Facultad de Ingeniería.
Escuela de Ciencias y Sistemas.
Ing. José Ricardo Morales Prado.
Primer Semestre 2015.

Análisis y Diseño de Sistemas II

FASE I

PEDIDOS DE PIZZA A DOMICILIO.

CARNET	NOMBRES
201113845	Mydelin Stephanie Valladares Lima.
201047998	Irwin Guillermo Hernández Guerra.
200815284	Rainman Janixz Sián Chiroy.
200714982	Alvaro Obrayan Hernández Garcia.
200312755	Herminio Rolando García Sánchez.

METODOLOGÍA Y HERRAMIENTAS

1. METODOLOGÍA:

SCRUM:

Es una metodología ágil, por medio de la cual se sigue un marco de trabajo y desarrollo de software, basado en un proceso iterativo e incremental. En dicho proceso se debe de aplicar de manera regular un conjunto de buenas prácticas para trabajar colaborativamente en equipo y obtener el mejor resultado posible.

Los objetivos básicos son:

- Adoptar una estrategia de desarrollo incremental.
- Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados.

Beneficios:

- Cumplimiento de expectativas.
- Flexibilidad a cambios.
- Mayor calidad del software.
- Mayor productividad.
- Predicciones de tiempo

Como Funciona:

Primero que nada se debe definir el Product Backlog, ya que este nos permitirá administrar los Sprints.

1. **Product Backlog**: Es una lista sobre las funcionalidades del software. Es elaborado por el Product Owner y las funciones están priorizadas según lo que es más y menos importante para el negocio.

2. **Sprint Backlog:** Es un subconjunto de ítems del Product Backlog, que son seleccionados por el equipo para realizar durante el Sprint sobre el que se va a trabajar**.** El equipo establece la duración de cada Sprint.

3. **Sprint Planning Meeting:** Es una reunión que debe realizarse al principio de cada Sprint y se define cómo se va a enfocar el proyecto que viene del Product Backlog las etapas y los plazos.

4. **Daily Scrum Meeting:** Reunión breve que se realiza a diario mientras dura el periodo de Sprint. Se responden individualmente tres preguntas: ¿Qué hice ayer?, ¿Qué voy a hacer hoy?, ¿Qué ayuda necesito? El Scrum Master debe tratar de solucionar los problemas u obstáculos que se presenten.

5. **Sprint Review**: Se revisa el sprint terminado, y ya debería haber un avance claro y tangible para presentárselo al cliente.

6. **Sprint Retrospective:** El equipo revisa los objetivos cumplidos del Sprint terminado. Se anota lo bueno y lo malo, para no volver a repetir los errores. Esta etapa sirve para implementar mejoras desde el punto de vista del proceso del desarrollo.

Participantes:

- **Product Owner:** Habla por el cliente, y asegura que el equipo cumpla las expectativas. Es el responsable del proyecto**.**
- **Scrum Master:** Lidera las reuniones y ayuda al equipo si es que tienen problemas**.**
- **Scrum Team:** Son los encargados de desarrollar y cumplir lo que les asigna el Product Owner.
- **Cliente:** Recibe el producto y puede influir en el proceso, entregando sus ideas o comentarios respecto al desarrollo.

ROLES:

- **Project Manager:** Recoge, analiza y publica información sobre la marcha del proyecto, supervisa el cumplimiento de las estimaciones en cada iteración.
- **Analista Programador:** Son el equipo encargado de hacer código funcional del proyecto.

ROL	NOMBRES
Analista Programadora.	Mydelin Stephanie Valladares Lima.
Analista Programador.	Irwin Guillermo Hernández Guerra.
Analista Programador.	Rainman Janixz Sián Chiroy.
Project Manager.	Alvaro Obrayan Hernández Garcia.
Analista Programador.	Herminio Rolando García Sánchez.

PRODUCT BACKLOG

Es el equivalente a los requisitos del sistema o del usuario.

PRIORIDAD	PUNTOS	HISTORIAS	DESCRIPCIÓN
1	40	Montar BD y Plataformas a utilizar	Instalación de herramientas necesarias para el desarrollo del proyecto.
2	25	Creación del Login.	Interfaz de Inicio de sesión
3	45	Interfaz de Inicio	Creación de la interfaz de inicio de la página web con todas las opciones disponibles a la vista
4	45	Mantenimiento de Usuarios	Página que recibirá nuevos datos o editará los datos existentes de los usuarios (clientes)
5	50	Módulo de Pedidos	Módulo para registrar el pedido del usuario, registrando el total y los descuentos aplicables según promociones. (El modulo aparecerá en todas las páginas que contengan productos para ordenar)
6	30	Página Promociones	Página con contenido promocional para ordenar
7	45	Página Crear Pizza	Página con contenido ya preestablecido en la tabla de productos para armar pizzas.
8	35	Página Pizza	Página con contenido ya preestablecido en la tabla de productos de la categoría pizzas.
9	40	Página Editar Pizza	Página con todos los ingredientes de la pizza antes seleccionada donde el usuario puede deshabilitar si no desea alguno de estos ingredientes.
10	45	Página Adicionales	Página con contenido ya preestablecido en la tabla de productos de la categoría adicionales (productos adicionales como wings, calzzones o bebidas, entre otros).

PILA DEL SPRINT

Es el registro de los requisitos detallados o tareas que va a desarrollar el equipo técnico en la iteración (actual o que está preparándose para comenzar)

Responsabilidades del Scrum Manager

- Supervisión y asesoría en la elaboración de la pila de la pila del sprint.

Responsabilidades del equipo técnico

- Elaboración de la pila del sprint.
- Resolución de dudas o comunicación de sugerencias sobre las historias de usuario con el gestor del producto.

Fase 1

PRIORIDAD	PUNTOS	HISTORIAS	DESCRIPCIÓN
1	40	Montar Bd y Plataformas a Utilizar	Instalación de herramientas necesarias para el desarrollo del proyecto.
2	25	Creacion de Login	Interfaz de Inicio de sesión

Fase 2

PRIORIDAD	PUNTOS	HISTORIAS	DESCRIPCIÓN
3	45	Interfaz de Inicio	Creación de la interfaz de inicio de la página web con todas las opciones disponibles a la vista
4	40	Mantenimiento de Usuarios	Página que recibirá nuevos datos o editará los datos existentes de los usuarios (clientes)

Fase 3

PRIORIDAD	PUNTOS	HISTORIAS	DESCRIPCIÓN
5	30	Módulo de Pedidos	Módulo para registrar el pedido del usuario, registrando el total y los descuentos aplicables según promociones. (El modulo aparecerá en todas las páginas que contengan productos para ordenar)
6	25	Página Promociones	Página con contenido promocional para ordenar
7	35	Página Crear Pizza	Página con contenido ya preestablecido en la tabla de productos para armar pizzas.

Fase 4

PRIORIDAD	PUNTOS	HISTORIAS	DESCRIPCIÓN
8	35	Página Pizza	Página con contenido ya preestablecido en la tabla de productos de la categoría pizzas.
9	35	Página Editar Pizza	Página con todos los ingredientes de la pizza antes seleccionada donde el usuario puede deshabilitar si no desea alguno de estos ingredientes.
10	20	Página Adicionales	Página con contenido ya preestablecido en la tabla de productos de la categoría adicionales (productos adicionales como wings, calzzones o bebidas, entre otros).

GRÁFICA DE AVANCE (BURDOWN CHART)

Gráfico que muestra el estado de avance del trabajo del sprint en curso.

Responsabilidades del Scrum Manager

- Supervisión de la actualización diaria por parte del equipo.

Responsabilidades del equipo técnico

- Actualización diaria del gráfico de avance.

SPRINT PLANNING

Se trabajará los sprint planning el primer día hábil después de finalizada la fase anterior. Tomando en cuenta los puntos de desarrollo para definir los puntos de historia según la velocidad calculada de los integrantes del equipo, el product owner revisará las historias, en las cuales se incluirán errores a corregir de la fase anterior y se analizará cuales se podían realizar en la nueva iteración.

Se define un Punto de Historia de la siguiente manera: 1 punto de historia = 1 hora/semana de trabajo de un miembro del equipo en dedicación exclusiva. El tamaño del sprint será de 2 semanas.

Lista de miembros y nivel de dedicación:

Participantes	Horas*día (dedicación)	% Efectividad	Velocidad Horas*día
Mydelin Stephanie Valladares Lima.	2	0.75	1.5
Irwin Guillermo Hernández Guerra.	2	0.75	1.5
Rainman Janixz Sián Chiroy.	2	0.75	1.5
Alvaro Obrayan Hernández Garcia.	2	0.75	1.5
Herminio Rolando García Sánchez.	2	0.75	1.5
			7.5

Total de horas trabajadas a la semana: 9*5 = 37.5hrs, por lo que la velocidad del equipo es de 75 puntos por sprint.

REUNIÓN TÉCNICA DIARIA

Puesta en común diaria del equipo con presencia del Scrum Manager de duración máxima de 10 minutos.

Responsabilidades del Scrum Manager

- Supervisión de la reunión y anotación de las necesidades o impedimentos que pueda detectar el equipo.
- Gestión para la solución de las necesidades o impedimentos detectados por el equipo.

Responsabilidades del equipo técnico

- Comunicación individual del trabajo realizado el día anterior y el previsto para día actual.
- Actualización individual del trabajo pendiente.
- Actualización del gráfico de avance para reflejar el estado de avance.
- Notificación de necesidades o impedimentos previstos u ocurridos para realizar las tareas asignadas.

REUNIÓN DE CIERRE DE SPRINT Y ENTREGA DEL INCREMENTO.

Reunión para probar y entregar el incremento al gestor del producto.
Características.

- Prácticas: sobre el producto terminado (no sobre simulaciones o imágenes).
- De tiempo acotado máximo de 2 horas.

Responsabilidades del gestor de producto

- Asistencia a la reunión.
- Recepción del producto o presentación de reparos.

Responsabilidades del Scrum Manager

- Moderación de la reunión

Responsabilidades del equipo técnico

- Presentación del incremento.

2. HERRAMIENTAS:

CONTROL DE VERSIONES:

Es un Sistema que registra los cambios que realizamos a un archivo o varios archivos con el transcurso del tiempo, ya que por medio de esto se puede recuperar versiones específicas más adelante.
No solo código fuente se puede tener bajo el control de versiones, se puede cualquier tipo de archivo que encuentres en tu computadora poner en el control de versiones.

Sistemas de control de versiones centralizados:

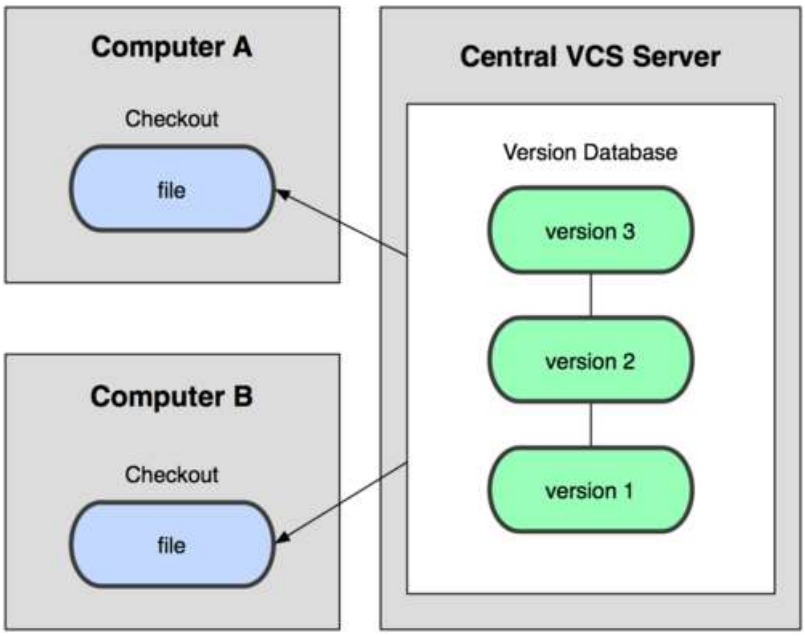
Sus siglas (CVCSs) Centralized Version Control Systems, estos sistemas Subversion y Perforce, tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos desde ese lugar central. Sin embargo tiene ventajas y desventajas.

VENTAJAS:

- Los administradores tienen el control detallado de que puede hacer cada uno.
- Se puede ver en que están trabajando los demás.

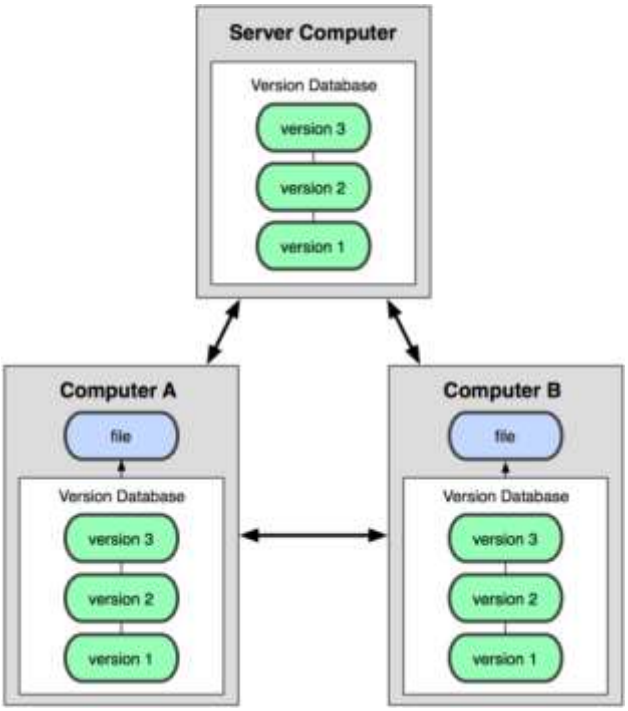
DESVENTAJAS:

- Si el servidor se cae, nadie puede hacer cambios versionados.
- Si el disco duro se corrompe y no se tiene copia de seguridad, esto se pierde, a menos que alguien tenga guardado en su máquina alguna versión.



Sistemas de control de versiones distribuidos:

DVCSs sus siglas (Distributed Version Control System), estos sistemas se puede decir que los clientes descargan la última versión de los archivos. La gran ventaja de este sistema es que si se cae, y estaban trabajando, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo.



Software para el manejo de versiones:

- Visual SourceSafe – herramienta de control de versiones de Microsoft; orientada a equipos pequeños.
- Visual Studio Team Foundation Server (anteriormente Team System) – Plataforma que automatiza el proceso de entrega del software y le da las herramientas que necesita para gestionar eficazmente los proyectos de desarrollo de software a través del ciclo de vida de IT.

SubVersion (SVN – Inspirado en CVS)

- **Mercurial**: Escrito en Python como un recambio en software libre de Bitkeeper; descentralizado, que pretende ser rápido, ligero, portable y fácil de usar.

INTEGRACIÓN CONTINUA:

Tiene como objetivo principal comprobar que cada actualización del código fuente no genere problemas que se está desarrollando.

Consiste en activar en cada integración un proceso basado en una plataforma que verifica automáticamente el funcionamiento de la aplicación para que las anomalías sean detectadas.

Aquí están algunas palabras que son de utilidad para la integración continua:

- **BUILD:** Es la creación de entregables al momento de ejecutar los test (FUNCIONAL, UNITARIOS).
- **COMMIT:** Es la operación que permite la validación de las actualizaciones del código en el directorio de trabajo local de la máquina del desarrollador/cliente por medio de la herramienta de gestión de configuración. Esto se hace desde el directorio de trabajo local hacia el referencial de la herramienta.
- **UPDATE:** Es la operación que permite la actualización a partir del referencial de la herramienta de gestión de configuración del directorio local.
- **CHECKOUT:** Es la operación de extracción de una versión que se está trabajando del administrador de configuración en un directorio de trabajo local.

Como Funciona:

El desarrollador hace un commit sobre el referencial del administrador de configuración.

El servidor de integración continua detecta el commit, hace un checkout ejecuta las operaciones de compilación y de test.
En caso de error se genera una notificación para el jefe del proyecto y/o al equipo de desarrollo.

El desarrollador que cometió el error hace un update del referencial de gestión de configuración y corrige el problema.

Servidores de Integración:

- **Cruise Control:** De código abierto y gratuito, muy conocido, mucha documentación, permite testear aplicaciones J2EE y aplicaciones en .Net. Es la referencia de la integración continua.

- **Hudson:** De código abierto y gratuito, muy popular, permite testear aplicaciones J2EE. Utilizado por SUN

- **Continuum:** De código abierto y gratuito sostenido por la fundación Apache

- **Bamboo:** De código abierto pero de pago.

- **Jenkins:** Es un servidor de integración continua, gratuito, open-source y actualmente uno de los más empleados para esta función. La base de Jenkins son las tareas, donde indicamos que es lo que hay que hacer en un BUILD. Se puede programar una tarea en la que se compruebe el repositorio de control de versiones cada cierto tiempo, y cuando un desarrollador quiera subir su código al control de versiones, este se compile y se ejecuten las pruebas.
  Si hay algún error, notifica al desarrollador y al equipo, ya sea por correo o por otro medio que especifiquemos. Si es correcto, entonces indicaremos a Jenkins que trate de integrar el código y subirlo al repositorio de versiones.

PRUEBAS UNITARIAS:

El objetivo de estas pruebas es detectar errores en los datos, lógica, algoritmos. Participan los programadores, Método a utilizar es el de la caja blanca.

Herramienta:

- **JUNIT:** Especialmente diseñada para implementar y automatizar la realización de pruebas de unidad en JAVA.
- **SIMPLE TEST:** Entorno de pruebas para aplicaciones realizadas en PHP
- **NUNIT:** Versión del framework para plataforma .NET

PRUEBAS FUNCIONALES AUTOMATIZADAS:

Es un proceso para procurar encontrar discrepancias entre las funcionalidades del programa y la especificación funcional.
El objetivo es detectar errores de interfaces y relaciones entre componentes. Los métodos a utilizar son: Caja Blanca (tipo de prueba de software que se realiza sobre las funciones internas de un módulo), Top Down (se formula un resumen del sistema, sin especificar detalles.), Bottom up.

Herramienta:

- **SELENIUM:** Es un conjunto de utilidades que facilita la labor de obtener juegos de pruebas para aplicaciones web. Para ello nos permite grabar, editar y depurar casos de prueba, que podrán ser ejecutados de forma automática e iterativa posteriormente.
  Además de ser una herramienta para registrar acciones, permite editarlas manualmente o crearlas desde cero. Las acciones se basan en el uso de diferentes API's en diferentes lenguajes (PHP, Ruby, JAVA, Javascript, etc).

- **WATIR:** Es una familia de librerías Ruby de Código Abierto (Open Source) para la automatización de navegadores web. Le permite a su usuario escribir pruebas fáciles de leer y mantener. Sencilla y flexible. Tiene la capacidad de hacer clic en enlaces, llenar formularios de pantallas con datos y presionar botones. Watir también revisa los resultados, incluyendo verificar si los textos esperados se muestran en las páginas.

- **WATIJ:** Es sinónimo de pruebas de aplicaciones web en JAVA.

PRUEBAS DE STRESS:

El objetivo es el comportamiento de una aplicación bajo una demanda excesiva. Es poder generar una gran cantidad de peticiones al software y verificar su comportamiento y de esta manera poder garantizar el número máximo de peticiones bajo las cuales el software puede trabajar.

Se deben de considerar varios aspectos:

- Debe realizarse a un ambiente parecido al de producción.
- Tener varios clientes realizando pruebas y estos deberán estar en diferentes computadoras.

Algo muy importante es que para poder realizar este tipo de pruebas deberá tenerse un plan bien definido de la arquitectura de servidores / computadoras que las realizarán, un método de recolección y análisis centralizado.

Herramienta:

- **JMETER:** Es una herramienta de carga para llevar a cabo simulaciones sobre cualquier recurso de Software. Diseñada para pruebas de stress en aplicaciones WEB, su arquitectura ha evolucionado no solo para llevar a cabo pruebas de stress en componentes habilitados en INTERNET (HTTP), base de datos, perl, Requisiones FTP.

- **JCRAWLER:** Aplicación Opensource para realizar test de estrés a aplicaciones web. Le pasas una URL y puedes realizar una navegación. Admite redirecciones HTTP y cookies. Es independiente de la plataforma, posee un modo consola y es sencillo de configurar.

    - Es apropiado para portales complejos en los que hay que testear todas las páginas del portal y no solo algunas URLs.
    - JCrawler está basado en ataques/segundo y no en "X" hilos atacando una web (ya que en estos últimos casos puede que realmente se estén dando por ejemplo solo 2 ataques por segundo aun teniendo 200 hilos).
    - Testeo de Http redirects y cookies.

- **THE GRINDER:** Ofrece una manera sencilla para los desarrolladores para realizar pruebas de carga antes de la entrega de la solicitud de aseguramiento de la calidad.

PLAN DE INTEGRACIÓN CONTINUA Y PRUEBA DE CONCEPTO

HERRAMIENTAS A UTILIZAR:

- **Control de Versiones-SVN:** Es una herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros.

- **Integración Continua-Jenkins:** Es un servidor gratuito, open-source y actualmente uno de los más empleados para esta función.

ESTRUCTURA DE DIRECTORIOS A UTILIZAR PARA ORGANIZAR EL PROYECTO:

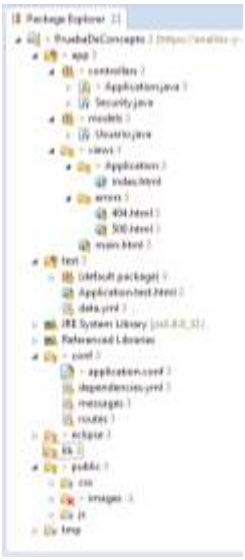
La estructura será de la siguiente manera, con fin de tener un trabajo ordenado.

- **CONTROLLERS**
  Empleado como un mediador entre el medio gráfico ("View") y el modelo ("Model"), coordina las acciones que son llevadas a cabo entre ambos.

- **MODELS**
  Concentra las funcionalidades relacionadas con el Modelo de datos, esto es, el acceso y manipulación de depósitos informativos como Bases de Datos y Archivos.

- **VIEW**
  Se basa en el aspecto visual/gráfico que será empleado por la aplicación en cuestión

- **TEST**
  En dicha carpeta se guardan las clases de test que se encargaran de probar el correcto funcionamiento de nuestra aplicación. Aquí por ejemplo podemos guardar nuestros test unitarios de JUnit.



ESQUEMA DE IDENTIFICACIÓN DE OBJETOS (TIPOS DE ARCHIVOS, NOMBRES, ESTÁNDARES)

Variables De Clase Static

Primero las variables de clase public, después protected, después las de nivel de paquete (Sin modificador de acceso), y después las private.

Declaración por variables, ya que facilita los comentarios.
También como alternativa utilizar tabulación

```
Int     palabra;
Int     tamano;
```

Para la inicialización de una variable local, se intente inicializar donde se declaran, cuando no se inicialicen es cuando el valor de la misma depende de algunos cálculos que deben ocurrir.
Poner las declaraciones solo al principio de los bloques (un bloque es cualquier código encerrado por llaves {}), no esperar al primer uso para declararlas. Ejemplo

```
Void metodo1()
  Int int1 = 0;
    If condicion1 {
      Int entero1 = 0;
    }
  }
```

Las Declaraciones De Clases E Interfaces

Se usan reglas para codificar clases e interfaces →
Ningún espacio en blanco entre el nombre de un método y el paréntesis.
La llave de apertura { aparece al final de la misma línea de la sentencia declaración.
La llave de cierre empieza una nueva línea, excepto cuando no existen sentencias entre ambas, debe aparecer inmediatamente después de la de apertura {

```
class Ejemplo extends Object {
int ivar1;
int ivar2;
Ejemplo(int i, int j) {
ivar1 = i;
ivar2 = j;
}
int metodoVacio() {}
}
```

Los métodos se separan con una línea en blanco
Para las sentencias se debe contener como mucho una sentencia.

```
Argv++;
Argc--;
```

Las sentencias compuestas son sentencias que contienen listas de sentencias encerradas entre llaves.
Las sentencias encerradas deben indentarse un nivel más que la sentencia compuesta.
Las llaves se usan en todas las sentencias, incluso las simples, cuando forman parte de una estructura de control, como las sentencias if-else o for. Esto hace más sencillo añadir sentencias sin incluir bugs accidentalmente por olvidar las llaves.
Una sentencia return con un valor no debe usar paréntesis a menos que hagan el valor de retorno mas obvio de alguna manera.
Return;
Return metoto.size();
Return (tamañpordefecto);

La sentencia if-else debe ser asi:

```
if (condicion) {
sentencias;
}
if (condicion) {
sentencias;
} else {
sentencias;
}
```

La sentencia for debe de ir asi:

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

La sentencia While:

```
while (condicion) {
sentencias;
}
```

La sentencia Switch:

```
switch (condicion) {
case ABC:
sentencias;
/* este caso se propaga */
case DEF:
sentencias;
break;
case XYZ:
sentencias;
break;
default:
sentencias;
break;
}
```

La sentencia Try-Catch

```
try {
sentencias;
}
catch (ExceptionClass e) {
sentencias;
} finally {
sentencias;
}
```

NOMBRE DE CLASES:

Los nombres de las clases deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases simples y descriptivas. Usar palabras completas, evitar acrónimos y abreviaturas.

NOMBRE DE INTERFACES:

Los nombres de las interfaces siguen la misma regla que las clases.

NOMBRE DE METODOS:

Los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

NOMBRE DE VARIABLES:

Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas. Los nombres de variables no deben empezar con los caracteres subguión "_" o signo del dolar "$", aunque ambos estan permitidos por el lenguaje.
Los nombres de las variables deben ser cortos pero con significado

NOMBRE DE CONSTANTES:

Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un subguión ("_"). (Las constantes ANSI se deben evitar).

ASIGNACION DE VARIABLES:

Evitar asignar el mismo valor a varias variables en la misma sentencia
No usar el operador de asignación en un lugar donde se pueda confundir con el de igualdad.

ESQUEMA DE NOMENCLATURA DE VERSIONES:

Para distinguir entre las diversas versiones se utilizara la siguiente nomenclatura:

X.Y.Z

Funcionalidad.Mejora.Error

- **Funcionalidad:** Se refiere a los cambios que pueda tener el proyecto en tanto a las funcionalidades.
- **Mejora:** Se refiere a las mejoras que cada función alcance.
- **Error:** Este valor cambia cuando se ajustó un desliz en el software.

PROCESO DE LIBERACION Y MAPEO A HERRAMIENTAS UTILIZADAS

Integración Continua (Jenkins): Jenkins es un software de Integración continua open source escrito en Java. Está basado en el proyecto Hudson y es, dependiendo de la visión, un fork del proyecto o simplemente un cambio de nombre.

Control de versionamiento: Subversion (SVN) es una herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros. Es software libre bajo una licencia de tipo Apache/BSD.
Utiliza el concepto de revisión para guardar los cambios producidos en el repositorio. Entre dos revisiones sólo guarda el conjunto de modificaciones (delta), optimizando así al máximo el uso de espacio en disco. SVN permite al usuario crear, copiar y borrar carpetas con la misma flexibilidad con la que lo haría si estuviese en su disco duro local. Dada su flexibilidad, es necesaria la aplicación de buenas prácticas para llevar a cabo una correcta gestión de las versiones del software generado.

Framework: Play es un framework de aplicaciones web de código abierto, escrito en Java y Scala, que sigue la-vista-controlador (MVC) patrón arquitectónico. Su objetivo es optimizar la productividad del desarrollador utilizando convención sobre configuración, código caliente recarga y visualización de errores en el navegador.
Soporte para el lenguaje de programación Scala ha estado disponible desde la versión 1.1 de las Directrices. En la versión 2.0, el núcleo marco fue reescrito en Scala. Construir y el despliegue se ha migrado a SBT, y utilizar plantillas Scala en lugar de Groovy.

PRUEBA DE CONCEPTO:

RESUMEN DE REQUERIMIENTOS

El proyecto consiste en realizar un software de aplicación online que será utilizado para realizar pedidos de productos alimenticios en línea para entrega a domicilio, la aplicación contara con un registro de usuarios en el cual se deberán de almacenar los datos personales del usuario tales como, el nombre, apellido, dirección de entrega, el número de nit para la factura entre otros, también contara con una interfaz de login, y una de selección de producto así como de ingredientes adicionales que desee agregarle a su orden, la aplicación deberá ser capaz de almacenar un historial de compras realizadas anteriormente, como un historial de entregas fallidas y entregas exitosas, esta aplicación deberá ser accesible desde cualquier tipo de navegador, la aplicación deberá también ser capaz de realizar el cálculo del total a cancelar el cual será cancelado cuando se entregue el producto en el domicilio y en efectivo, la aplicación no realizara cobros de ningún tipo, también dentro de la aplicación deberemos contar con una interfaz que nos permita realizar cambios dentro del perfil del usuarios y mantener actualizados sus datos dentro de la aplicación, la aplicación contara con una interfaz amigable que sea sencilla de comprender para los usuarios.

Los requerimientos básicamente son:

1. Proceso de Administración de Clientes y Empleados

Registro: Para crear o actualizar un cliente se debe tener en cuenta la siguiente información:

- Número de documento (*) (único)
- Nombre completo del cliente(*)
- Teléfono
- Dirección
- Email (*)
- NIT

Consultas: Se deben poder consultar los usuarios registrados. La consulta debe presentar la información básica de los clientes.

Modificación: Se pueden modificar todos los datos de un usuario. Para modificarlo se debe permitir hacer búsqueda por código o nombre.

Eliminación: Para eliminar usuarios, el programa debe permitir la búsqueda por número de documento, el resultado de esta búsqueda se debe mostrar en una tabla con las columnas: número de documento, nombre y email. Se debe poder seleccionar el cliente que se desea borrar, antes de ser eliminado se debe mostrar un mensaje de confirmación.

2. Proceso de Administración de Menús y Pedidos

Este proceso incluye creación, actualización, consulta y eliminación de menús o pedidos.

Creación: Para crear un menú o pedido se debe incluir la siguiente información:

- MENU: Debe contener una serie de productos relacionados y un código para identificarlo.
- PEDIDO: Contiene una seria de menús y/o productos, la fecha, el estado, forma de pago y el NIT o Código del cliente.

Consulta:

- MENU: Se podrán consultar según el código del menú.
- PEDIDO: Se podrán consultar según el código del cliente y fecha.

Modificación y Eliminación:

- MENU: Se pueden modificar/eliminar los datos de un menú. Para realizar esto se debe permitir hacer búsqueda por código.
- PEDIDO: Se podrán modificar/eliminar según el código del cliente y fecha.

3. Proceso de Administración de Cola de Espera

- Según se vayan realizando los pedidos, se ingresarán a una cola.
- Una vez se haya entregado el pedido, el empleado podrá poner el pedido con estado de ENTREGADO.
- Una vez terminado el pedido se eliminara de la cola de espera.

4. Reportes

- Reporte de Ventas por producto.
- Reporte de Pagos por forma de pago.
- Reporte del catálogo de Menú.
- Reporte del catálogo de Usuarios.

5. Otros

- La aplicación deberá que responder en un máximo de 5min.
- Tendrá que ser segura, lo cual se verificara con el login.
- Deberá soportar varios clientes concurrentemente, lo cual se comprobara a través de las pruebas de stress.

PLAN DE RELEASES

La administración de las historias del proyecto se organizará de la siguiente manera: