un sistema anterior, pero que en la actualidad ya no se lo ocupa, en la misma trabajan dos personas que se encargan de todas las labores, el administrador es el Licenciado Luís Benalcázar. Entre las actividades que se realizan en la librería está, el registro de artículos, actualización del stock, el precio de compra, precio de venta al público, a si mismo la información de los proveedor y la actualización de la misma, Generación y Registro de facturas, notas de venta y notas de entrega (para lo que son las actividades de venta a la Politécnica). Toda esta información es manejada con la Herramienta Excel de Microsoft, lo cual se ha vuelto obsoleto y redundante, el sistema operativo con el que cuentan es XP Professional, poseen una computadora de 3.0GHz, 512 MB de memoria RAM y 120 GB de disco duro, una impresora matricial, con lo que se puede empezar con el desarrollo del sistema de inventario y facturación.

1.2 Justificación

Justificación teórica

En la actualidad, la capa física, con todos los datos almacenados, en los diferentes motores, se los desarrolla y manipula de manera distinta y separada, con la nueva tecnología de LINQ se integra la manipulación de los datos al momento de extraerlos a una aplicación. Esto conlleva a un ahorro de tiempo significativo, ya que se evita que un programador aprenda un lenguaje específico para la extracción de datos como TRANSACT-SQL.

LINQ integra el lenguaje de consultas al lenguaje de aplicación lo que permite un ahorro de tiempo al momento de desarrollar una aplicación que consuma datos de un repositorio de información.

En lo concerniente al lenguaje de aplicación a emplear, se ha escogido Visual Basic 9.0, por la familiaridad con el mismo, además que es el lenguaje que se lo ha utilizado por tres años en el desarrollo de proyectos informáticos. El IDE con el que se desarrollará la aplicación es Microsoft Visual Studio 2008 ya que permite utilizar el IntelliSense, y verificación de tipo en tiempo de compilación, los mismos que son de gran ayuda para los programadores.

Justificación práctica

El desarrollo del sistema de inventario y facturación para la librería politécnica mejorará el control de los artículos existentes y el tiempo requerido en la atención al cliente.

Al sistema se lo ha dividido en varios módulos los cuales se detallan a continuación:

- ✓ Control de Inventario
  - o Registro de artículos (ingresos, actualizaciones, modificaciones de sus datos)
  - o Consultas sobre las estadísticas de los artículos
  - o Registro de los proveedores (ingresos, actualizaciones, modificaciones de sus datos)

- ✓ Facturación
  - o Registro de clientes

- Registro de departamentos de la ESPOCH
- Registro de Facturas, Notas de Venta
- Notas de Entrega (adecuado exclusivamente para la utilización de la librería ESPOCH)
- Impresión de los comprobantes de venta

Con la información citada, debidamente registrada en el sistema, se generarán reportes informativos y estadísticos que sirvan para el apoyo a la toma de decisiones.

1.3 Objetivos

1.3.1 Objetivo General

Realizar un estudio sobre la nueva tecnología LINQ (Language Integrated Query) que ofrece Microsoft en su Framework 3.0, para el desarrollo del sistema de inventario y facturación de la librería politécnica.

1.3.2 Objetivos Específico

- Estudiar una tecnología que nos permita la integración de consultas de capa física en un mismo lenguaje
- Analizar las características del Framework 3.0 de Microsoft, para poder determinar los aspectos más sobresalientes sobre la tecnología LINQ.
- Analizar las métricas a utilizar para poder determinar la productividad.

- Desarrollar el sistema de inventario y facturación de la Librería ESPOCH
- Comparar la productividad de una aplicación construida con la tecnología LINQ frente a una aplicación desarrollada con las técnicas tradicionales de acceso a datos.

1.4 Hipótesis

La utilización de la tecnología integrada de consultas, definida como LINQ (Lenguaje Integrated Query) mejorará la productividad a la hora de crear aplicaciones empresariales de datos habilitados.

CAPÍTULO II

LINQ

2.1 Introducción a LINQ

2.1.1 ¿Que es LINQ?

Lenguaje Integrated Query (LINQ) es un modelo de programación que introduce consultas nativas semejantes a SQL embebidas en cualquier tipo de lenguaje Microsoft .NET (Inicialmente con Visual Basic 9.0 y C# 3.0).

2.1.2 Ventajas de LINQ

- Mejora la productividad, reduciendo la necesidad para los desarrolladores de aprender y usar lenguajes múltiples cuando construyen aplicaciones, lo hace extendiendo el .NET Framework para incluir operaciones de consultas semejantes a las de SQL integradas en el lenguaje de programación.

- Provee de un conjunto de extensiones para los lenguajes .NET así como un conjunto de librarías que provee de consultas integradas para objetos, datos XML y bases de datos usando sintaxis de lenguajes nativos.

2.1.3 ¿Cómo trabajar con LINQ?

Para entender a lo que se refiere con Lenguaje Integrado de Consultas, veamos el siguiente ejemplo:

```
Customer[] Customers = GetCustomers();
var query =
from c in Customers
where c.Country == "Italy"
select c;
```

La compilación de este código dará el siguiente resultado:

```
Customer[] Customers = GetCustomers();
IEnumerable<Customer> query =
Customers
.Where( c => c.Country == "Italy" );
```

Luego de observar este ejemplo, podemos ir separando las características del lenguaje:

Por una parte está la aparición de la clase “IEnumerable”, la misma que es la base sobre la cual LINQ trabaja, LINQ puede aplicarse sobre listas, sean estas de objetos, datos, estructuras, etc. En otras palabras que se pueda contar.

Por otra parte podemos observar que dentro de la consulta “Where” se hace llamadas a otras funciones “*c => c.Country == "Italy*”, esto sucede porque LINQ hace llamadas a métodos denominados de “Extensión”, que en la actualidad han sido incorporados en las nuevas características de los lenguajes C# 3.0 y VB 9.0, esto se profundizará en el capítulo III – “Lenguajes y Sintaxis de LINQ en Visual Basic 9.0”.

Otro aspecto importante es que LINQ no se ejecuta mientras no tenga recuperado datos, que es ahí donde es el verdadero trabajo de LINQ, el acceso a los datos. LINQ actúa sobre datos recuperados.

```
var query = from c in Customers ...
foreach ( string name in query ) ...
```

Hay otros métodos que integra LINQ, la cual es el de producir una copia persistente de datos en memoria. Por ejemplo el método “ToList” produce un tipo de List <T> del namespace “System.Collection.Generic”.

Ejemplo:

```
var query = from c in Customers ...
List<Customer> customers = query.ToList();
```

Cuando se actúa sobre base de datos como Microsoft SQL Server, LINQ genera una sentencia SQL equivalente para la obtención de datos, en vez de obtener los datos y procesar dichos datos en memoria, estas sentencias no serán enviadas al servidor de base de datos hasta que el bucle foreach sea ejecutado o el método "ToList" sea llamado.

2.1.4 Modelo relacional y modelo jerárquico

A simple vista, LINQ puede aparecer como otro dialecto SQL, esto tiene su origen por la manera relacional como se llevan los datos:

```
var query =
from c in Customers
join o in Orders
on c.CustomerID equals o.CustomerID
select new { c.CustomerID, c.CompanyName, o.OrderID
};
```

Esto es similar a la forma regular de consulta de datos en un modelo relacional. Sin embargo, LINQ no es limitado a un único dominio de datos como el modelo relacional lo es. En un modelo jerárquico, supongamos que cada cliente tiene su propio conjunto de órdenes, y cada orden tiene su propia lista de los productos. En LINQ, podemos obtener la lista de productos solicitados por cada cliente de la siguiente manera:

```
var query =
from c in Customers
from o in c.Orders
select new { c.Name, o.Quantity,
o.Product.ProductName };
```

La consulta anterior no contiene ninguna suma. La relación entre clientes y pedidos es expresada por el segundo de la cláusula, que utiliza c.Orders a decir "obtener todas las órdenes de la c Cliente. "La relación entre las órdenes y los productos se expresa en el miembro del Producto de la instancia de la Orden. El será el nombre del producto para cada fila de órdenes utilizando o.Product.ProductName.

Las relaciones jerárquicas son expresadas en tipos definidos con referencias a otros objetos como podemos ver en el siguiente ejemplo:

```
public class Customer {
public string Name;
public string City;
public Order[] Orders;
}
public struct Order {
public int Quantity;
public Product Product;
}
public class Product {
public int IdProduct;
public decimal Price;
public string ProductName;
}
```

2.1.5 Futuro de LINQ

LINQ hizo su primera aparición en septiembre de 2005 como una vista previa técnica. Desde entonces, ha evolucionado de una extensión de Visual Studio 2005 a una parte integrada en la versión del .NET Framework (versión 3.5) y en el lanzamiento del último Visual Studio (cuyo nombre en código es "Orcas").

La primera versión de LINQ apoya directamente algunos campos de datos, como "Tipos LINQ" (To XML, To SQL, To Object). Sin embargo, LINQ se puede ampliar para apoyar otros campos

de datos. Posibles extensiones podría ser algo como LINQ a SharePoint, LINQ a Exchange, y LINQ para OLAP, sólo para nombrar unos pocos ejemplos. En realidad, algunos de las posibles implementaciones ya están disponibles mediante LINQ a los objetos.

Otro punto de extensibilidad de LINQ es el proveedor de modelo incluido en LINQ para SQL y LINQ para entidades. Las actuales versiones de LINQ apoyan sólo las bases de datos SQL Server, pero es posible a la aplicación de un proveedor de cualquier otra base de datos relacional.

LINQ es probable que tenga un impacto sobre la forma en las solicitudes están codificados. Sería un error LINQ creo que va a cambiar la aplicación de arquitecturas, porque su objetivo es proporcionar un conjunto de herramientas que mejoren la ejecución de código mediante la adaptación a diferentes arquitecturas. Sin embargo, actualmente no podemos evitar decir que LINQ podría afectar a algunas partes críticas de las n capas de una solución de nivel. Por ejemplo, podemos imaginar el uso de LINQ en SQLCLR (Common Languaje Runtime) un procedimiento almacenado, con una transferencia directa de la consulta a la expresión del motor de SQL en lugar de utilizar un SQL declaración.

Muchos posibles evoluciones podrían provenir de LINQ, y no se debe olvidar que es un SQL ampliamente adoptado la norma que no puede ser fácilmente reemplazado por otro, sólo por razones de rendimiento. Sin embargo, LINQ es un interesante paso en la evolución actual de la programación. El carácter declarativo de su sintaxis puede ser interesante para otros usos que el acceso a los datos. Otro estudio sobre LINQ es PLINQ, un proyecto de investigación sobre LINQ en paralelo, sobre varios procesadores.

Muchos otros servicios pueden ser ofrecidos por una ejecución de programa escrito utilizando un mayor nivel de abstracción, como el ofrecido por un LINQ. Una buena comprensión de esta nueva tecnología podría ser importante el día de hoy, pero podría llegar a ser fundamental mañana.

2.2 Sintaxis Fundamental de LINQ

Modernos lenguajes de programación y desarrollo de software basados en arquitecturas son más y más sobre el diseño orientado a objetos.

Como resultado de ello, muy a menudo es necesario consultar y manejar objetos y colecciones de objetos, en lugar de los registros y tablas de datos. También necesitamos de herramientas y lenguajes específicos para diferentes orígenes de datos o diferentes capas. El Lenguaje Integrado de Consultas (LINQ) permite a los desarrolladores la consulta y manejo de secuencias de ítems (objetos, entidades, registros de bases de datos, nodos XML, etc) dentro de sus soluciones

de software, utilizando un único lenguaje de programación independiente de su origen. La característica fundamental de LINQ es su integración con el ampliamente utilizado lenguaje de programación sobre datos Transact SQL.

2.2.1 Consultas LINQ

LINQ se basa en un conjunto de operadores de consulta, que se define como los métodos de extensión, que todo el trabajo con cualquier objeto que implementa IEnumerable <T>. (Para más detalles acerca de los métodos de extensión, véase el capítulo III, "Características del Lenguaje Microsoft Visual Basic 9.0"). Este enfoque transforma a LINQ en un framework de consultas de propósito general debido a que muchos listas aplican el tipo IEnumerable(T), y cualquier desarrollador puede implementarlo de acuerdo a sus necesidades. Este framework de consulta también es muy ampliable. Dando a la arquitectura de la solución los métodos de extensión, los desarrolladores pueden especificar método de acceso a los orígenes como LINQ para SQL y XML LINQ a Objetos, y todas las que vengan.

2.2.2 Sintaxis de las consultas

Qué es una expresión de consulta

Una *consulta de expresión* es un árbol de expresión que opera sobre una o más fuentes de información mediante la aplicación de uno o más operadores de consulta, ya sea el grupo de la norma consulta de los operadores o de dominio específico de los

operadores. En general, la evaluación de una consulta expresión resultados en una secuencia de valores. Una consulta de expresión se evalúa sólo cuando su contenido se enumera.[1]

Las consultas LINQ están escritas en C# o en VB.NET (actualmente no hay soporte LINQ para otros lenguajes) y sigue una sintaxis que es similar a la sintaxis de XQuery FLWR (For-Let-Where-Return) y que tiene el siguiente aspecto:

var query = from [variable] in [source set]
group [group syntax]
where [criteria]
orderby [column syntax]
select [variable | projection]

2.2.2 Operadores

Para la utilización de los operadores de consultas de LINQ se debe tener referenciado nuestra aplicación con el namespace "System.Linq":

Operadores Where

Este operador también conocido como operador de restricción, nos permite realizar filtros a nuestros datos que queremos recuperar.

([1])Paolo,Marco. "*Introducing Microsoft LINQ*". (Mayo.2007). Pag 73.

```
var expr =
from c in customers
where c.Country == “Italia”
select new { c.Name, c.City };
```

Como podemos observar en el ejemplo, estamos recuperando una lista de clientes los cuales cumplan con la condición de que el país sea Italia.

Operadores de Proyección

Estos operadores sirven para seleccionar (proyectar) los datos desde un origen IEnumerable hacia un resultado IEnumerable.

- ✓ **Select**

  Con este operador, se puede extraer campos específicos en una consulta hacia un origen de datos.

```
var expr = customers.Select(c => c.Name);
```

Como podemos observar en el ejemplo, se llena una lista de tipo IEnumerable, con los nombres de los clientes.

Operadores de Ordenado

Otro de los operadores son los usados para determinar el orden y la dirección de la salida de la información.

- ✓ **OrderBy and OrderByDescending**

  Estos comandos son de amplia ayuda al momento de aplicar el ordenamiento a los resultados de una consulta, con LINQ podemos ordenar los datos de forma ascendente y descendente.

  ```
  var expr =
  from c in customers
  where c.Country == Countries.Italy
  orderby c.Name descending
  select new { c.Name, c.City };
  ```

  Con el ejemplo anterior, estamos ordenando el conjunto de resultados de acuerdo al nombre del cliente (customer) en forma ascendente, en caso contrario se utilizaría el comando OrderByDescending.

- ✓ **ThenBy and ThenByDescending**

  En caso de que se desee ordenar con varias claves se puede hacer uso de los comandos ThenBy y ThenByDescending.

Estos comandos tienen una función parecida a la del OrderBy y el OrderByDescending, con la única diferencia en que los comandos ThenBy y ThenByDescending únicamente pueden ser aplicados luego de haber aplicado un comando de OrderBy u OrderByDescending:

```
var expr = customers
.Where(c => c.Country == Countries.Italy)
.OrderByDescending(c => c.Name)
.ThenBy(c => c.City)
.Select(c => new { c.Name, c.City } );
```

Como se observa en el ejemplo, se aplica el comando OrderByDescending, con el mismo se ordena por el Nombre del cliente, mientras que luego se aplica el comando ThenBy para poder ordenar por la ciudad del cliente.

- ✓ **Operador de Reversa**

  En algunos casos se necesita que la información de los datos mostrados se nos muestre en una forma inversa a la que se nos muestra, donde el último item aparezca en el primer lugar, para ello se utilizará el comando Reverse.

```
var expr =
customers
.Where(c => c.Country == Countries.Italy)
.OrderByDescending(c => c.Name)
.ThenBy(c => c.City)
.Select(c => new { c.Name, c.City } )
.Reverse();
```

Este comando se lo puede aplicar al final de todo resultado de conjunto de datos.

Reverse simplemente enumera una secuencia y produce los mismos valores en orden inverso. A diferencia de OrderBy, Reverse no tiene en cuenta los propios valores para determinar el orden, sino que se apoya únicamente en el orden en que los valores son producidos por la fuente subyacente. El operador OrderBy impone una ordenación sobre una secuencia de valores.

Operadores de Agrupamiento

Entre los operadores de consulta estándar también se incluye el operador GroupBy, que impone la partición en grupos de los valores de una secuencia en base a una función de extracción de

clave. El operador GroupBy devuelve una secuencia de valores IGrouping, uno para cada valor de clave distinto encontrado. Un IGrouping es un IEnumerable que adicionalmente contiene la clave que fue utilizada para extraer su contenido:

La aplicación más simple de GroupBy es similar a la siguiente:

```
string[] names = { "Albert", "Burke", "Connor", "David",
                   "Everett", "Frank", "George", "Harris"};
// agrupar por longitud
var groups = names.GroupBy(s => s.Length);
foreach (IGrouping<int, string> group in groups) {
   Console.WriteLine("Strings of length {0}", group.Key);
   foreach (string value in group)
      Console.WriteLine("  {0}", value);
}
```

Al ser ejecutado, este programa imprime lo siguiente:

```
Strings of length 6
  Albert
  Connor
  George
  Harris
Strings of length 5
```

```
  Burke
  David
  Frank
Strings of length 7
  Everett
```

Del mismo modo que Select, GroupBy permite suministrar una función de proyección que será utilizada para poblar los miembros del grupo.

```
string[] names = { "Albert", "Burke", "Connor", "David",
                   "Everett", "Frank", "George", "Harris"};
// agrupar por longitud
var groups = names.GroupBy(s => s.Length, s => s[0]);
foreach (IGrouping<int, char> group in groups) {
   Console.WriteLine("Strings of length {0}", group.Key);
   foreach (char value in group)
      Console.WriteLine("  {0}", value);}
```

Esta variante imprime lo siguiente:

```
Strings of length 6
  A
  C
  G
  H
```

Strings of length 5

B

D

F

Strings of length 7

E

Operadores de encuentro

En un programa orientado a objetos, los objetos relacionados con otros están generalmente enlazados mediante referencias a objetos fáciles de navegar. Esto generalmente no se cumple para fuentes de información externa, donde los registros de datos frecuentemente no tienen otra opción que “apuntar” a otros de manera simbólica, mediante claves externas u otros datos que permitan identificar unívocamente a la entidad apuntada. El concepto de encuentros se refiere a la operación de combinar los elementos de una secuencia con los elementos con los que ellos “coinciden” de otra secuencia.

Si se utiliza SelectMany se hace exactamente eso, buscar coincidencias de cadenas con personas cuyos nombres son esas cadenas. Sin embargo, para este propósito específico, el enfoque basado en SelectMany no es muy eficiente – recorrerá todos los elementos de people para todos y cada uno de los elementos de names.

Utilizando toda la información de este escenario – las dos fuentes de información y las “claves” por las que se deben combinar – en una única llamada a método, el operador Join es capaz de hacer un mejor trabajo:

```
string[] names = { "Burke", "Connor", "Frank", "Everett",
    "Albert", "George", "Harris", "David" };

var query = names.Join(people, n => n, p => p.Name, (n,p)
=> p);
```

Esto puede parecer complicado, pero nos permitirá ver cómo las piezas encajan: el método Join es aplicado a la fuente de datos “externa”, names. El primer argumento es la fuente de datos “interna”, people. El segundo y tercer argumentos son expresiones lambda para extraer claves de los elementos de las secuencias externa e interna, respectivamente. Estas claves son las que el método Join utiliza para buscar las coincidencias de elementos.

Aquí queremos que los propios nombres coincidan con la propiedad Name de las personas. La expresión lambda final es entonces responsable de producir los elementos de la secuencia resultante: es llamada para cada pareja de elementos coincidentes n y p, y es utilizada para dar forma al resultado. En este caso, se

ha elegido descartar n y devolver p. El resultado final es la lista de elementos Person de people cuyo Name está en la lista names.

Un pariente más potente de Join es el operador GroupJoin. GroupJoin se diferencia de Join en el modo en que se utiliza la expresión lambda que da forma al resultado: en vez de ser invocada para cada pareja individual de elementos externo e interno, será llamada solamente una vez para cada elemento externo, con una secuencia de todos los elementos internos que coinciden con ese elemento externo. Poniendo un ejemplo concreto:

```
string[] names = { "Burke", "Connor", "Frank", "Everett",
    "Albert", "George", "Harris", "David"};


var query = names.GroupJoin(people, n => n, p => p.Name,
(n, matching) =>
new { Name = n, Count = matching.Count() }
);
```

Esta llamada produce una secuencia de los nombres iniciales emparejados con la cantidad de personas que tiene ese nombre. Por lo tanto, el operador GroupJoin permite basar los resultados

en el “conjunto de coincidencias” entero para un elemento externo.

2.3 Características de IDE de Visual Studio y herramientas para LINQ

Visual Studio 2008 fue liberado (RTM) el 17 de Noviembre de 2007 en inglés, mientras que la versión en castellano no fue liberada hasta el 2 de Febrero de 2008.[2]

El nuevo framework (.Net 3.5) está diseñado para aprovechar las ventajas que ofrece el nuevo sistema operativo "Windows Vista" a través de sus subsistemas "Windows Communication Foundation" (WCF) y "Windows Presentation Foundation" (WPF). El primero tiene como objetivo la construcción de aplicaciones orientadas a servicios mientras que el último apunta a la creación de interfaces de usuario más dinámicas que las conocidas hasta el momento.

A las mejoras de desempeño, escalabilidad y seguridad con respecto a la versión anterior, se agregan entre otras, las siguientes novedades.

- ✓ **Detección de Errores**

  La mejora en las capacidades de Pruebas Unitarias permiten ejecutarlas más rápido independientemente de si lo hacen en el entorno IDE o desde la línea de comandos. Se incluye además un nuevo soporte para diagnosticar y optimizar el

[2] WIKIPEDIA®(2008).”Microsoft Visual Studio”. http://es.wikipedia.org/wiki/Microsoft_Visual_Studio. [Consulta 12-10-2008]

sistema a través de las herramientas de pruebas de Visual Studio. Con ellas se podrán ejecutar perfiles durante las pruebas para que ejecuten cargas, prueben procedimientos contra un sistema y registren su comportamiento; y utilizar herramientas integradas para depurar y optimizar.

- ✓ **Compatibilidad hacia atrás**

.NET framework 3.5 continúa la línea iniciada por Fx3.0 en cuanto al mantenimiento del CLR. Por tanto, y dado que lo único que hace es añadir ensamblados a las librerías presentes con las versiones 2.0 y 3.0 del framework, las aplicaciones actuales no se verán afectadas. Eso sí, necesitará los Service Packs 1 de ambas plataformas.

- ✓ **Generación multiplataforma**

Visual Studio 2008 incluye la capacidad de crear proyectos para múltiples plataformas .NET, es decir, la 2.0, 3.0 y 3.5, desde el mismo entorno. Por tanto, no será necesario tener VS2005 instalado para generar ensamblados para .NET 2.0.

- ✓ **Multitud de novedades en C# 3.0 y VB9**

Propiedades automáticas, delegados "relajados", inicializadores de objetos, inferencia de tipos, tipos

anónimos, métodos de extensión, funciones lambda y métodos parciales, entre otros.[3]

Pero no sólo eso, dado el punto 3 (generación multiplataforma), podremos usar estas nuevas características de nuestros lenguajes favoritos y generar para .NET 2.0.

- ✓ **LINQ**

Se trata de una de las grandes revoluciones que aporta este nuevo conjunto de herramientas. Language Integrated Query es un nuevo método de acceso a datos totalmente integrado en nuestro lenguaje habitual y de una forma muy independiente de la fuente de donde provengan (colecciones, XML, motores de bases de datos, etc.).

- ✓ **Novedades para ASP.NET**

Visual Studio, así como el nuevo framework, ya incluirán ASP.NET AJAX de serie, así como 3 nuevos controles (ListView, DataPager y LinqDataSource). Además, el IDE ha sido muy mejorado e incluye soporte para intellisense y depuración de Javascripts, ¡también para ASP.NET 2.0!, y un nuevo diseñador que permite anidar páginas maestras.

[3] TRUJILLO JOAQUÍN(2008). "10 puntos importantes sobre Visual Studio 2008 y .NET Framework 3.5". http://joaquintrujillo.wordpress.com/2008/01/16/10-puntos-importantes-sobre-visual-studio-2008-y-net-framework-35/ . [Consulta: 16-10-2008]

- ✓ **Para el desarrollo en cliente**

VS2008 incluirá nuevas plantillas de proyectos, así como un diseñador para WPF integrado con soporte para la comunicación WPF-WinForms. También se ha añadido el soporte para Firefox de la tecnología ClickOnce y XBAP (XAML Browser Applications).

- ✓ **Para el desarrollador de Office**

Se ofrece soporte total para las personalizaciones (customisations) de Office 2007, así como para las plantillas de Office 2003.

- ✓ **Para desarrollo en servidor**

Se han incluido nuevas plantillas para WCF y WF, y se han introducido mejoras interesantes en el primero, como el modelo de programación HTTP (sin SOAP) o serialización JSON.

- ✓ **Para el desarrollo en dispositivos móviles**

Hay decenas de nuevas características, como el soporte para las versiones compactas de LINQ y WPF, o, a nivel de IDE, Unit Testing for Devices.

CAPÍTULO III

LENGUAJES Y SINTAXIS DE LINQ EN VISUAL BASIC 9.0

3.1 LINQ en Visual Basic 9.0

3.1.1 Visual Basic 9.0

El nuevo Visual Basic 9.0 implementa nuevas particularidades en cuanto a la programación y en su mejor desempeño al momento de crear y desarrollar aplicaciones.

Las características más relevantes se pueden visualizar en la reducción de líneas de código por la implementación de lo siguiente:

- Inferencia de tipos
- Inicializadotes de objetos
- Métodos de extensión
- Tipos anónimos

En lo relacionado al acceso a diversos orígenes de datos soporta la tecnología LINQ.

3.1.2 Comparación entre Visual Basic 9.0 y C# 3.0

Realizando una comparación entre estos dos lenguaje vemos que son muy parecidos al momento de implementar las diversas funciones que poseen, las características implementadas por C# 3.0 las tienen visual Basic 9.0 , los programadores deberían elegir el lenguaje de programación con el que tengan mayor familiaridad al momento de desarrollar aplicaciones ya que las diferencias no son relevantes.

3.1.3 Tipos locales de inferencia

En una declaración de variable local con establecimiento implícito de tipos, el tipo de variable local se infiere de la expresión de inicializador en el lado derecho de una instrucción de declaración local. Por ejemplo, el compilador infiere los tipos de las siguientes declaraciones variables:

```
Dim population = 31719
Dim name = "Belize"
Dim area = 1.9
Dim country = New Country With { .Name = "Palau", ...}
```

De este modo, son exactamente equivalentes a las siguientes declaraciones con tipo explícito:

```
Dim population As Integer = 31719
Dim name As String = "Belize"
Dim area As Float = 1.9
    Dim country As Country = New Country With { .Name =
    "Palau", ...}
```

3.1.4 Métodos de extensión

Los métodos de extensión permiten a los desarrolladores añadir nuevo métodos al contrato público de un tipo ya existente en el CLR, sin tener que recompilar el tipo original. Los métodos de extensión permiten mezclar la flexibilidad del soporte "duck typing" de los lenguajes dinámicos de hoy con el rendimiento y la validación en tiempo de compilación de los lenguajes fuertemente tipados.

Los métodos de extensión posibilitan la aparición de una gran variedad de escenarios, y ayudan a hacer posible el poder del framework LINQ.

Ejemplo de un método de extensión simple:

Este ejemplo muestra como chequear si una dirección de email es correcta mediante el uso de un método de extensión declarado en una clase existente (string).

Vamos a usar un nombre como el siguiente para el método de extensión: IsValidEmailAddress y lo vamos a declarar dentro de la clase string el cual devolverá cierto o falso dependiendo de la dirección de correo enviada.

```
Dim email = “rene@hotmail.com.ec”
If email.IsValidEmailAddress() THEN
End If
```

La manera de agregar el método es la siguiente:

```
System.Runtime.CompilerServices
Module StringExtensions
   <Extension()> _
   Public Function IsValidEmailAddress(ByVal aString As String)
      Dim regex = New Regex("@^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$")
      return regex.IsMatch(aString);
```

```
    End Function
End Module
```

Observe que la definición de método de extensión se marca con el atributo de extensión <Extension()>. Marcar el módulo en el que se define el método es opcional, pero se debe marcar cada método de extensión. System.Runtime.CompilerServices se debe importar para tener acceso al atributo de extensión.

Dentro del cuerpo del método IsValidEmailAddress() podemos acceder a todas las propiedades/métodos/eventos públicos de la instancia del string actual, y devolver true/false dependiendo de si es un email válido o no.

Para añadir este método de extensión específico a las instancias de string de mi código, sólo usamos un “imports” estándar para importar el modulo que contiene la implementación de los métodos de extensión:

```
Imports  ConsoleApplication2.StringExtensions
```

ConsoleApplication2 es el nombre de la aplicación en la que se declaró al método de extensión

3.1.5 Expresiones de inicialización de objetos

En Visual Basic, la declaración With simplifica el acceso a varios miembros de un valor agregado sin especificar las expresión de destino varias veces. Dentro del bloque de instrucciones With, se evalúa una expresión de acceso de miembro que empieza con un punto como si éste fuera precedido por la expresión de destino de la declaración With. Por ejemplo, las declaraciones siguientes inicializan una instancia nueva Country y, posteriormente, inicializan sus campos a los valores necesarios:

```
Dim palau As New Country()
With palau
  .Name = "Palau"
  .Area = 458
  .Population = 16952
End With
```

Los nuevos ***inicializadores de objeto*** de Visual Basic 9.0 son una forma basada en expresiones de With para crear instancias de objeto complejas de forma concisa. Con los inicializadores de objeto, podemos capturar las dos declaraciones anteriores en una sola declaración local (con establecimiento de tipos implícito), como se aprecia a continuación:

```
Dim palau = New Country With { _
  .Name = "Palau", _
```

.Area = 458, _

.Population = 16952 _

}

3.1.6 Expresiones LAMBDA

Una expresión lambda es una función sin nombre que calcula y devuelve un solo valor.

Para definir una expresión lambda, usaremos la instrucción **Function** a la que le indicaremos los parámetros que recibirá y como "cuerpo" de la función solo tendrá el cálculo de una expresión en la que se "manejarán" esos parámetros (como mínimo debe haber un parámetro), y el tipo de datos de esa función es del mismo tipo que el valor "calculado".

Por ejemplo, si queremos calcular el área de un círculo, tendríamos que escribir algo como esto:

*Dim area = Function(ByVal radius As Double) Math.PI * radius * radius*

Console.WriteLine("El área de un círculo de radio 5 es " & area(5))

3.1.7 Soporte para XML

Visual Basic 9.0 es un lenguaje que implementa una particular sintaxis de LINQ apoyo a XML, Consistente en literales XML y

más tarde XML vinculante. Para describir estas características, vamos a utilizar algunas clases que forman parte de LINQ a XML: XDocument, XElement, y XAttribute.

Estas clases son suficientes para saber como representan un documento XML, elemento, y atributo, respectivamente
Ejemplo:

```
Dim ourBook As XElementourBook = _
<Book Title="Introducing LINQ">
<Author>Rene Barragan</Author>
<Author>Paul Paguay</Author>
</Book>
```

Se ha asignado a ourBook un XElement, el nombre de libros que tiene un atributo Título que contiene "La introducción de LINQ", y dos elementos interiores Autor que contienen nuestros nombres.

3.1.8 Tipos anónimos

Los tipos anónimos, nos permiten crear tipos en línea sin necesidad de definir explícitamente y formalmente una clase para ese tipo, en los tipos anónimos no se puede crear objetos de ese tipo de la forma, **dim variable tipo,** ya que si se podría ya no sería un tipo anónimo**.**

Un ejemplo de un tipo anónimo sería:

```
var link = new { Text = "google", Title = "Ir a google", HRef = "http://www.google.com/" };
```

Solo se lo puede crear de esta manera y Visual Basic 9.0 lo interpreta.

Los tipos anónimos son de gran ayuda al momento de trabajar con, LINQ ya que permite crear un sinfín de tipos no establecidos tácitamente.

3.2 APLICANDO LINQ

3.2.1 Introducción a linq con objetos

3.2.1.1 Introducción de LINQ a Objetos

El proveedor LINQ to Objects permite consultar colecciones y matrices en memoria. Si un objeto admite las interfaces IEnumerable o IEnumerable<(Of <(T>)>), el proveedor LINQ to Objects permite consultarlo.

Puede habilitar el proveedor LINQ to Objects importando el espacio de nombres System.Linq, que se importa de forma predeterminada para todos los proyectos de Visual Basic.

3.2.1.2 APIS para LINQ a Objetos

La API de Linq to Objects permite consultas sobre cualquier colección .NET en memoria, como arrays y listas genéricas. Su

nombre es System.Linq que se encuentra dentro de System.Core.dll.

3.2.1.3 Consultas LINQ con Objetos

Al utilizar LINQ para consultar colecciones IEnumerable no genéricas, como ArrayList, debe declarar explícitamente el tipo de la variable para reflejar el tipo específico de los objetos de la colección. Por ejemplo, si tiene una estructura ArrayList de objetos Student, su cláusula From (Cláusula, Visual Basic) debería ser similar a la siguiente:

```
Dim query = From student As Student In arrList
```

Al especificar el tipo de la variable, convierte cada elemento de ArrayList en Student. Utilizar una variable con un tipo declarado explícitamente en una expresión de consulta es equivalente a llamar al método Cast<(Of <(TResult>)>). Cast<(Of <(TResult>)>) inicia una excepción si no se puede realizar la conversión de tipos especificada. Cast<(Of <(TResult>)>) y OfType<(Of <(TResult>)>) son los dos métodos de operador de consulta estándar que actúan en tipos IEnumerable no genéricos

Ejemplo

En el ejemplo siguiente se muestra una consulta simple sobre ArrayList. Observe que en este ejemplo se utilizan inicializadores de objeto cuando el código llama al método Add, pero no es obligatorio.

```
Imports System.Collections
Imports System.Linq

Module Module1

    Public Class Student
        Public FirstName As String
        Public LastName As String
        Public Scores As Integer()
    End Class
    Sub Main()

Dim student1 As New Student With {.FirstName = "Svetlana", _
                                  .LastName = "Omelchenko", _
                                  .Scores = New Integer() {98, 92, 81, 60}}
Dim student2 As New Student With {.FirstName = "Claire", _
                                 .LastName = "O'Donnell", _
                                 .Scores = New Integer() {75, 84, 91, 39}}
Dim student3 As New Student With {.FirstName = "Cesar", _
                                 .LastName = "Garcia", _
```

```
                    .Scores = New Integer() {97, 89, 85, 82}}
Dim student4 As New Student With {.FirstName = "Sven", _
                    .LastName = "Mortensen", _
                    .Scores = New Integer() {88, 94, 65, 91}}
Dim arrList As New ArrayList()
arrList.Add(student1)
arrList.Add(student2)
arrList.Add(student3)
arrList.Add(student4)

' Use an explicit type for non-generic collections
Dim query = From student As Student In arrList _
                Where student.Scores(0) > 95 _
                 Select student

For Each student As Student In query
Console.WriteLine(student.LastName & ": " & student.Scores(0))
            Next
             ' Keep the console window open in debug mode.
            Console.WriteLine("Press any key to exit.")
             Console.ReadKey()
    End Sub
    End Module
```

3.2.2 Convirtiendo SQL a LINQ

3.2.2.1 Conceptos básicos

El proveedor LINQ to SQL permite consultar y modificar los datos de una base de datos de SQL Server. De esta forma, es fácil asignar el modelo de objetos de una aplicación a las tablas y los objetos de una base de datos.

3.2.2.2 Sentencias SQL en LINQ

Las sentencias existentes en LINQ similares al formato del lenguaje de consultas Transac-SQL son:

- Where
- Order by
- Select
- From

Para mayor información mirar el capítulo 2 sección 2.2.3

3.3 LINQ con XML

3.3.1 Introduciendo LINQ a XML

Visual Basic proporciona compatibilidad para LINQ to XML a través de literales XML y propiedades de eje XML. Esto permite utilizar una sintaxis familiar y adecuada para trabajar con XML en el código de Visual Basic. Los literales XML permiten incluir directamente XML en el código. Las propiedades de eje XML

permiten tener acceso a los nodos secundarios, los nodos descendientes y los atributos de un literal XML.

3.3.4 Consultas LINQ a XML

Primero antes de todo tenemos q importar las librerías que nos permitirán trabajar con LINQ:

```
using System.Linq;
using System.Xml.Linq;
```

La Clase XElement está al final de la jerarquía de Linq to XMl, es la clase fundamental del trabajo con XML, ya que los árboles XML están constituidos por elementos de este tipo, además para crear un archivo XML, tenemos que representarlo primeramente al momento de codificar, para ello usamos la clase XDocument con que trabajaremos más adelante para realizar consultas y otras operaciones.

Ahora vamos a ver como se crea un árbol XML mediante LINQ:

```
1:  XDocument documento;
2:  string xmlPath = Server.MapPath("Personas.xml");
3:  documento = new XDocument(
4:  new XDeclaration("1.0", "utf-8", "yes"),
5:          new XElement("Personas",
6:          new XElement("Persona",
7:                  new XAttribute("ID", 0),
```

```
8:              new XElement("Nombre", txtNombre.Text),
9:              new XElement("Apellidos", txtApellidos.Text),
10:                     new XElement("Edad", txtEdad.Text))));
11:     documento.Save(xmlPath);
```

Lo que hemos hecho es crear un objeto que represente al XML y lo hemos llamado documento, entonces lo instanciamos y podemos ver que agregamos una declaración, donde contiene la versión xml, la codificación del documento (Encoding) y la cadena que especifica si es independiente de las entidades externas (standalone); luego de ello creamos la raíz del árbol xml dando el nombre (“Personas”) y por consiguiente especificamos la colección de parámetros que contendrá, claro está con sus respectivos valores; por últimos guardamos el documento en una dirección dada.

CAPÍTULO IV

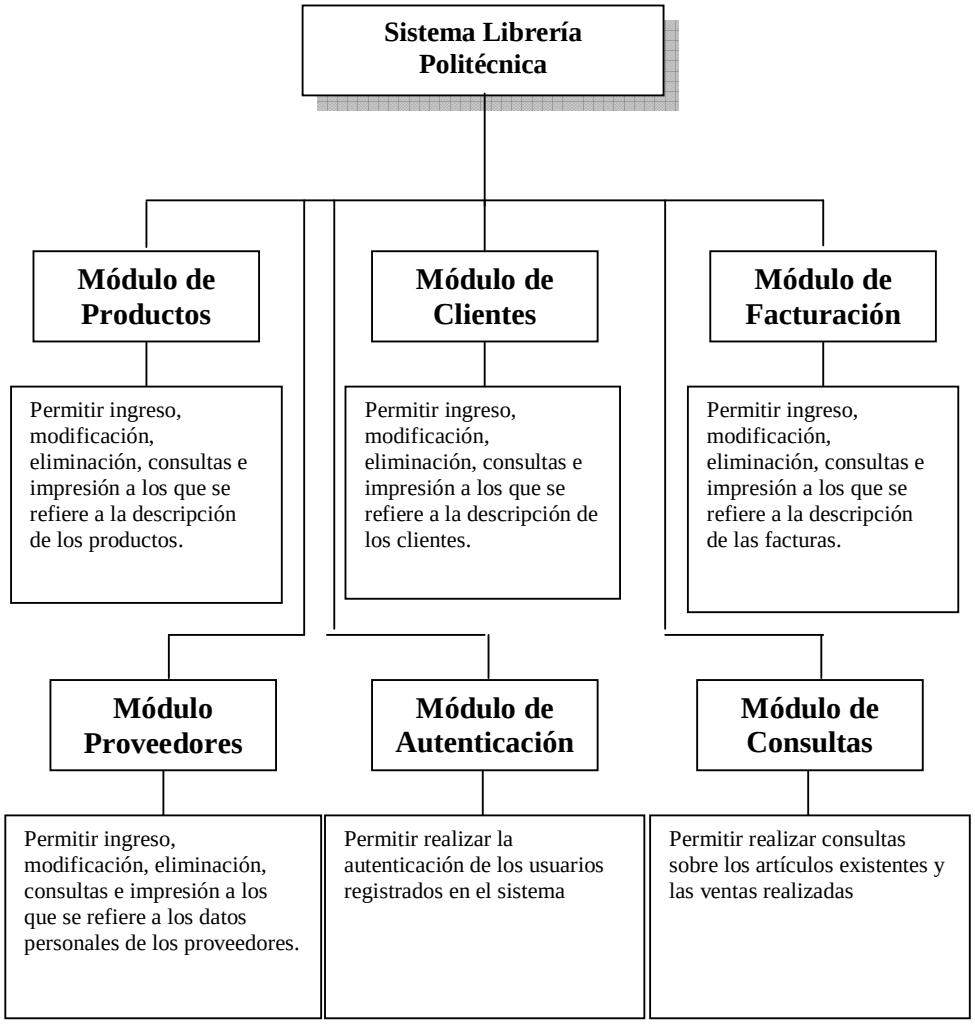
SISTEMA DE INVENTARIO Y FACTURACIÓN DE LA LIBRERÍA ESPOCH

4.1. Planificación y Especificación de Requisitos

La planificación y especificación se lo realiza con el fin de determinar las necesidades del usuario, en esta fase básicamente se definen como quedarán las interfaces y se establece de manera concreta hasta que punto se desarrollará la aplicación (límites y alcances).

A continuación se muestra los módulos del sistema de inventario y facturación de la Librería politécnica (ver figura 4.1).

Figura 4.1: Módulos del SIFLE



Para mayor información ver anexo 2 Documentación del sistema de inventario y facturación de la Librería Politécnica pagina 11 (sección Especificación de Requerimientos de usuario).

4.1.1. Factibilidad

La factibilidad del proyecto se la realiza para determinar si existe el suficiente hardware, software además de si existen los recursos económicos para la realización del sistema de

inventario y facturación de la Librería politécnica, considerando las bases legales para la realización del mismo.

Luego de la investigación se concluyó que era factible tanto económica, operativa y legal la realización del sistema SIFLE, para mayor información consultar el anexo 2 Documentación del sistema de inventario y facturación de la Librería Politécnica pagina 79 (sección 1.9.2 Factibilidad).

4.2. Diseño

En la fase de diseño se desarrollarán los casos de uso de la aplicación, los diagramas de secuencia, colaboración, estado y el diseño de la base de datos.

4.2.1 Diagramas de casos de uso

A continuación se presentará el caso de uso de registrar factura (ver figura 4.2). Para mayor información acerca de los casos de uso diagramas de secuencia, diagramas de colaboración y de estado ver anexo 2 Documentación del sistema de inventario y facturación de la Librería Politécnica, fase de construcción de Bajo nivel página 2 (sección 1. Definir los casos de uso reales).

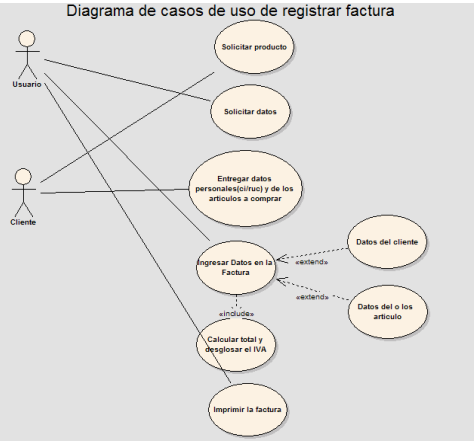


Figura 4.2: Caso de uso registrar factura

4.2.2. Esquema de Base de Datos

En este apartado se bosqueja como quedará la base de datos para almacenar la información, además de poder tener un correcto nivel de normalización.

El esquema de base de datos de la Librería Politécnica es el siguiente (Ver figura 4.3):

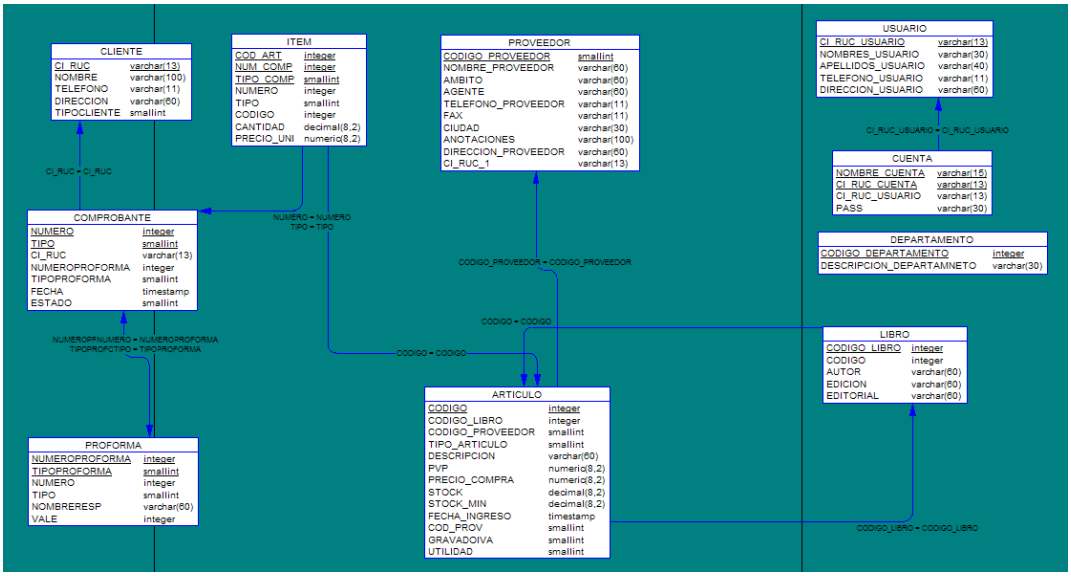


Figura 4.3: Esquema de base de datos

El esquema para el sistema se lo extrajo luego del tercer nivel de normalización, en este esquema se explica como se distribuye la solución en lo concerniente al almacenamiento de la información.

4.2.3. Diagrama del modelo físico y la arquitectura del sistema

El diagrama físico y lógico se lo realiza para ver como quedará implantada la solución, es decir como se realizarán las operaciones del sistema y como se emplearán los recursos hardware y software que se tienen.

Para mayor información ver anexo 2 Documentación del sistema de inventario y facturación de la Librería Politécnica, fase de construcción de bajo nivel página 16 (sección 6. Refinar el modelo físico y la arquitectura del sistema).

4.3. Desarrollo

El desarrollo de la aplicación es la fase donde se implementaron los requerimientos citados en la especificación de requerimientos (Ver anezo 2 “Documentación del sistema de inventario y facturación de la librería Politécnica”, página 11 (sección Especificación de Requerimientos de usuario), además de la construcción de los dos sistemas el primero sin el uso de LINQ, y el segundo con la utilización de LINQ.

CAPÍTULO V

ESTUDIO DE RESULTADOS

En muchos estudios, incluidos la mayoría de los desarrollos de software, es necesario comparar ciertas características en dos o más herramientas de programación. Tal sería el caso, por ejemplo, si pensamos que un proceso nuevo puede tener un porcentaje de mejoría mayor que otro estándar, o cuando nos planteamos si utilizar una tecnología mejora la productividad al momento de desarrollar un sistema.

Para este fin, se ha planteado el siguiente escenario:

Tenemos 2 aplicaciones, **la primera** que fue desarrollada de una manera tradicional creando clases, enlazándolas mediante referencias y creando métodos que interactúen con esas clases, sin olvidarnos de la capa de acceso a datos, para la comunicación con el servidor de Base de Datos que para los dos casos es SQL Server 2000, y **el segundo** en el que tenemos una aplicación creada con la

tecnología LINQ, la misma que posee una capa de reglas de negocio con las clases que nos crea automáticamente LINQ, los métodos de extensión que necesitamos para cada clases y los métodos para interactuar con cada una de las clases.

Para ambos casos tenemos un conjunto de catálogos que corresponden a los métodos que interactúan con las clases, la diferencia está en este punto donde la forma para obtener nuestra información cambia completamente la una de la otra, es aquí donde encontramos dos conjuntos de métodos y funciones creadas para cada una de las aplicaciones estos métodos necesitan una cantidad de Líneas de Código y Cantidad de tiempo, que son las 2 de las 3 variables utilizadas para la comparación.

Para la demostración de la hipótesis se utilizará la distribución normal Z junto con los 2 conjuntos de resultados.

5.1. Modificación de la capa de reglas de Negocio y aplicando LINQ.

El sistema SIFLE que fue construido mediante los métodos tradicionales de creación de clases, con referencias entre ellos, y métodos para interactuar con las clases, posee 4 capas (Ver figura 5.4):

- ✓ Presentación
- ✓ Reglas de Negocio
- ✓ Acceso a Datos y
- ✓ Base de Datos

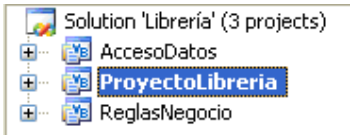


Figura 5.4: Arquitectura de SIFLE Método Tradicional

Existen cambios al realizar la aplicación con la tecnología LINQ, desde su arquitectura como podemos observar la eliminación de las Capas de Acceso a Datos, mientras que la capa de Reglas de Negocio se le ha cambiado de nombre a rLINQ, ya que dentro de esta capa se encuentran unidas las clases creadas por LINQ y los métodos para la recuperación de la información:

Las siguientes capas son (Ver figura 5.5):

- ✓ Presentación
- ✓ Reglas de Negocio con LINQ
- ✓ Base de Datos

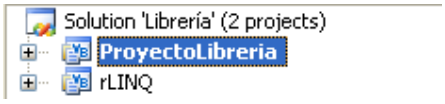


Figura 5.5: Arquitectura de SIFLE con LINQ

5.2 Establecimiento de variables.

Para realizar la comparación de resultados entre el primer método de programación (Sin LINQ) y el segundo (Con LINQ) se estableció 3 variables con la siguiente ponderación (Ver tabla 5.I):

Tabla 5.I: Ponderación de Variables y pesos

Variable	**Peso**
LDC total del programa	10%
Tiempo	40%
LDC de código al programar	50%
	100%

Definición de las variables

A continuación se definen las variables y como se las utilizarán en la comparación:

- ✓ **Total de líneas de código fuente**

  La definición de línea de código, aunque es básica para muchas métricas del software, es ambigua. El significado de línea de código varía de un lenguaje a otro, pero también dentro de un mismo lenguaje de programación.

  En el lenguaje de programación Visual Basic, por ejemplo, una línea de código puede ser:

1.- Una instrucción acabada en un salto de línea o

2.- Cualquier línea del programa que acabe en un salto de línea (comentarios incluidos).

Por ejemplo:

1.- Dim variable as integer

2.- Dim variable as integer 'Este es un comentario

Para contar el total de líneas de código de una aplicación se utilizará la herramienta "Practiline Source Code Line Counter".

- ✓ **Tiempo**

El tiempo es la magnitud física que mide la duración o separación de las cosas sujetas a cambio, de los sistemas sujetos a observación, esto es, el período que transcurre entre el estado del sistema cuando éste aparentaba un estado X y el instante en el que X registra una variación perceptible para un observador. Es la magnitud que permite ordenar los sucesos en secuencias, estableciendo un pasado, un presente y un futuro[4]

- ✓ **Líneas de Código al Programar**

Esta variable lo hemos tomado, ya que al momento de programar, podemos optar por ocupar tecnologías o herramientas de programación que nos generan código, que probablemente al finalizar la aplicación esta

[4] WIKIPEDIA®(2008)."Tiempo". http://es.wikipedia.org/wiki/Tiempo . [Consulta 12-09-2008]

puede tener más líneas de código que una que no haya ocupado estas herramientas o tecnologías.

Tomando en cuenta esto se contarán las líneas de código que después de haber ocupado estas herramientas o tecnologías se necesiten para crear métodos o funciones complementarias.

De la misma manera para poder comparar esta variable se ha sacado una muestra de métodos utilizados en la aplicación de un total de 54 métodos aplicando la fórmula de muestreo queda:

Formula 3. Tamaño de la muestra

$$n = \frac{N\sigma^2 Z^2}{(N-1)E^2 + \sigma^2 Z^2}$$

$n =$ Tamaño de la muestra

$N =$ Universo de la población

$\sigma^2 =$ Varianza $(0.5)^2$

$Z =$ Nivel de Confianza

$E =$ Límite aceptable del Error maestral

Para el nivel de confianza (Z) utilizamos la distribución Gussiana.

Tabla 5.II: Coeficientes de confianza.

Coeficiente de Confianza	50%	68,27%	90%	95%	95,45%	99%	99,37%
Z	0,647	1,00	1,645	1,96	2,00	2,58	3,00

Datos con los se cuenta:

$n = X$

N = 54

$\sigma^2 = (0.5)^2$

Z = 1,96

E = 0,05

Aplicando la Fórmula 3:

$$n = \frac{(54)(0,5)^2(1,96)^2}{(54-1)(0,05)^2 + (0,5)^2(1,96)^2}$$

$$n = \frac{51,8616}{1,0929}$$

$n = 47,45 \cong 48$

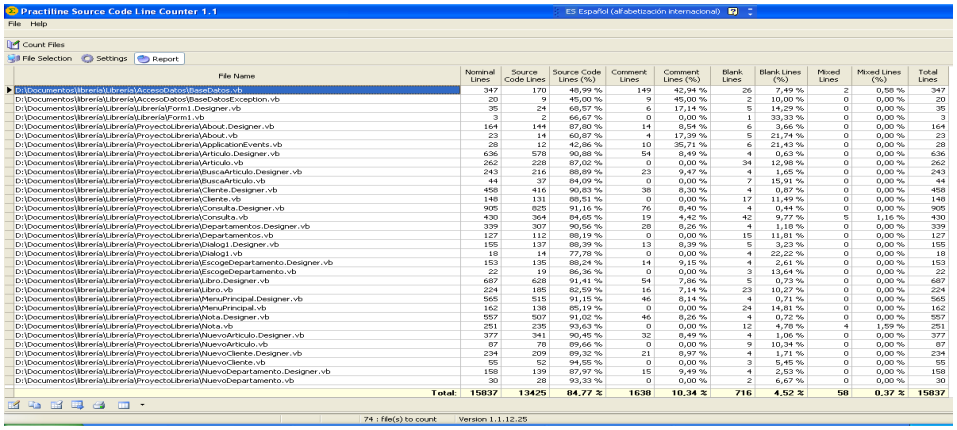
$n = 48$

Por lo tanto la muestra con los cuales se ha trabajado en el apartado 5.3 son 48 métodos, los mismos que se tabularán con la herramienta SPSS10.0 para la generación del reporte de media, mediana, moda, desviación estándar, error típico y gráfico correspondiente.

5.3. Generación de resultados de las aplicaciones.

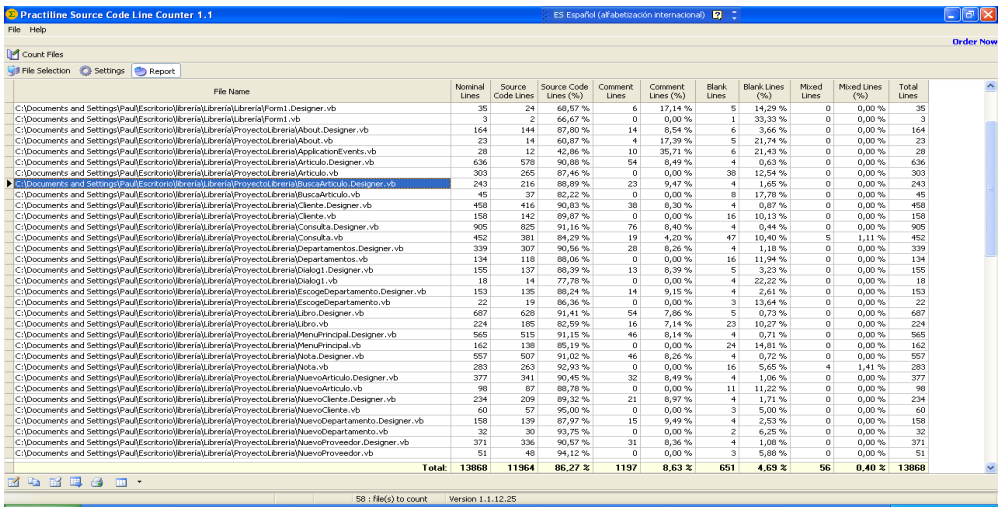
Variable "Total de Líneas de Código"

Los resultados arrojados por la aplicación "Practiline Source Code Line Counter" son los siguientes (Ver figura 5.6 y 5.7):



File Name	Nominal Lines	Source Code Lines	Source Code Lines (%)	Comment Lines	Comment Lines (%)	Blank Lines	Blank Lines (%)	Mixed Lines	Mixed Lines (%)	Total Lines
D:\Documentos\libreria\Libreria\AccesoDatos\BaseDatos.vb	347	170	48,99 %	149	42,94 %	26	7,49 %	2	0,58 %	347
D:\Documentos\libreria\Libreria\AccesoDatos\BaseDatosException.vb	20	9	45,00 %	9	45,00 %	2	10,00 %	0	0,00 %	20
D:\Documentos\libreria\Libreria\Libreria\Form1.Designer.vb	35	24	68,57 %	6	17,14 %	5	14,29 %	0	0,00 %	35
D:\Documentos\libreria\Libreria\Libreria\Form1.vb	3	2	66,67 %	0	0,00 %	1	33,33 %	0	0,00 %	3
D:\Documentos\libreria\Libreria\ProyectoLibreria\About.Designer.vb	164	144	87,80 %	14	8,54 %	6	3,66 %	0	0,00 %	164
D:\Documentos\libreria\Libreria\ProyectoLibreria\About.vb	23	14	60,87 %	4	17,39 %	5	21,74 %	0	0,00 %	23
D:\Documentos\libreria\Libreria\ProyectoLibreria\ApplicationEvents.vb	28	12	42,86 %	10	35,71 %	6	21,43 %	0	0,00 %	28
D:\Documentos\libreria\Libreria\ProyectoLibreria\Articulo.Designer.vb	636	578	90,88 %	54	8,49 %	4	0,63 %	0	0,00 %	636
D:\Documentos\libreria\Libreria\ProyectoLibreria\Articulo.vb	262	228	87,02 %	0	0,00 %	34	12,98 %	0	0,00 %	262
D:\Documentos\libreria\Libreria\ProyectoLibreria\BuscaArticulo.Designer.vb	243	216	88,89 %	23	9,47 %	4	1,65 %	0	0,00 %	243
D:\Documentos\libreria\Libreria\ProyectoLibreria\BuscaArticulo.vb	44	37	84,09 %	0	0,00 %	7	15,91 %	0	0,00 %	44
D:\Documentos\libreria\Libreria\ProyectoLibreria\Cliente.Designer.vb	458	416	90,83 %	38	8,30 %	4	0,87 %	0	0,00 %	458
D:\Documentos\libreria\Libreria\ProyectoLibreria\Cliente.vb	148	131	88,51 %	0	0,00 %	17	11,49 %	0	0,00 %	148
D:\Documentos\libreria\Libreria\ProyectoLibreria\Consulta.Designer.vb	905	825	91,16 %	76	8,40 %	4	0,44 %	0	0,00 %	905
D:\Documentos\libreria\Libreria\ProyectoLibreria\Consulta.vb	430	364	84,65 %	19	4,42 %	42	9,77 %	5	1,16 %	430
D:\Documentos\libreria\Libreria\ProyectoLibreria\Departamentos.Designer.vb	339	307	90,56 %	28	8,26 %	4	1,18 %	0	0,00 %	339
D:\Documentos\libreria\Libreria\ProyectoLibreria\Departamentos.vb	127	112	88,19 %	0	0,00 %	15	11,81 %	0	0,00 %	127
D:\Documentos\libreria\Libreria\ProyectoLibreria\Dialog1.Designer.vb	155	137	88,39 %	13	8,39 %	5	3,23 %	0	0,00 %	155
D:\Documentos\libreria\Libreria\ProyectoLibreria\Dialog1.vb	18	14	77,78 %	0	0,00 %	4	22,22 %	0	0,00 %	18
D:\Documentos\libreria\Libreria\ProyectoLibreria\EscogeDepartamento.Designer.vb	153	135	88,24 %	14	9,15 %	4	2,61 %	0	0,00 %	153
D:\Documentos\libreria\Libreria\ProyectoLibreria\EscogeDepartamento.vb	22	19	86,36 %	0	0,00 %	3	13,64 %	0	0,00 %	22
D:\Documentos\libreria\Libreria\ProyectoLibreria\Libro.Designer.vb	687	628	91,41 %	54	7,86 %	5	0,73 %	0	0,00 %	687
D:\Documentos\libreria\Libreria\ProyectoLibreria\Libro.vb	224	185	82,59 %	16	7,14 %	23	10,27 %	0	0,00 %	224
D:\Documentos\libreria\Libreria\ProyectoLibreria\MenuPrincipal.Designer.vb	565	515	91,15 %	46	8,14 %	4	0,71 %	0	0,00 %	565
D:\Documentos\libreria\Libreria\ProyectoLibreria\MenuPrincipal.vb	162	138	85,19 %	0	0,00 %	24	14,81 %	0	0,00 %	162
D:\Documentos\libreria\Libreria\ProyectoLibreria\Nota.Designer.vb	557	507	91,02 %	46	8,26 %	4	0,72 %	0	0,00 %	557
D:\Documentos\libreria\Libreria\ProyectoLibreria\Nota.vb	251	235	93,63 %	0	0,00 %	12	4,78 %	4	1,59 %	251
D:\Documentos\libreria\Libreria\ProyectoLibreria\NuevoArticulo.Designer.vb	377	341	90,45 %	32	8,49 %	4	1,06 %	0	0,00 %	377
D:\Documentos\libreria\Libreria\ProyectoLibreria\NuevoArticulo.vb	87	78	89,66 %	0	0,00 %	9	10,34 %	0	0,00 %	87
D:\Documentos\libreria\Libreria\ProyectoLibreria\NuevoCliente.Designer.vb	234	209	89,32 %	21	8,97 %	4	1,71 %	0	0,00 %	234
D:\Documentos\libreria\Libreria\ProyectoLibreria\NuevoCliente.vb	55	52	94,55 %	0	0,00 %	3	5,45 %	0	0,00 %	55
D:\Documentos\libreria\Libreria\ProyectoLibreria\NuevoDepartamento.Designer.vb	158	139	87,97 %	15	9,49 %	4	2,53 %	0	0,00 %	158
D:\Documentos\libreria\Libreria\ProyectoLibreria\NuevoDepartamento.vb	30	28	93,33 %	0	0,00 %	2	6,67 %	0	0,00 %	30
Total:	**15837**	**13425**	**84,77 %**	**1638**	**10,34 %**	**716**	**4,52 %**	**58**	**0,37 %**	**15837**

Figura 5.6: Resultados sin LINQ Líneas de Código



File Name	Nominal Lines	Source Code Lines	Source Code Lines (%)	Comment Lines	Comment Lines (%)	Blank Lines	Blank Lines (%)	Mixed Lines	Mixed Lines (%)	Total Lines
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\Libreria\Form1.Designer.vb	35	24	68,57 %	6	17,14 %	5	14,29 %	0	0,00 %	35
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\Libreria\Form1.vb	3	2	66,67 %	0	0,00 %	1	33,33 %	0	0,00 %	3
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\About.Designer.vb	164	144	87,80 %	14	8,54 %	6	3,66 %	0	0,00 %	164
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\About.vb	23	14	60,87 %	4	17,39 %	5	21,74 %	0	0,00 %	23
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\ApplicationEvents.vb	28	12	42,86 %	10	35,71 %	6	21,43 %	0	0,00 %	28
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Articulo.Designer.vb	636	578	90,88 %	54	8,49 %	4	0,63 %	0	0,00 %	636
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Articulo.vb	303	265	87,46 %	0	0,00 %	38	12,54 %	0	0,00 %	303
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\BuscaArticulo.Designer.vb	243	216	88,89 %	23	9,47 %	4	1,65 %	0	0,00 %	243
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\BuscaArticulo.vb	45	37	82,22 %	0	0,00 %	8	17,78 %	0	0,00 %	45
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Cliente.Designer.vb	458	416	90,83 %	38	8,30 %	4	0,87 %	0	0,00 %	458
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Cliente.vb	158	142	89,87 %	0	0,00 %	16	10,13 %	0	0,00 %	158
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Consulta.Designer.vb	905	825	91,16 %	76	8,40 %	4	0,44 %	0	0,00 %	905
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Consulta.vb	452	381	84,29 %	19	4,20 %	47	10,40 %	5	1,11 %	452
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Departamentos.Designer.vb	339	307	90,56 %	28	8,26 %	4	1,18 %	0	0,00 %	339
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Departamentos.vb	134	118	88,06 %	0	0,00 %	16	11,94 %	0	0,00 %	134
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Dialog1.Designer.vb	155	137	88,39 %	13	8,39 %	5	3,23 %	0	0,00 %	155
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Dialog1.vb	18	14	77,78 %	0	0,00 %	4	22,22 %	0	0,00 %	18
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\EscogeDepartamento.Designer.vb	153	135	88,24 %	14	9,15 %	4	2,61 %	0	0,00 %	153
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\EscogeDepartamento.vb	22	19	86,36 %	0	0,00 %	3	13,64 %	0	0,00 %	22
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Libro.Designer.vb	687	628	91,41 %	54	7,86 %	5	0,73 %	0	0,00 %	687
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Libro.vb	224	185	82,59 %	16	7,14 %	23	10,27 %	0	0,00 %	224
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\MenuPrincipal.Designer.vb	565	515	91,15 %	46	8,14 %	4	0,71 %	0	0,00 %	565
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\MenuPrincipal.vb	162	138	85,19 %	0	0,00 %	24	14,81 %	0	0,00 %	162
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Nota.Designer.vb	557	507	91,02 %	46	8,26 %	4	0,72 %	0	0,00 %	557
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\Nota.vb	283	263	92,93 %	0	0,00 %	16	5,65 %	4	1,41 %	283
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\NuevoArticulo.Designer.vb	377	341	90,45 %	32	8,49 %	4	1,06 %	0	0,00 %	377
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\NuevoArticulo.vb	98	87	88,78 %	0	0,00 %	11	11,22 %	0	0,00 %	98
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\NuevoCliente.Designer.vb	234	209	89,32 %	21	8,97 %	4	1,71 %	0	0,00 %	234
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\NuevoCliente.vb	60	57	95,00 %	0	0,00 %	3	5,00 %	0	0,00 %	60
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\NuevoDepartamento.Designer.vb	158	139	87,97 %	15	9,49 %	4	2,53 %	0	0,00 %	158
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\NuevoDepartamento.vb	32	30	93,75 %	0	0,00 %	2	6,25 %	0	0,00 %	32
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\NuevoProveedor.Designer.vb	371	336	90,57 %	31	8,36 %	4	1,08 %	0	0,00 %	371
C:\Documents and Settings\Paul\Escritorio\libreria\Libreria\ProyectoLibreria\NuevoProveedor.vb	51	48	94,12 %	0	0,00 %	3	5,88 %	0	0,00 %	51
Total:	**13868**	**11964**	**86,27 %**	**1197**	**8,63 %**	**651**	**4,69 %**	**56**	**0,40 %**	**13868**

Figura 5.7: Resultados con LINQ Líneas de Código

La siguiente tabla muestra los resultados de la aplicación "Practiline Source Code Line Counter", de la cual solo se va a tomar los valores marcados con "útil", ya que son los valores que resaltan para comprobar nuestro objetivo (Ver tabla 5.III).

Tabla 5.III: Generación de resultados de la aplicación "Practiline Source Code Line Counter"

Líneas	**Valor sin LINQ**	**Valor con LINQ**
Líneas en blanco	716	651
Líneas comentadas	1638	1197
Líneas de código fuente	13425 (útil)	11964 (útil)
Total	15837	13868

De los valores de la tabla 5.III se tomarán solo los valores marcados como útil.

Variable "Tiempo"

Para la variable Tiempo los valores serán reales, es decir el tiempo que nos hemos demorado en programar el sistema de inventario y facturación de la librería ESPOCH sin y con LINQ:

Ts = 4 semanas (Tiempo sin LINQ)

Tc = 2 semanas (Tiempo con LINQ)

Variable "Líneas de Código al Programar"

Para obtener las líneas de código al programar se utilizará el resultado obtenido en la definición de la variable "Líneas de código al programar" que es 48, donde se ha obtenido los siguientes resultados:

La siguiente tabla muestra la cantidad de líneas de código al momento de programar con y sin LINQ (Ver tabla 5.IV), en el catálogo artículos.

Tabla 5.IV: Catálogo Artículos

Método	LDC (con LINQ)	LDC (sin LINQ)
ObtenerArticulos	12	31
ObtenerArticulo	14	26
NuevoArticulo	10	28
ModificarArticulo	9	29
EliminarArticulo	10	20
ObtenerUltimoArticulo	7	25
Total	88	159

En el total de líneas de la tabla 5.IV se puede visualizar que el número de líneas escritas con LINQ es menor, con relación al redactado sin LINQ.

La siguiente tabla muestra la cantidad de líneas de código al momento de programar con y sin LINQ (Ver tabla 5.V), en el catálogo clientes.

Tabla 5.V: Catálogo clientes

Método	LDC (con LINQ)	LDC (sin LINQ)
ObtenerClientes	11	32
ObtenerCliente	11	28
NuevoCliente	7	21
ModificarCliente	7	21
EliminarCliente	7	17
Total	54	119

En el total de líneas de la tabla 5.V, se puede visualizar que el número de líneas escritas con LINQ es menor, con relación al redactado sin LINQ.

La siguiente tabla muestra la cantidad de líneas de código al momento de programar con y sin LINQ (Ver tabla 5.VI), en el catálogo comprobantes.

Tabla 5.VI: Catálogo comprobantes

Método	**LDC (con LINQ)**	**LDC (sin LINQ)**
AnularComprobante	10	24
Total	10	24

En el total de líneas de la tabla 5.VI, se puede visualizar que el número de líneas escritas con LINQ es menor, con relación al redactado sin LINQ.

La siguiente tabla muestra la cantidad de líneas de código al momento de programar con y sin LINQ (Ver tabla 5.VII), en el catálogo departamentos.

Tabla 5.VII: Catálogo departamentos

Método	**LDC (con LINQ)**	**LDC (sin LINQ)**
ObtenerDepartamentos	10	30
ObtenerDepartamentos	12	29
ObtenerDepartamento	10	25
NuevoDepartamento	10	20
ModificarDepartamento	10	21

EliminarDepartamento	10	20
ObtenerUltimoDepartamento	9	25
Total	71	170

En el total de líneas de la tabla 5.VII, se puede visualizar que el número de líneas escritas con LINQ es menor, con relación al redactado sin LINQ.

La siguiente tabla muestra la cantidad de líneas de código al momento de programar con y sin LINQ (Ver tabla 5.VIII), en el catálogo fac_not.

Tabla 5.VIII: Catálogo fac_not

Método	**LDC (con LINQ)**	**LDC (sin LINQ)**
ObtenerComprobante	16	29
ObtenerFac_NotDelDia	17	29
ObtenerItemsFac_Not	19	30
ConfirmarFac_Not	10	23
IniciarFac_Not	5	14
CancelarFac_NotsPendientes	8	8
CancelarFac_Not	5	13
ObtenerFAc_NotsPendientes	20	28
ObtenerUltimaFactura	7	26
ObtenerUltimaNota	7	25
Total	114	225

En el total de líneas de la tabla 5.VIII, se puede visualizar que el número de líneas escritas con LINQ es menor, con relación al redactado sin LINQ.

La siguiente tabla muestra la cantidad de líneas de código al momento de programar con y sin LINQ (Ver tabla 5.IX), en el catálogo proformas

Tabla 5.IX: Catálogo proformas

Método	**LDC (con LINQ)**	**LDC (sin LINQ)**
ObtenerProformasDelDia	17	29
ObtenerProforma	15	28
ObtenerItemsProforma	17	28
ConfirmarProforma	10	23
IniciarProforma	5	17
UpdateProforma	5	17
CancelarProfromasPendientes	8	8
CancelarProforma	9	21
ObtenerProformasPendientes	18	32
ObtenerUltimaProforma	7	25
Total	111	228

En el total de líneas de la tabla 5.IX, se puede visualizar que el número de líneas escritas con LINQ es menor, con relación al redactado sin LINQ.

La siguiente tabla muestra la cantidad de líneas de código al momento de programar con y sin LINQ (Ver tabla 5.X), en el catálogo proveedores

Tabla 5.X: Catálogo proveedores

Método	LDC (con LINQ)	LDC (sin LINQ)
ObtenerProveedores	10	34
ObtenerProveedor	11	32
ObtenerProveedores	10	35
NuevoProveedor	7	28
ModificarProveedor	10	30
ElimnarProveedor	10	20
ObtenerUltimoProveedor	7	25
Total	65	204

En el total de líneas de la tabla 5.X, se puede visualizar que el número de líneas escritas con LINQ es menor, con relación al redactado sin LINQ.

La siguiente tabla muestra la cantidad de líneas de código al momento de programar con y sin LINQ (Ver tabla 5.XI), en el catálogo usuarios

Tabla 5.XI: Catálogo usuarios

Método	LDC (con LINQ)	LDC (sin LINQ)
ObtenerUsuario	10	27
Total	10	27

En el total de líneas de la tabla 5.XI, se puede visualizar que el número de líneas escritas con LINQ es menor, con relación al redactado sin LINQ.

Con la muestra ingresada en el programa SPSS100 se obtuvieron los siguientes resultados:

Sin LINQ

Media = 24,1875

Variabilidad de la desviación = 6,7718

Varianza = 45,8577

Moda = 25,00

Para mayor información consultar anexo 1, tabla generada por el software spss con los datos las líneas de código al programar con y sin LINQ.

Con LINQ

Media = 10,5417

Variabilidad de la desviación = 3,7754

Varianza = 14,2535

Moda = 10,00

Para mayor información consultar anexo 1, tabla generada por el software SPSS con los datos las líneas de código al programar con y sin LINQ.

5.4. Estudio comparativo de los resultados de ambas aplicaciones.

Con los resultados obtenidos en el apartado anterior se ha calculado el porcentaje de mejora de la aplicación desarrollada con LINQ respecto de la que no se utilizó, aplicando la siguiente fórmula de regla de tres:

$$\begin{array}{ccc} A & \longrightarrow & B \\ X & \longrightarrow & Y \end{array} \qquad Y = \frac{B \cdot X}{A}$$

Donde se obtuvieron los siguientes resultados:

Variable "Total de Líneas de Código"

Sin LINQ = 13425 LDC Con LINQ = 11964 LDC

13425 — 100%

11964 — Y

$Y = 89{,}12\,\%$

$Mejora = 100\% - 89{,}12\%$

$Mejora = 10{,}88\%$

Con la aplicación de la fórmula anterior se demuestra que las líneas de código generadas al final de la aplicación se redujeron. Es decir con la utilización de LINQ se mejora en un 10,88%, lo que significa que se escribieron una menor cantidad de líneas de código.

Variable "Tiempo"

Sin LINQ = 4 (semanas) Con LINQ = 2 (semanas)

4 — 100%

2 — Y

$Y = 50\%$

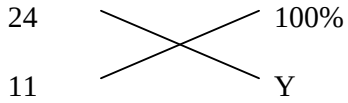
$$Mejora = 100\% - 50\%$$

$$Mejora = 50\%$$

Al programar con LINQ se obtuvo una mejora del 50 % en el tiempo empleado, lo que significa que mediante el empleo de LINQ se desarrollo en menor tiempo la aplicación de inventario y facturación de la Librería Politécnica.

Variable "Líneas de Código al Programar"

Sin LINQ = 24 LDC Con LINQ = 11 LDC



$$Y = 89{,}12$$

$$Mejora = 100\% - 45{,}83\%$$

$$Mejora = 54{,}17\%$$

Mediante la utilización de LINQ se logro una mejora en la productividad del programador del 54,17 % en las líneas de código al programar, lo que significa que un programador puede reducir su trabajo con la utilización de LINQ.

El gráfico siguiente muestra las medias de la utilización de LINQ y el no empleo de LINQ (Ver figura 5.8).

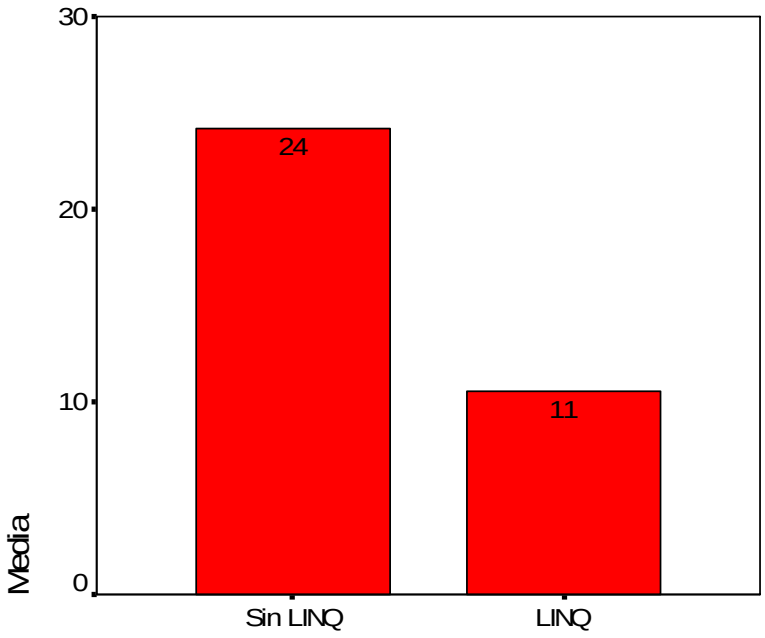


Figura 5.8: Resultados gráficos con LINQ y sin LINQ

La figura 5.8 muestra de manera explicita que mediante la utilización de LINQ en las líneas de código al momento de programar, la media de LINQ es inferior en relación a la de sin LINQ. Por lo que se comprueba la mejora mostrada del 54,17% en la variable de líneas de código al programar.

Resultado final

Tomando como base la tabla "Ponderación de Variables" y los datos de mejora de todas las variables se ha aplicada el siguiente proceso:

- ✓ Variable "Total Líneas de Código"

$$MejoraFinal = (10{,}88\%)(10\%)$$

$$MejoraFinal = 1{,}09\%$$

- ✓ Variable “Tiempo”

$$MejoraFinal = (50\%)(40\%)$$

$$MejoraFinal = 20\%$$

- ✓ Variable “Líneas de Código al Programar”

$$MejoraFinal = (54{,}17\%)(50\%)$$

$$MejoraFinal = 27{,}09\%$$

Luego en la tabla de ponderación se tiene:

Tabla 5.XII: Ponderación de la mejora final

Variable	**Peso**
LCD total del programa	1,09%
Tiempo	20,00%
LCD de código al programar	27,09%
	48,18%

Sumando todos los resultados de las variables ponderadas nos dan un total de **48,18%** de mejora en el desarrollo de un proyecto realizado con LINQ respecto de uno sin la utilización de LINQ.

5.5. Demostración de la hipótesis.

Para la demostración de la hipótesis se ha tomado la variable “Líneas de Código al Programar” junto con la distribución normal Z y la teoría de Hipótesis.

Hipótesis: “La utilización de la tecnología integrada de Consultas, definida como LINQ(Languaje Integrated Query) mejorará la productividad a la hora de crear aplicaciones empresariales de datos habilitados”

$H0 =$ Desarrollar un sistema sin LINQ es igual en líneas de código al programar que desarrollar un sistema con LINQ

$H1 =$ Desarrollar un sistema sin LINQ es diferente en líneas de código al programar que desarrollar un sistema con LINQ

$H0 = \mu SLINQ = uCLINQ$
$H1 = \mu SLINQ <> uCLINQ$

Datos SLINQ

$ns = 48$

$Media = Uxs = 24{,}1875$

Desv. Estand.$= Ss = 6{,}7718$

Datos CLINQ

$nc = 48$

$Media = Uxc = 10{,}5417$

Desv. Estand.$= Sc = 3{,}7754$

Error Estándar= 5%

Cálculo de la Desviación Estándar X1-X2

$$S_{\bar{X}_1-\bar{X}_2} = \sqrt{\frac{Ss^2}{nc} + \frac{Sc^2}{ns}}$$

$$S_{\bar{X}_1-\bar{X}_2} = \sqrt{\frac{(6{,}7718)^2}{48} + \frac{(3{,}7754)^2}{48}}$$

$$S_{\bar{X}_1-\bar{X}_2} = \sqrt{0{,}9554 + 0{,}2970}$$

$$S_{\bar{X}_1-\bar{X}_2} = 1{,}1191$$

Cálculo de Z

$$Z = \frac{\bar{X}_1 - \bar{X}_2}{S_{\bar{X}_1-\bar{X}_2}}$$

$$Z = \frac{24{,}1875 - 10{,}5417}{1{,}1191}$$

$$Z = 12{,}193$$

Utilizando el software DIAGNOS 1.0 se obtuvo el siguiente resultado:

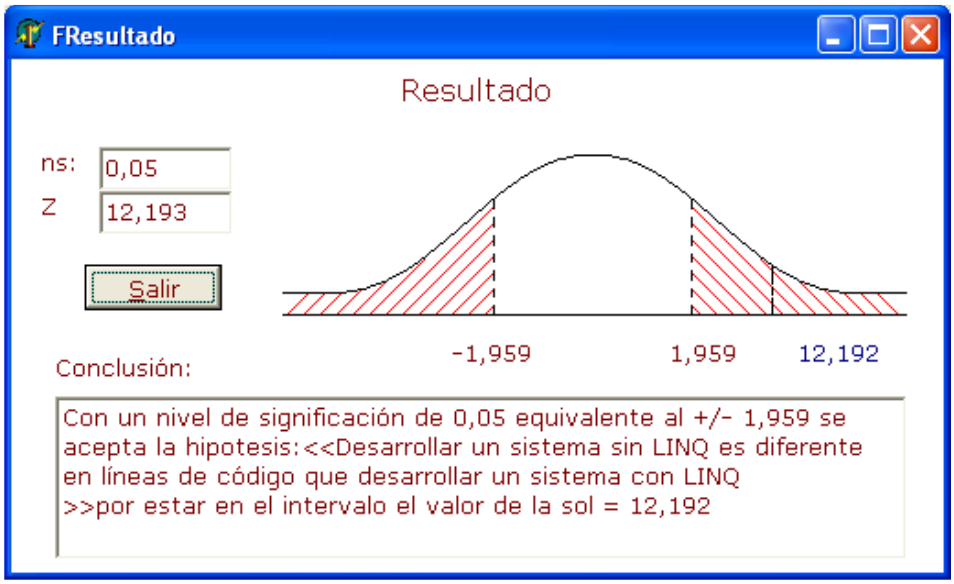


Figura 5.9: Gráfico demostración de la hipótesis

Basándonos en el gráfico de la demostración de la hipótesis representado en la figura 5.9 con un nivel de significación de 0,05 equivalente al +/- 1,959 se acepta la hipótesis alternativa “Desarrollar un sistema sin LINQ es diferente en líneas de código al programar que desarrollar un sistema con LINQ”, por encontrarse el valor de Z en la cola derecha se concluye que:

“Desarrollar un sistema sin LINQ requiere más líneas de código al programar que desarrollar un sistema con LINQ”.

Por tanto si se requiere más líneas de código y el resultado es el mismo entonces:

“Desarrollar un sistema con LINQ es más productivo que el desarrollo sin LINQ.”

Nota: Ambas maneras de programar (con LINQ y sin LINQ) se deben conocer por parte de los desarrolladores para que la comparación sea válida.

CONCLUSIONES

- La realización del estudio comparativo entre la nueva tecnología LINQ y la manera tradicional para el desarrollo de un sistema, permitió determinar en base a algunos parámetros que la tecnología LINQ mejora en un 48,18 %, en el desarrollo de una aplicación. Respecto a la manera tradicional.
- En escenarios similares en los que se necesitan crear métodos en las mismas condiciones y necesitando el mismo resultado, la cantidad de líneas de código al programar se redujo en un 54,17%.
- La utilización de la tecnología LINQ mejoró el tiempo de desarrollo del sistema de inventario y facturación de la librería politécnica en un 50 %.
- Al momento de utilizar la nueva tecnología LINQ se evita que un programador conozca un sin número de lenguajes de consulta para cada repositorio de información. LINQ permite integrar los varios lenguajes de consulta en uno solo.
- El total de líneas de código con las que muchas metodologías se basan para estimar los costos de uno u otro sistema disminuyó en un 10,88%.
- La tecnología LINQ es nueva al momento de realizar esta tesis por lo que tiene todavía muchos detractores así como desarrolladores que no la conocen.

RECOMENDACIONES

- Se debe tener en cuenta que al momento de realizar la comparación hay que definir y calcular correctamente los parámetros para obtener conclusiones válidas.
- Al momento de seleccionar el personal para el desarrollo de una aplicación informática en Visual Studio .Net 2008, con el objetivo de reducir el tiempo se debe asegurar que los mismos conozcan a fondo la nueva tecnología LINQ.
- Se recomienda que la Escuela de Ingeniería en Sistemas incluya en su pensum de estudios la nueva tecnología LINQ, ya que se comprobó que conociendo la misma, los nuevos graduados podrán mejorar su productividad, al momento de desarrollar una aplicación informática.

RESUMEN

El objetivo de realizar un estudio sobre la nueva tecnología LINQ (Language Integrated Query) que ofrece Microsoft en su Framework 3.0, para el desarrollo del sistema de inventario y facturación de la librería politécnica, es determinar la productividad de la utilización de la misma.

Para su realización se utilizó herramientas: Visual Studio 2008, Visual Basic 9.0, LINQ y SqlServer 2000. Los métodos utilizados fueron el científico, deductivo y comparativo además de técnicas estadísticas aplicadas en todo el estudio basado en parámetros.

Se ha llegado a obtener los siguientes resultados: La productividad que se gana al momento de programar utilizando LINQ es de 28,18% en lo referente a las líneas de código, el tiempo utilizando LINQ mejoró en un 20%, comprobándose que la utilización de LINQ mejora en un 48.18 % al momento de desarrollar un sistema informático

Se concluye que LINQ mejoró la productividad al desarrollar el sistema de inventario y facturación para la Librería Politécnica, alcanzándose el objetivo propuesto.

Se recomienda que, al momento de desarrollar un sistema informático, se utilice la nueva tecnología LINQ ya que reduce el tiempo y mejora la productividad de un programador.

SUMMARY

The objective of carrying out a study on the new technology LINQ (Language Integrated Query) which is offered by Microsoft in its Framework 3.0, for the development of an inventory and invoice system of the Politécnica library, is determining the productivity of the use of this.

To carry this out the following tools were used: Visual Studio 2008, Visual Basic 9.0, LINQ and SqlServer 2000, the used methods were the scientific, deductive and the comparative ones as well as the statistical techniques applied in the whole study based on parameters.

The following results were obtained: productivity which is gained at programming using LINQ is 28.18% as to the code lines, the time using LINQ improved by 20%, demonstrating that the LINQ use improves by 48.18% at the moment of developing the inventory and invoice system for the Politécnica library, attaining the proposed objective. It is recommended, at the moment of developing an informatics system, to use the new LINQ technology as it reduces the time and improves programmer productivity.

GLOSARIO

ESPOCH.- Escuela Superior Politécnica de Chimborazo

APPOCH.- Asociación de Profesores Politécnicos de Chimborazo

FEPOCH.- Federación de Estudiantes Politécnicos de Chimborazo

DESITEL.- Departamento de Sistemas y Telemática

SIFLE.- Sistema de Inventario y Facturación de la Librería ESPOCH

LINQ.- Lenguaje Integrado de Consultas (Language Integrated Query)

CLIENTE.- Persona o Institución que recibe y usa el producto.

Estándar.- Norma aprobada y aceptada por una Institución de normalización.

Formularios.- Forma de realizar las interfaces, diferentes pantallas.

Interfaz.- Conexión física y funcional entre dos sistemas independientes,

Misión.- Objetivo de cada empresa o Institución.

Modelo.- Guía que ha sido aprobada y aceptada para seguir en un proceso de ingeniería de software.

Prototipo.- Ejemplar original o primer molde en que se fabrica una figura u otra cosa.

Proyecto.- Conjunto de actividades planificadas, coordinadas y presupuestadas que se emprenden para alcanzar objetivos específicos.

Requisito.- Circunstancia o condición necesaria para satisfacer alguna necesidad.

Servicio.- Producto intangible que es el resultado de realizar por lo menos una actividad en la interfaz.

Sistema.- Conjunto de elementos interrelacionados e interactuantes en uno o mas de los procesos de proporcionan la capacidad de satisfacer las necesidades u objetivo definido.

Usuario.- Persona que usa el sistema para realizar una función específica.

Visión.- Planea a futuro de una empresa o Institución.

Fines.- Término, remate o consumación de una cosa. Objeto o motivo con que se ejecuta una cosa.

Antecedentes.- Que antecede. Acción, dicho o circunstancia anterior que sirve para juzgar hechos posteriores.

Objetivos.- Pertenece o relaciona al objeto en si y no a nuestro modo de pensar o de sentir.

Eslogan.- Fórmula breve y original, utilizada para publicidad, propaganda política, etc.

ANEXOS

Anexo 1

Estadísticas líneas de código con LINQ y sin LINQ

		Sin LINQ	LINQ
N	Validos	48	48
	No encontrados	3	3
Media		24,1875	10,5417
Error estándar		,9774	,5449
Mediana		25,0000	10,0000
Moda		25,00	10,00
Variabilidad de la desviación		6,7718	3,7754
Varianza		45,8577	14,2535
Coeficiente de asimetría		-1,006	,799
Variabilidad del coeficiente de asimetría		,343	,343
Coeficiente de apuntamiento		,853	,148
Variabilidad del coeficiente de apuntamiento		,674	,674
Rango		30,00	15,00
Mínimo		5,00	5,00
Máximo		35,00	20,00

Suma		1161,00	506,00
Percentiles	10	13,9000	6,8000
	20	20,0000	7,0000
	25	20,2500	7,2500
	30	21,0000	8,7000
	40	24,6000	10,0000
	50	25,0000	10,0000
	60	28,0000	10,0000
	70	28,3000	11,0000
	75	29,0000	12,0000
	80	29,2000	14,0000
	90	32,0000	17,0000

ANEXO 2

“DOCUMENTACIÓN DEL SISTEMA DE INVENTARIO Y FACTURACIÓN DE LA LIBRERÍA POLITÉCNICA”

BIBLIOGRAFÍA

- ✓ PIALORSI, Paolo. Introducing Microsoft LINQ. Washington: Redmond, 2007. pp. 1-195
- ✓ GRANADOS, Jimmy. Languaje Integrated query Framework. Neuronal Training Team, 2005, http://www.neuronaltraining.net/_Docs/LINQ-%20Project.pdf [2008/04/28]
- ✓ HERNÁNDEZ, Octavio. El Proyecto LINQ. MSDN- Microsoft. 2006, http://www.microsoft.com/spanish/msdn/articulos/archivo/041206/voices/LINQ_Project.mspx [2008/04/28]
- ✓ HERNÁNDEZ, Octavio. El Proyecto LINQ. MSDN, 2006, http://www.microsoft.com/spanish/msdn/articulos/archivo/041206/voices/LINQ_Project.mspx [2008/04/28]
- ✓ MONEGAL, Mariona. Introducción al spss, p. 104, http://books.google.com.ec/books?id=TqkWSd_88bIC&pg=PA107&lpg=PA107&dq=parametros+del++spss+10.0+%2B+mean+%2B+variance+%2B+Skewness&source=bl&ots=p57B_IXGE3&sig=6z_3IG4WPkGUCuNSDxnQAJK4fkw&hl=es&sa=X&oi=book_result&resnum=1&ct=result#PPA104,M1 [12/10/2008]
- ✓ SOM, Guillermo. LINQ y el acceso a datos con Visual Basic 2008, http://scribd.com/doc/5047163/Linq-y-el-acceso-a-datos-en-Visual-Basic-2008 [2008/04/28]

- ✓ PRESSMAN, Roger S. Ingeniería de software, Editorial: Mc Graw Hill, 1997. pp. 51-61
- ✓ MEIJER, Eric. SILVER, Amanda. Microsof Corporation, 2007, http://www.microsoft.com/spanish/msdn/articulos/archivo/020407/voices/ms364068.mspx [2008/10/5]
- ✓ MSDN. Microsof Corporation, Información general de Visual Studio 2008, 2007, http://msdn.microsoft.com/es-ec/vstudio/products/bb931331.aspx [2008/10/06]
- ✓ Microsoft Centro de descarga. Página del centro de descarga, http://www.microsoft.com/downloads/details.aspx?familyid=333325FD-AE52-4E35-B531-508D977D32A6&displaylang=es [2008/10/06]
- ✓ JIMENEZ, Alexander. Microsof Student Partner Lead – Venezuela, http://alexjimenez. com/2007/09/10/linq-definicion-como-usarlo/ [2008/10/06]
- ✓ GUTHRIE, Scout. LINQ, LINQ to SQL (1ra Parte), 2007, http://thinkingindotnet.wordpress.com/2007/05/20/usando-linq-to-sql-1%c2%aa-parte/ [2008/10/06]
- ✓ GUTHRIE, Scout. LINQ, Expresiones Lambda, 2007, http://thinkingindotnet.wordpress.com/author/vioswords/page/16/ [2008/10/06]